HAW HAMBURG

BACHELOR THESIS
Marion Becker

Entwicklung einer verteilten Anwendung zum zeitoptimierten Download von Darknet-Daten

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Marion Becker

Entwicklung einer verteilten Anwendung zum zeitoptimierten Download von Darknet-Daten

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Eingereicht am: 07. Juni 2024

Marion Becker

Thema der Arbeit

Entwicklung einer verteilten Anwendung zum zeitoptimierten Download von Darknet-Daten

Stichworte

Darknet, Download, Verteilte Systeme

Kurzzusammenfassung

In dieser Arbeit wird ein verteiltes Downloadsystem für das Darknet vorgestellt, das die Downloadgeschwindigkeit durch parallele Datenübertragungen optimiert. Das Hauptproblem, das gelöst werden soll, ist die geringe Bandbreite und Geschwindigkeit bei Downloads von Onion Services im Tor-Netzwerk. Ziel der Arbeit ist es, ein Programm zu entwickeln, das Downloads auf mehrere Clients verteilt, um die Effizienz zu steigern.
Das entwickelte System besteht aus drei Hauptkomponenten: einem Client-Programm, das die Daten herunterlädt, einem Koordinator-Programm, das die Aufgabenverteilung steuert, und einem Speicherprogramm, das die heruntergeladenen Daten zentral speichert. Die Kommunikation zwischen diesen Komponenten erfolgt über Java Sockets und JSON-Nachrichten, um eine einfache und plattformunabhängige Datenübertragung zu gewährleisten.
Die Evaluierung des Systems zeigt, dass verteilte Downloads die Effizienz im Vergleich zu herkömmlichen Downloadmethoden erheblich verbessern können. Allerdings gibt es noch Optimierungspotenzial in Bezug auf die Kommunikationsmethoden und die automatische Anpassung der Client-Anzahl.
Zukünftige Forschung könnte sich auf die Leistungsanalyse und die Auswirkungen verteilter Downloads auf Quellserver und Tor-Relays konzentrieren. Weitere Verbesserungen könnten die Entwicklung eines Peer-to-Peer-Netzwerks und die Implementierung dynamischer Anpassungsmechanismen umfassen.

Marion Becker

Title of Thesis

Keywords

Darknet, Donwload, Distributed Applications

Abstract

This thesis presents a distributed download system for the Darknet that optimizes download speed through parallel data transfers. The main problem addressed is the low bandwidth and speed when downloading from Onion Services in the Tor network. The goal of the project is to develop a program that distributes downloads across multiple clients to increase efficiency.

The developed system consists of three main components: a client program that downloads the data, a coordinator program that manages task distribution, and a storage program that centrally stores the downloaded data. Communication between these components is carried out using Java Sockets and JSON messages to ensure simple and platform-independent data transfer.

The evaluation of the system shows that distributed downloads can significantly improve efficiency compared to conventional download methods. However, there is still optimization potential in terms of communication methods and the automatic adjustment of the number of clients.

Future research could focus on performance analysis and the impact of distributed downloads on source servers and Tor relays. Further improvements could include the development of a peer-to-peer network and the implementation of dynamic adjustment mechanisms.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

1 Einleitung **1**
1.1 Inspiration 1
1.2 Zielsetzung der Arbeit 2

2 Grundlagen **3**
2.1 Grundlagen des Datenverkehrs 3
2.2 Bereitstellung von Daten im Netz 3
2.3 Herunterladen von Daten 4
2.4 Das Tor Netzwerk 4
2.5 Hidden Services 5
2.6 Bottlenecks auf dem Weg vom Client zu den Hidden Services 7
2.6.1 Client-Endpunkt 8
2.6.2 Entry Guard - Einstiegsknoten 8
2.6.3 Relays - Middle Knoten 8
2.6.4 Hidden Service Directories 9
2.6.5 Introduction Point 9
2.6.6 Rendezvous Point 9
2.6.7 Hidden Server 9
2.6.8 Zusammenfassung Bottlenecks 9
2.7 Methoden/Grundlagen der Zeitoptimierung 11
2.7.1 Einleitung 11
2.7.2 Netzwerk Auslastung - Wann ist eine gute Zeit 11
2.7.3 CDN 12
2.7.4 Paralleliserung 13
2.7.5 Parallelelisierung von Downloads 14
2.7.6 Zusammenfassung zeitoptimierender Methoden 17

3 Untersuchung von existierenden Tools **18**
3.1 Wie die Downloads aus dem Darknet derzeit durchgeführt werden 18
3.2 Tool Internet Download Manager 18
3.3 Erkenntnis 20

4 Situationsanalyse **21**
4.1 Situation 21
4.2 Lösungsansätze 21

5 Das Werkzeug zum Verteiltem Download **22**
5.1 Einführung 22
5.2 Anforderung-Analyse 22
5.2.1 Zielsetzung 23
5.2.2 Umfang 23
5.2.3 Stakeholder Analyse 23
5.2.4 Funktionale Anforderungen 23
5.2.5 Nicht-funktionale Anforderungen 24
5.2.6 Anwendungsfälle (USE Cases) 25
5.3 Das System an sich - Design 29
5.3.1 Einleitung 29
5.3.2 Programm-Architektur 30
5.3.3 Koordinator Architektur 30
5.3.4 Client Architektur 32
5.3.5 Speicher Architektur 33
5.3.6 Bibliotheks Funktionen 35
5.3.7 Die Nachrichten Messages 35
5.3.8 Nachrichten verschicken 35
5.4 Abläufe bei verschiedenen Tätigkeiten im System 35
5.4.1 Initialisierung 36
5.4.2 Laufender Betrieb 37
5.4.3 Das Ende 39
5.5 Das System Umsetzung Anmerkungen 40
5.5.1 Einbindung des Tor Cleints 41
5.5.2 Kommunikation zwischen den Komponenten 41
5.5.3 Ausführen des Programms 42
5.6 Auswertung 44

6 Fazit **47**
6.1 Ausbau Möglichkeiten des Tools 48
6.2 Mögliche weiter Forschungsansätze 48

Literaturverzeichnis **50**

A Anhang **52**
Selbstständigkeitserklärung 59

Abbildungsverzeichnis

2.1 Eine einfache Darstellung der Verbindung von einem Client zu einem Hidden Server 10
2.2 Ein Parallel Download einfach Dargestellt 16
2.3 Ein Concurrent Download einfach Dargestellt 16

5.1 Eine Übersicht über die Koordinator Komponente 31
5.2 Ein Diagramm über die Klasse Client 32
5.3 UML Diagramm des Speichers 34
5.4 Sequenz Diagramm zum Aufbau des Systems 37
5.5 Sequenz Diagramm zum Arbeiten des Systems 39
5.6 Sequenz Diagramm zum Abbau des Systems 40
5.7 UML-Diagramm mit allen wesentlichen Klassen 41
5.8 Abbildung der Messung eines Downloads mit unterschiedlich vielen Clients. 45
5.9 Effizienz der Netzwerkverbindungen im Download. Drei Messreihen. Normales Internet, Verbindung über Tor zu einem normalen Server und Verbindung über Tor zu einem Hidden Server. 46

A.1 Sequenz Diagramm für den Aufbau des Systems 52
A.2 Sequenz Diagramm für den betrieb des Systems 53
A.3 Sequenz Diagramm für den Abbau des Systems 54
A.4 Ein Überblick über alle Klassen im System 55
A.5 Einen Überblick über die Client Komponente 56
A.6 Ein Überblick über die Speicher Komponente 57
A.7 Ein Überblick über die Koordinator Komponente des Systems 58

1 Einleitung

1.1 Inspiration

Die Herausforderung der Internetkriminalität begleitet das Aufkommen des allgemein zugänglichen Internets von Anfang an. Heutzutage stellt sich nicht mehr die Frage, ob Systeme von Kriminellen angegriffen werden, sondern vielmehr, ob diesen Angriffen erfolgreich begegnet werden kann. Leider gelingt es den Angreifern häufig, ihre Ziele zu erreichen.

Ein konkretes Beispiel für die weitreichenden Auswirkungen solcher Angriffe ereignete sich am 29. Dezember 2022, als die Hochschule für Angewandte Wissenschaften (HAW) Opfer eines erfolgreichen Cyberangriffs wurde, wie aus einem Bericht [1] hervorgeht. Als Folge dieser Attacke musste die HAW ihre gesamte Systeminfrastruktur wiederherstellen, da eine beträchtliche Menge sensibler Daten von den Angreifern gestohlen und im Darknet veröffentlicht wurde.

Die Inspiration für diese Bachlorarbeit ist genau disere Erfolgreiche Attacke von Angreifern auf die HAW. Dabei sind nämlich vile Daten von den Angreifern gestohlen und dann im Dark Net veröffentlicht worden. Und das Herunterladen dieser Daten hat der HAW mehrer Wochen gedauert. Dadurch kommt es zu dieser Arbeit die die Entwicklung eines Tool zu einem zeitoptimierten Download aus dem Dark Net anstrebt.

Daraus ergibt sich das zentrale Thema dieser Arbeit: die Entwicklung einer Anwendung zur zeitoptimierten Extraktion von Daten von einem Onion-Server im Darknet.

1.2 Zielsetzung der Arbeit

Das Hauptziel dieser Arbeit ist die Entwicklung einer Anwendung, die es ermöglicht, Downloads aus dem Darknet zeitoptimiert durchzuführen. Um dieses Ziel zu erreichen, werden zunächst die Strukturen und Mechanismen des Darknets analysiert, um ein tiefgreifendes Verständnis für die Funktionsweise dieses Netzwerks zu erlangen. Anschließend werden verschiedene Strategien zur Optimierung von Downloads untersucht, darunter Peer-to-Peer-Netzwerke, die Implementierung effizienter Routing-Algorithmen und die Analyse optimierter Datenübertragungsprotokolle. Basierend auf diesen Erkenntnissen wird eine individuelle Lösung entwickelt und präsentiert, die eine zeitoptimierte Datenextraktion aus dem Darknet ermöglicht.

2 Grundlagen

In diesen Kapitel werden die Grundlagen erklärt und Definiert. Wir werden kurz und bündig das Internet anschauen und wie man Daten herunterlädt und uns dann dem "Tor Netzwerkßuwenden. Auch werden wir kurz auf die Geschichte von Hidden Services und ihrer Bedeutung für das Tor Netzwerk eingehen

2.1 Grundlagen des Datenverkehrs

Das Routing von Internet-Verbindungen spielt eine entscheidende Rolle beim Daten herunterladen. Wenn ein Client eine Anfrage sendet, muss das Routing die effizienteste Route bestimmen, um die Daten vom Server zum Client zu übertragen. Dieser Prozess erfolgt über verschiedene Netzwerk-knoten und Router entlang des Pfades, wobei das Routing-Protokoll die optimale Route basierend auf verschiedenen Faktoren wie Latenzzeit, Bandbreite und Netzwerk-Auslastung auswählt.

2.2 Bereitstellung von Daten im Netz

Daten, die im Netz zur Verfügung gestellt werden, befinden sich auf einem Server, der über das Netz erreichbar sein muss. Um den richtigen Server zu erreichen, von dem Daten heruntergeladen werden sollen, muss die Adresse dieses Servers bekannt sein. Die Auflösung des Namens der Adresse erfolgt durch den DNS-Server (Domain Name Server). Die bereitgestellten Daten befinden sich in der Regel in einem Verzeichnis, das über das Web zugänglich ist. Häufig wird das Hypertext Transfer Protocol (HTTP) verwendet, um diese Daten zwischen Server und Client zu übertragen. HTTP ist ein zustandsloses Protokoll, das die Anforderung und Bereitstellung von Text-, Bild-, Video- oder Audiodateien ermöglicht.

2.3 Herunterladen von Daten

Das Herunterladen von Daten aus dem Internet, genauer gesagt von einem Server, ist ein Prozess, der wie folgt beschrieben werden kann. Beim Herunterladen werden Daten von einem entfernten System bezogen [17]. Ein Client sendet eine Anfrage an einen Server, und über eine TCP-Verbindung werden die angeforderten Daten vom Server an den Client übertragen. Im Gegensatz zum Streamen, das UDP verwendet, setzt das Herunterladen auf TCP, da es eine reihenfolgesichere Übertragung gewährleistet. Bei der Verwendung von UDP können Pakete sich gegenseitig überholen, was beispielsweise bei Voice-over-IP keine große Rolle spielt. In solchen Fällen nimmt man lieber den Verlust von Paketen in Kauf, anstatt eine dauerhaft verzögerte Verbindung zu haben, bei der auf das erste Paket für den nächsten Abschnitt gewartet werden muss.

2.4 Das Tor Netzwerk

Die Idee eines Onion Routings, um anonym im Netz zu sein, begann bereits Mitte der 90er Jahre. Und das Tor Netzwerk ist nicht das erste Projekt um das Anonymisieren von Benutzern im Internet zu erreichen.

Das Tor-Netzwerk[7] ist ein dezentrales Netzwerk von Servern, das Benutzern ermöglicht, im Internet anonym zu sein. Es handelt sich um ein Overlay-Netzwerk, das bedeutet, es besteht über der Internet Struktur, insbesondere dem World Wide Web. Die Anonymität wird durch die Onion-Routing-Technologie [7] erreicht, bei der der Datenverkehr über mehrere Knoten geleitet wird, um die Identität und die Spuren des Benutzers zu verschleiern.

Tor ist ziemlich groß mit über 7000 Relays wie Tor über sich selbst aussagt [3]. Etwa 4000 davon melden sich als sind Guard Knoten.

Tor ist ein sogenanntes Low-Latency-Netzwerk, was bedeutet, dass es eine geringe Zeitverzögerung im Netzwerk hat, damit die Benutzer normal mit dem Internet interagieren können. Ein einfaches Design für eine niedrige Latenz ist ein sogenannter Anonymizer, ein einzelner Server, der die Daten von ihrer ursprünglichen Adresse befreit, bevor er sie

an das Ziel weiterleitet. Da es jedoch nur einen Server gibt, kann der Datenverkehr trotz Anonymisierung verfolgt werden, indem Verkehrsdaten wie Zeit und Dauer zurückverfolgt werden.

Da das Tor-Netzwerk ein dezentrales Netzwerk[2] ist, gibt es nicht nur eine Route, die der Verkehr durch das Netzwerk nehmen kann. Viele verschiedene Organisationen stellen Relays zur Verfügung. Jeder Benutzer benötigt einen Onion-Proxy, um das Tor-Netzwerk zu nutzen. Dies ist die Software, die den Zugriff auf das Tor-Netzwerk ermöglicht.

Im Tor-Netzwerk werden Nachrichten in Zellen übermittelt. Es gibt Relay-Zellen und Steuerungs-Zellen. Relay-Zellen bringen die Daten von Einstiegs- zu Ausstiegsknoten, während Steuerungs-Zellen dazu dienen, den Circuit im Tor-Netzwerk zu steuern. Ein Circuit im Tor-Netzwerk besteht in der Regel aus drei Knoten, über die der Verkehr geleitet wird.

Als Verteidigungsmaßnahme gegen Angreifer, die sowohl den ersten als auch den letzten Knoten in einem Circuit kontrollieren, wurden die sogenannten Entry Guards in "Defending Anonymous Communication Against Passive Logging Attack" von Wright[21] für das Tor Netzwerk eingeführt.

Der Onion-Proxy wählt aus einer Liste von drei Knoten, die als seine Entry Guards fungieren, und beginnt seinen Circuit immer nur mit diesem. Dadurch wird das Entdeckungsrisiko für jeden Client von N/N auf $3/N$ reduziert. Dabei nimmt der Client zwar auch neue Entry Guards in seine Liste auf aber er versucht diese möglichst klein zu halten.
Der Onion-Proxy versucht etwa alle Minute, einen neuen Circuit aufzubauen, um die Anonymität bestmöglich zu gewährleisten.

2.5 Hidden Services

Hidden Services oder zu deutsch Versteckte Dienste sind genau das was ihr Name sagt. Internet Dienste, die sich verstecken, deren IP Adresse dem Nutzer also nicht bekannt

ist. So kann niemand diese Dienste im normalen Internet finden und auch niemand herausfinden unter welcher IP Adresse sie agieren.
Hidden Services, auch bekannt als Onion-Services, sind eine bedeutende Komponente des Tor-Netzwerks, die es Benutzern ermöglicht, anonyme Dienste im Internet anzubieten und zu nutzen, ohne dabei ihre Identität preiszugeben. Hidden Services wurden erstmals im Jahr 2004 von der Electronic Frontier Foundation (EFF) eingeführt, um Benutzern eine sichere und anonyme Möglichkeit zu bieten, im Internet zu kommunizieren und Dienste bereitzustellen.

Die Idee der Hidden Services entstand aus dem Bedürfnis nach einer sicheren und anonymen Möglichkeit, im Internet zu agieren, insbesondere in Zeiten zunehmender Überwachung und Zensur. Durch die Nutzung des Tor-Netzwerks konnten Benutzer ihre Identität und Standorte verschleiern und gleichzeitig auf eine Vielzahl von Diensten zugreifen, die im regulären Internet möglicherweise nicht verfügbar waren.

Hidden Services wurden schnell zu einem integralen Bestandteil des Tor-Netzwerks und erlebten eine rasche Entwicklung und Verbreitung, da sie eine Vielzahl von Anwendungen und Möglichkeiten für Benutzer und Dienstanbieter boten. Von anonymen Foren und sozialen Netzwerken bis hin zu Marktplätzen für illegale Waren und Dienstleistungen, Hidden Services wurden für eine Vielzahl von Zwecken genutzt, sowohl legal als auch illegal.

Die Einführung von Hidden Services stärkte das Tor-Netzwerk und trug maßgeblich zur Popularität und Verbreitung bei. Indem sie Benutzern eine sichere und anonyme Möglichkeit boten, Dienste im Internet anzubieten und zu nutzen, erweiterten Hidden Services den Anwendungsbereich des Tor-Netzwerks erheblich und machten es zu einem unverzichtbaren Werkzeug für diejenigen, die Wert auf Privatsphäre und Anonymität legen.

Hidden Services spielten auch eine wichtige Rolle bei der Förderung von Meinungsfreiheit und Informationsfreiheit im Internet, insbesondere in Regionen mit eingeschränkter Internetfreiheit und Zensur. Indem sie es Benutzern ermöglichten, anonyme Kommunikationsplattformen und Informationsquellen bereitzustellen und zu nutzen, trugen Hidden Services dazu bei, die Grenzen der freien Meinungsäußerung im digitalen Raum zu erweitern.

Im Jahr 2006 wurden bedeutende Erweiterungen für Hidden Services eingeführt, die in "Locating Hidden Servers" von Syverson und Øverlier[16] vorgestellt worden, den Entwicklern und Forschern hinter dem Tor-Projekt. Diese Erweiterungen sollten den Schutz und die Sicherheit von Hidden Services weiter verbessern und das Tor-Netzwerk insgesamt robuster machen.

Eine der wichtigsten Erweiterungen war die Einführung von sogenannten Hidden-Service-Direktoren. Diese Hidden-Service-Direktoren fungieren als spezielle Knotenpunkte im Tor-Netzwerk, die den Datenverkehr zu Hidden Services leiten und dabei zusätzliche Sicherheitsmaßnahmen implementieren. Durch die Verwendung von Hidden-Service-Direktoren wurde die Effizienz und Sicherheit der Kommunikation mit Hidden Services verbessert, indem potenzielle Angriffe und Überwachung erschwert wurden.

Zusätzlich zu den Hidden-Service-Direktoren wurden auch weitere Verbesserungen im Bereich der Verschlüsselung und Authentifizierung eingeführt, um die Integrität und Vertraulichkeit von Hidden Services weiter zu stärken. Diese Verbesserungen umfassten die Implementierung von stärkeren Verschlüsselungsalgorithmen und die Verwendung von Authentifizierungsmechanismen wie dem Hidden-Service-Identitätsschlüssel, um die Echtheit von Hidden Services zu gewährleisten und Angriffe durch falsche Dienste zu verhindern.

Insgesamt trugen die Erweiterungen im Jahr 2006 erheblich zur Verbesserung der Sicherheit und Zuverlässigkeit von Hidden Services im Tor-Netzwerk bei. Sie machten das Tor-Netzwerk zu einem noch robusteren und vertrauenswürdigerem Werkzeug für diejenigen, die Wert auf Anonymität und Privatsphäre legen, und unterstrichen die kontinuierlichen Bemühungen des Tor-Projekts, die Sicherheit und Privatsphäre seiner Benutzer zu schützen.

2.6 Bottlenecks auf dem Weg vom Client zu den Hidden Services

Bottlenecks in Netzwerken sind die Stellen, die die Geschwindigkeit des Netzwerks bestimmen. Da das Tor-Netzwerk ein Overlay-Netzwerk ist, das auf dem normalen Internet aufbaut, kann es verschiedene Engpässe geben, die die Geschwindigkeit beeinflussen.

Der Weg vom Client zu den Hidden Services verläuft über eine Reihe von verschiedenen Teilnehmern, wie in Abbildung 2.1 dargestellt. Im Folgenden werden wir diese speziellen Teilnehmer und ihre Möglichkeiten, als Bottleneck zu agieren, besprechen. Wir betrachten einige mögliche Engpässe im Onion-Netzwerk und bewerten sie danach, was man tun kann, um diese zu umgehen.

2.6.1 Client-Endpunkt

Der Client ist der erste Punkt in einer Verbindung zu den Hidden Services. Der Client, der den Download durchführt, kann auch ein Bottleneck sein. Wenn unser Client das langsamste Mitglied in der Verbindung zu den Hidden Services ist, sind das gute Nachrichten. Unser eigenes System können wir ändern und kontrollieren, um eine bessere Verbindung zu schaffen.

Falls es unsere Hardware ist, die den Download ausbremst, kann man die Hardware verbessern. Wenn unsere eigene Firewall das Problem ist, kann man diese entsprechend konfigurieren.

2.6.2 Entry Guard - Einstiegsknoten

Der Entry Guard ist der erste Knotenpunkt im Tor-Netzwerk. Beim Start des Tor-Clients wird ein Entry Node ausgewählt und während der Sitzung nur im Falle eines Ausfalls gewechselt. Dies ist ein Sicherheitsmechanismus, und den Entry Guard zu wechseln ist nicht trivial. Dafür muss man den Tor-Client schließen, seine Liste an ausgewählten Entry Nodes löschen und ihn sich dann neu aufbauen lassen.

Allerdings hat jeder Client seine eigene Liste an Entry Guards.

2.6.3 Relays - Middle Knoten

Die mittleren Knoten, oder auch Relay-Knoten, im Tor-Netzwerk werden regelmäßig gewechselt. Auch wenn ein Relay-Knoten für eine gewisse Zeit die Verbindung drosselt, ist es nur eine Frage der Zeit, bis der Tor-Client einen neuen Relay-Knoten auswählt. Letztendlich gibt es keine Möglichkeit, den Middle Knoten zu beeinflussen, daher ist es zufallsabhängig, ob einer dieser Knoten ein Bottleneck darstellt.

2.6.4 Hidden Service Directories

Die Hidden Service Directories sind für die Hidden Services das, was DNS für normale Webseiten ist. Anstatt IP-Adressen geben sie Introduction-Server für ihren Namen heraus, über die man sie kontaktieren kann. Fällt einer dieser Server aus, kann dies die Verbindung verzögern oder sogar verhindern.

2.6.5 Introduction Point

Der Introduction Point stellt eine Verbindung zwischen Client und Hidden Service her, indem er den Rendezvous Point herausgibt. Er ist lediglich ein Vermittlungspunkt und kommt als Bottleneck nicht wirklich in Frage, da über ihn keine großen Datenmengen laufen.

2.6.6 Rendezvous Point

Der Rendezvous Point ist der Punkt, über den die Verbindung dann tatsächlich läuft. Auch dieser kann bei jedem Neuaufbau der Verbindung neu gewählt werden.

2.6.7 Hidden Server

Der Hidden Service selbst kann natürlich auch ein Bottleneck sein. Wenn der Quellserver der Daten das langsamste Glied in der Kette ist, gibt es nichts, was man tun kann. Jeder Hidden Server hat auch seinen eigenen Entry Guard, den wir ebenfalls nicht beeinflussen können.

2.6.8 Zusammenfassung Bottlenecks

Insgesamt geht die Verbindung von einem Client zu einem Hidden Server über mehrere Knoten: den Entry Node des Clients, die festgelegte Anzahl an Middle Nodes im Tor-Netzwerk, den Rendezvous Point, den Entry Guard des Hidden Servers und schließlich zum Hidden Server selbst (siehe Abbildung 2.1).

Da der Tor-Client alle 10 Minuten einen neuen Circuit aufbaut [4], ändern sich die Relay Nodes ständig, sodass man nach 10 Minuten ganz andere Übertragungsgeschwindigkeiten

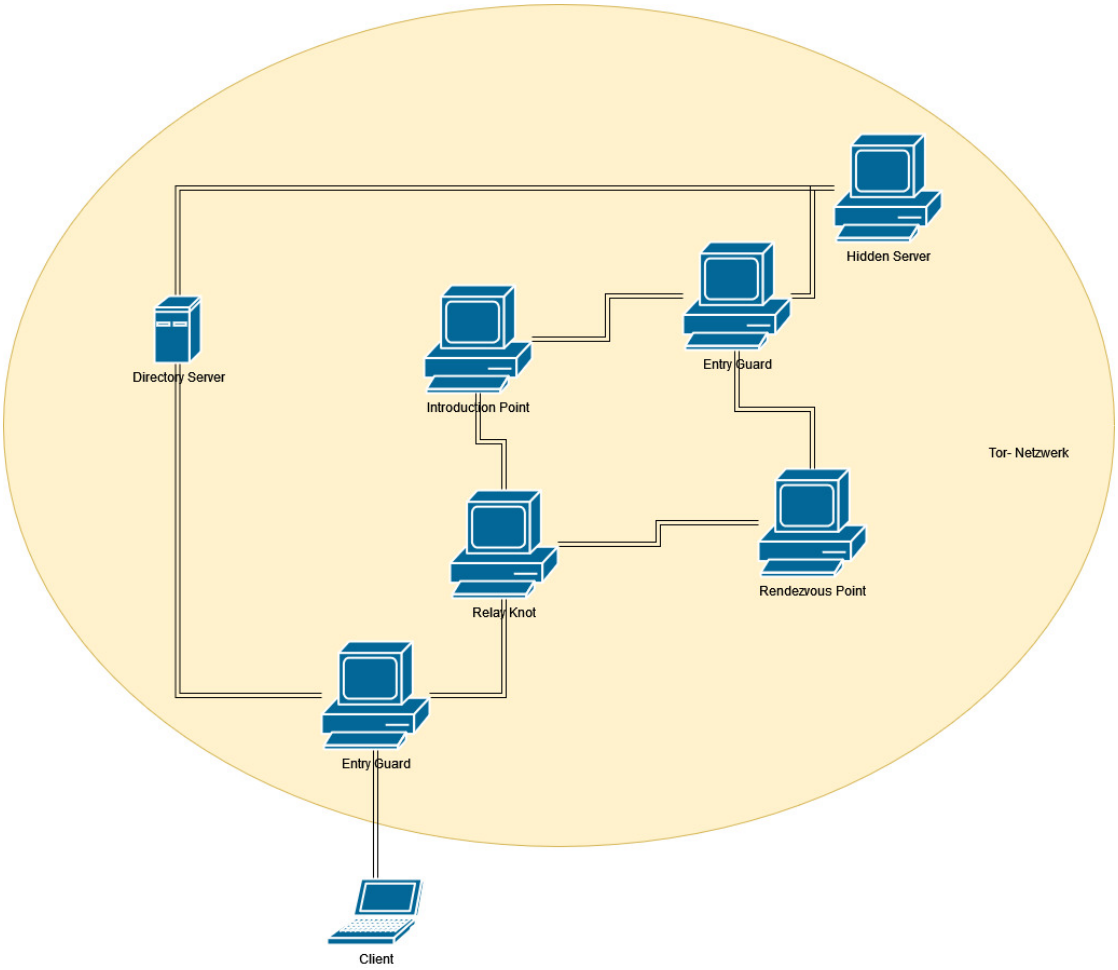

Abbildung 2.1: Eine einfache Darstellung der Verbindung von einem Client zu einem Hidden Server

haben kann.
Wenn allerdings ein TCP-Stream über 10 Minuten hinausgeht, wird der Circuit nicht erneuert, da Tor laufende TCP-Verbindungen nicht unterbricht.

2.7 Methoden/Grundlagen der Zeitoptimierung

2.7.1 Einleitung

Die Optimierung der Download-Zeit ist kein neues Problem, und es gibt bereits viele Lösungsansätze für einen zeitoptimierten Download.
Um die Zeit eines Downloads zu optimieren, gibt es verschiedene Ansätze. Diese reichen von der Verbesserung des eigenen Rechners über bessere Netzwerkstrukturen bis hin zu Software, die einen schnelleren Download verspricht.

Einige dieser Verbesserungsansätze müssen bereits vorhanden sein, wie beispielsweise Content Delivery Networks (CDNs), während andere clientseitig eingesetzt werden können. Einige Lösungsansätze sind serverseitig, wie zum Beispiel die Verwendung einer Spiegel-Seite. Da wir keinen Zugriff auf den Server haben, der die Daten bereitstellt, kommen solche Strategien für uns nicht in Frage.

2.7.2 Netzwerk Auslastung - Wann ist eine gute Zeit

Die Wahl des Zeitpunkts für einen Daten-Download kann einen signifikanten Einfluss auf die Effizienz des Prozesses haben. Insbesondere die Tageszeit kann sich erheblich auf die Netzwerkauslastung auswirken. Die Studie "Characterizing residential broadband networks"(Charakterisierung von Wohnbreitbandnetzwerken)[8] zeigt, dass es zu unterschiedlichen Uhr-Zeiten deutliche Unterschiede in der Auslastung des Internets gibt. Während der Hauptbetriebszeiten des jeweiligen Netzwerks, bei Wohungs Netzen typischerweise in den Abend- und Wochenendstunden, und bei Firemn Netzten meist deren Arbeitszeiten, ist die Auslastung tendenziell höher.

Infolgedessen können Downloads in diesen Zeiträumen langsamer sein, da die verfügbare Bandbreite aufgrund der Vielzahl von gleichzeitigen Aktivitäten begrenzt ist. Bei Downloads, die über mehrere Tage dauern, wie beispielsweise große Dateien oder Software-Updates, stellt sich möglicherweise nicht die Frage, wann der Download durchgeführt werden soll, da er zwangsläufig in die Hauptbetriebsphasen des Netzwerks fällt. Um jedoch die besten Ergebnisse zu erzielen, ist es ratsam, strategisch günstige Zeitfenster für

Downloads zu identifizieren. Diese könnten in den frühen Morgenstunden oder späten Nachtstunden liegen, wenn die Netzwerkauslastung tendenziell niedriger ist. Durch die Nutzung solcher Zeitfenster können Benutzer von schnelleren und zuverlässigeren Downloads profitieren. Es ist jedoch wichtig zu beachten, dass die optimale Zeit für einen Download je nach geografischer Lage und individuellen Netzwerkbedingungen variieren kann. Daher ist Flexibilität und Anpassungsfähigkeit entscheidend, um die bestmöglichen Ergebnisse zu erzielen.

2.7.3 CDN

Content Delivery Networks (CDNs)[20] sind Overlay-Netzwerke, die aus einem Ursprungsserver und mehreren Spiegel- oder Cache-Servern bestehen. Diese Netzwerke ermöglichen es Benutzern, Daten von einem Server herunterzuladen, der sich physisch am nächsten zu ihnen befindet, was zu verkürzten Ladezeiten und einem verbesserten Benutzererlebnis führt.

Bevor CDNs populär wurden, waren Spiegelseiten eine frühe Form der dezentralen Bereitstellung von Inhalten im Internet. Spiegelseiten sind Kopien einer Website, die auf verschiedenen Servern weltweit gehostet werden. Ähnlich wie bei CDNs ermöglichen Spiegelseiten Benutzern, Inhalte von einem Server herunterzuladen, der sich geografisch nahe befindet, was zu schnelleren Ladezeiten führt und die Belastung des Ursprungsservers verringert.

CDNs spielen jedoch eine viel größere Rolle bei der Bereitstellung von Inhalten als Spiegelseiten, da sie über fortschrittliche Routing-Algorithmen und Cache-Technologien verfügen, um Inhalte effizient zu liefern und die Netzwerkbelastung zu optimieren. Darüber hinaus sind CDNs in der Lage, dynamische Inhalte zu verarbeiten und auf Anfragen in Echtzeit zu reagieren, während Spiegelseiten in der Regel statische Kopien von Inhalten sind.

Insgesamt haben CDNs die Art und Weise revolutioniert, wie Inhalte im Internet bereitgestellt werden, indem sie die Leistung verbessern, die Verfügbarkeit erhöhen und die Last auf die Ursprungsserver reduzieren. Durch die Nutzung von geografisch verteilten

Servern bieten CDNs eine effiziente Lösung für die Bereitstellung von Inhalten auf globaler Ebene und tragen wesentlich zur Optimierung des Internetverkehrs bei.

Insgesamt sind CDNs und Spiegelseiten eine gute Möglichkeit für Betreiber von Webdiensten, um ihre Dienste besser erreichbar zu machen. Sie haben zudem den positiven Nebeneffekt, dass Flash Events und Denial-of-Service-Attacken es schwerer haben, den Dienst bzw. die Website komplett vom Internet zu nehmen.

2.7.4 Paralleliserung

Etwas parallel, also gleichzeitig, zu machen um so zeit zu sparen ist ein altbekanntes Rezept. Nun gibt es verschiedene Arten solche Dinge zu parallelisieren.

Ein bemerkenswertes Beispiel für Data Harvesting mit Parallelisierung um das ganze zu beschleunigen ist die Parallel Harvesting-Methode[19], die von Hussein Suleman in seinem Paper über Parallels harvesting in Digitale Bibliotheke[19] vorgestellt wird. Sulemans Arbeit konzentriert sich auf die Herausforderung, die rasante Zunahme von digitalen Bibliotheken effizient zu katalogisieren und zu verarbeiten, um sie sowohl für Administratoren als auch für Benutzer zugänglich zu machen. Ein wichtiger Aspekt dabei ist die effiziente Verarbeitung von Metadaten, um die Katalogisierung zu beschleunigen.

Die Parallelisierungsmethode in Sulemans Paper bezieht sich speziell auf das Protokoll OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting). Bei diesem Protokoll sendet der Server dem Benutzer so viele Dateien wie möglich und hängt am Ende einen Resumption Token an. Mit diesem Token kann der Benutzer den nächsten Abschnitt der Daten herunterladen. Die Parallelisierung erfolgt, indem der Benutzer einen unbeschäftigten (idle) Peer identifiziert und ihm das Token übergibt, damit dieser den nächsten Abschnitt der Daten abrufen kann. Durch diese Methode wird die Verarbeitung der Daten parallelisiert und beschleunigt.

2.7.5 Parallelelisierung von Downloads

Es gibt einige Forschungsergebnisse zur parallelen Datenherunterladung. Diese Forschung unterscheidet sich in der Regel zwischen zwei Arten von parallelem Download: dem "parallelenünd dem "concurrenten"Herunterladen von Daten. Wir werden uns nun genauer mit diesen beiden Ansätzen befassen.

Parallelse Herunterladen

Paralleles Herunterladen bezeichnet das gleichzeitige Herunterladen einer oder mehrerer Dateien von mehreren Servern. Um paralleles Herunterladen ausführen zu können, benötigt es eine Infrastruktur seitens der Datenbereitsteller, mindestens einen Mirror-Server, von dem die gleiche Datei gleichzeitig heruntergeladen werden kann. Dies kann man auch in Abbildung 2.2 sehen.

Beim parallelen Herunterladen werden von verschiedenen Servern gleichzeitig verschiedene Teile der Datei heruntergeladen [11]. Dabei ist es sehr wichtig, dass diese verschiedenen Server tatsächlich die gleiche also identische Datei bereitstellen.

Paralleles Herunterladen funktioniert am besten, wenn genügend Breitbandnetz vorhanden ist [10]. Ab einem gewissen Zeitpunkt, wenn zu viele Server gleichzeitig angesprochen werden, kann der hohe Verkehr im Netzwerk dafür sorgen, dass der Download sich durch Paketverlust wieder verzögert [9].

Paralleles Herunterladen kann die Verzögerungen beim Herunterladen, durch die Art wie es funktioniert, verringern [18].

Ein beliebtes Tool zum paralleles Herunterladen ist auch BitTorrent[5].

BitTorrent[6] ist ein verteiltes Peer-to-Peer (P2P) Protokoll, das es Benutzern ermöglicht, große Dateien effizient herunterzuladen und über das Internet zu teilen. Im Gegensatz zu herkömmlichen Downloadmethoden, bei denen die Datei von einem zentralen Server

heruntergeladen wird, erfolgt der Download bei BitTorrent über verschiedene Peers im Netzwerk.

Anstatt die gesamte Datei von einem einzelnen Server herunterzuladen, werden bei BitTorrent kleine Dateifragmente von verschiedenen Peers heruntergeladen. Diese Fragmente werden anschließend vom Peer, der sie heruntergeladen hat, anderen Peers im Netzwerk zur Verfügung gestellt. Dabei bevorzugt das Protokoll Peers, die ihrerseits auch Daten hochladen, gemäß der "Tit-for-TatStrategie.

BitTorrent bietet daher die Möglichkeit, die Belastung von zentralen Servern zu entlasten, da der Download-Prozess dezentralisiert ist und von vielen verschiedenen Peers im Netzwerk unterstützt wird. Diese Funktionalität wird von Unternehmen wie Blizzard genutzt, die BitTorrent zur Verteilung von Updates für ihre Spiele verwenden.

Es ist jedoch wichtig zu beachten, dass BitTorrent darauf angewiesen ist, dass die Peers sich fair am Download-Prozess beteiligen. Wenn Benutzer nur herunterladen und keine Daten hochladen, kann dies zu einem Ungleichgewicht im Netzwerk führen und die Effizienz des Downloads beeinträchtigen. das ist Tatsächlich auch ein bereits erkanntes problem was in der Literatur bereits behalted worde unteranderen durch "Free Riding in BitTorrent is Cheap"[15].

Insgesamt ist BitTorrent eine faszinierende Technologie im Bereich verteilter Downloadprozesse. Während seine Prinzipien möglicherweise nicht direkt auf andere Bereiche übertragbar sind, bietet es dennoch eine interessante Inspiration für verteilte Systeme und die Entlastung von zentralen Servern.

Concurrent Downloading

Der Concurrent Download hingegen ist ein Download, bei dem die Parallelisierung nur von einem Rechner zu einem Server geht.
Das bedeutet, dass an dem Prozess nur ein Client und ein Server beteiligt sind, wie in der folgenden Abbildung 2.3 zu sehen ist. Ein Client baut dabei mehrere Verbindungen zu einem Server auf, um so schneller Daten herunterladen zu können [14].

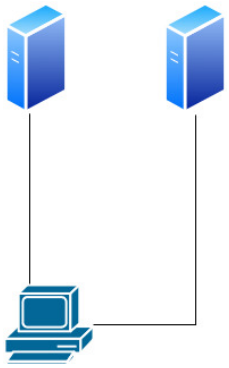

Abbildung 2.2: Ein Parallel Download einfach Dargestellt

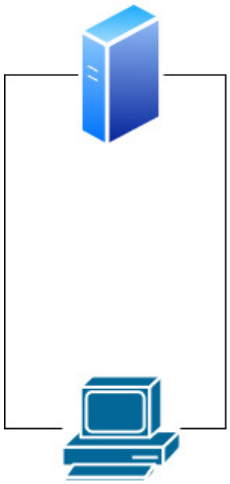

Abbildung 2.3: Ein Concurrent Download einfach Dargestellt

2.7.6 Zusammenfassung zeitoptimierender Methoden

Wir haben nun einen Blick auf zeitoptimierende Verfahren geworfen, insbesondere auf den Unterschied zwischen parallelem und concurrentem Herunterladen. Auch die Netzwerkbedingungen spielen beim Zeitoptimieren des Herunterladens eine große Rolle, sind jedoch meistens außerhalb unseres Einflussbereichs.

3 Untersuchung von existierenden Tools

In diesem Kapitel betrachten wir bereits existierende Tools, um zu verstehen, welche Anforderungen unser Tool erfüllen muss.

3.1 Wie die Downloads aus dem Darknet derzeit durchgeführt werden

Derzeit verwenden Menschen, die große Datenmengen aus dem Darknet herunterladen möchten, oft einfache Linux-Skripte. Anleitungen dazu sind im Internet verfügbar [13].

Jeder Betroffene, der seine veröffentlichten Daten aus dem Darknet herunterladen möchte, schreibt derzeit sein eigenes Skript für einen sequenziellen Download der Daten. Dies kann zeitaufwändig sein und ist kein optimiertes Vorgehen, sondern nur die erste Möglichkeit, die einem einfällt. Man muss nicht daneben sitzen, sondern kann das Skript einfach laufen lassen.

3.2 Tool Internet Download Manager

Der Download Managers IDM[12], entwickelt von der amerikanischen Firma Tonec, Inc., ist eine leistungsfähige Software für das Microsoft Windows Betriebssystem, die als Download Manager fungiert und eine Vielzahl von Funktionen bietet, um den Download-Prozess zu optimieren und zu verbessern.

IDM unterstützt sämtliche gängigen Browser und eine Vielzahl von Internetprotokollen, darunter HTTP/S, FTP und MMS. Durch die Integration in verschiedene Browser

ermöglicht es IDM den Benutzern, Downloads nahtlos und effizient durchzuführen, unabhängig vom verwendeten Webbrowser.

Eine der herausragenden Funktionen von IDM ist das Scheduling von Downloads, das es Benutzern ermöglicht, bestimmte Zeiten für den Start von Downloads festzulegen. Diese Funktion ist besonders nützlich für Benutzer, die Downloads außerhalb von Spitzenzeiten durchführen möchten, um von einer besseren Netzwerkleistung zu profitieren.
IDM bietet auch die Möglichkeit, mehrere Downloads nacheinander zu planen, was den Prozess der Verwaltung und Organisation von Downloads vereinfacht. Darüber hinaus ist IDM in der Lage, abgebrochene Downloads nahtlos wieder aufzunehmen, indem es den momentanen Fortschritt des Downloads mehrmals pro Minute speichert.
Eine weitere nützliche Funktion von IDM ist die Möglichkeit, ein bestimmtes Subset einer Website herunterzuladen, wie beispielsweise alle Bilder auf einer Webseite. Dies ermöglicht es Benutzern, gezielt bestimmte Inhalte von Websites herunterzuladen, anstatt die gesamte Seite zu durchsuchen.
Für einen schnellen Download verwendet IDM mehrere Verbindungen von dem Rechner zum Quell-Server. Wobei jede Verbindung ein eigenes Download Segment bekommt.
Eine herausragende Funktion des Internet Download Managers (IDM) ist die Dynamic Segmentation. Diese Technologie optimiert den Download-Prozess, indem sie die Dateien nicht nur vor dem Download in feste Segmente aufteilt, sondern diese Aufteilung dynamisch während des gesamten Downloads anpasst.

Im Gegensatz zu herkömmlichen Download-Managern, die Dateien in statische Segmente aufteilen, passt IDM die Größe und Anzahl der Segmente kontinuierlich an die aktuellen Netzwerkbedingungen und die Verfügbarkeit von Ressourcen an. Das bedeutet, dass IDM die Segmente während des Downloads neu verteilt und sie entsprechend der verfügbaren Bandbreite und anderen Faktoren optimiert.

Durch diese dynamische Anpassung der Segmente kann IDM die Leistung und Effizienz des Downloads maximieren, indem es sicherstellt, dass die verfügbare Bandbreite optimal genutzt wird. Dies führt zu schnelleren Download-Zeiten und einer verbesserten Stabilität des Downloads, da IDM flexibel auf Änderungen in der Netzwerkauslastung reagieren kann.

Die Dynamic Segmentation ist eine Schlüsselfunktion von IDM, die es Benutzern ermöglicht, von einer verbesserten Download-Erfahrung zu profitieren, insbesondere bei Downloads von großen Dateien oder unter schwierigen Netzwerkbedingungen. Durch die kontinuierliche Optimierung der Segmente während des Downloads gewährleistet IDM eine maximale Nutzung der verfügbaren Ressourcen und eine optimale Leistung beim Herunterladen von Dateien.

Sobald alle Segmente eines Downloads heruntergeladen sind, setzt IDM sie automatisch wieder zusammen, um die vollständige Datei zu erstellen.

IDM führt also einen Concurrent Download aus; von einem Client viele Verbindungen zu einem Server, alle über die selbe Netzwerkverbindung.

Insgesamt bietet IDM damit bereits mehrere Grundprinzipien zur Verbesserung von Down-loads. Die Parallelisierung des Downloads und die damit verbundene Aufteilung des Downloads sind besondere Merkmale von IDM.

3.3 Erkenntnis

Keine dieser Möglichkeiten versucht, verschiedene Wege zum Ziel aufzubauen, was aber eine entschiedene Engstelle beim Herunterladen aus dem Darknet ist.

Somit ist die Idee einen verteilten Concurent Download auf einem Hidden Server durchzuführen eine, die bis jetzt noch nicht umgesetzt worden ist.

4 Situationsanalyse

4.1 Situation

Aktuell befinden sich große Datenmengen auf einem versteckten Server im Darknet, die so schnell wie möglich heruntergeladen werden müssen. Die derzeit verfügbaren Tools stoßen jedoch auf signifikante Engpässe, da sie alle dieselbe Verbindung im Tor-Netzwerk nutzen und vom selben Client aus auf den Server zugreifen.

4.2 Lösungsansätze

Anstatt die Dateien mit einem einzigen Client vom Server herunterzuladen, wird vorgeschlagen, mehrere Clients zu verwenden. Diese Clients nutzen jeweils unterschiedliche Tor-Clients, was bedeutet, dass sie unterschiedliche Entry Guards und verschiedene Circuits verwenden. Dadurch entstehen unterschiedliche Wege zum Server, wodurch die Engpässe im Tor-Netzwerk umgangen werden können.

Trotzdem bleibt der Engpass bestehen, der durch den versteckten Server und dessen Entry Guard verursacht wird. Es ist zudem wichtig, sicherzustellen, dass dieser verteilte Download nicht unbeabsichtigt zu einer DDoS-Attacke führt, die die Stabilität und Erreichbarkeit des Versteckten Servers beeinträchtigen könnte.

5 Das Werkzeug zum Verteiltem Download

In diesem Kapitel wird die Entwicklung des Werkzeugs beschrieben, beginnend mit den Anforderungen, die es erfüllen soll, über die eigentliche Implementierung bis hin zu einer Bewertung über das gebaute System.

5.1 Einführung

Aktuell existiert kein Programm, das die Engpässe des Tor-Netzwerks umgehen kann. Das Ziel ist es, eine Anwendung zu entwerfen und zu entwickeln, die durch das Aufbauen verteilter Verbindungen, welche nicht alle denselben Weg durch das Netzwerk nehmen, einen besseren Durchsatz erreicht. Dies ermöglicht einen verteilten, gleichzeitigen Download von einem gemeinsamen Server.

5.2 Anforderung-Analyse

In dieser Abschnitt werden die Anforderungen an das Programm definiert. Das Werkzeug soll einen verteilten, zeitoptimierten Download aus dem Darknet durchführen. Dies soll es ohne Unterstützung von außen selbst organisieren. Das Programm ist von Informatikern für Informatiker konzipiert und soll einfach ausgeführt werden können, ohne dabei große Änderungen am Grundsystem eines Rechners vornehmen zu müssen. Das Programm bekommt eine URL und lädt dann diese und alle direkt darauf hinterlegten Dateien runter. Es muss keine Tiefen suche geschehen; wer das System dafür verwenden möchte, muss es entsprechend erweitern.

5.2.1 Zielsetzung

Ziel des System ist es einen verteilten Download auf einen Hidden Server durchzuführen. Die URL wird dabei angegeben und dann lädt das Programm alle direkt erreichbaren Dateien in ein lokales Verzeichnis herunter.
Es wird erwartet, das das System nach Aufsetzten und Start ohne weitere Interaktion des Benutzers läuft und der Benutzer nur in besonderen Fällen(z.B. Abbruch der Netzwerkverbindung) etwas am System ändern muss.

5.2.2 Umfang

Das System soll eine Grundlage für einen verteilten, parallelen Prozess zum Herunterladen von Dateien von einem Onion-Server bieten. Es soll IT-Verantwortlichen in einer Firma ermöglichen, Daten effizient aus dem Darknet herunterzuladen und dabei die Last auf mehrere Verbindungen zu verteilen. Das System muss nicht auf alle Eventualitäten reagieren können und muss auch nicht von Laien bedient werden können, sondern nur von Informatikern oder Personen mit entsprechenden Kenntnissen.

5.2.3 Stakeholder Analyse

Interessiert an der Entwicklung dieses Tools sind alle, die große Datenmengen aus dem Darknet herunterladen wollen. Dies könnte der Fall sein, wenn ihre Daten von einem Angreifer dort veröffentlicht wurden, oder aus anderen Gründen, warum diese Daten lokal benötigt werden.

Betroffene führen momentan einen sequenziellen Download aus dem Darknet durch, was sehr zeitaufwendig sein kann. Sie sind an einer schnellen Lösung interessiert und scheuen normalerweise weder Kosten noch Mühen, da es oft um das Überleben ihres Betriebs geht, indem wichtige Daten heruntergeladen werden müssen.

5.2.4 Funktionale Anforderungen

1. Herunterladen der Datein von dem Onion Server
    - Das System soll auf eine gegebene Onion-URL zugreifen können.

- Es soll alle auf der URL vorhandenen Dateien herunterladen.

2. Aufgaben aufteilung
   - Das System soll die Aufgabe des Herunterladen in Teilaufgaben aufteilen und diese auf verschiedene Clients verteilen.
3. Speicherung
   - Alle heruntergeladenen Dateien sollen an einem zentralem lokalen Speicherort abgelegt werden.
4. Fehlerbehandlung
   - Sollte ein Client seine Aufgabe nicht erfüllen, muss das System diese Aufgabe neu verteilen.

Aus diesen Anforderungen ergeben sich also drei essentielle Aufgaben an das System. Das Herunterladen der Inhalte. Das Organisieren der Verteilung der Inhalte. Und das Speichern der Inhalte auf einem lokalen Speicher Ort.
Alle diese Anforderungen sind Essentiell zum Funktionieren des Systems.

5.2.5 Nicht-funktionale Anforderungen

1. Performance
   - Da das System zur Zeitoptimierung dient, darf es nicht ineffizient sein und den Prozess negativ beeinflussen.
2. Zuverlässigkeit
   - Das System soll zuverlässig arbeiten, da es unter Umständen Tage lang laufen muss.
3. Sicherheit
   - Die Sicherheit des Systems ist zu vernachlässigen, da es nur im eigenen Netzwerk läuft und keinen Angriffen ausgesetzt sein sollte.
4. Benutzer Freundlichkeit

- Die Benutzer Freundlichkeit ist zweitrangig, jedoch soll das System von Informatikern genutzt und weiterentwickelt werden können.

5.2.6 Anwendungsfälle (USE Cases)

Anwendungfall: Aufbau des Systems

Durch diese detaillierten Schritte wird sichergestellt, dass das System korrekt aufgebaut wird und funktionsfähig ist. Die Vorbedingungen stellen sicher, dass alle notwendigen Voraussetzungen erfüllt sind, bevor die eigentliche Installation und Konfiguration beginnt. Dies minimiert das Risiko von Fehlern und Unterbrechungen während des Aufbaus des Systems.

- Ziel: Das System ist für den Einsatz aufgebaut und kann loslegen.
- Akteure:
  - System Administrator
  - Koordinator
  - Speicher
  - Client
- Vorbedingungen:
  - Der Admin muss genug Rechte auf seinen Systemen haben.
  - Das System ist in Java geschrieben also muss auch eine Java Virtual Machine zur Verfügung stehen
  - Zugang zu dem Internet
  - Der Tor Browser muss installiert sein und funktionieren
- Nachbedingungen/Erfolgskriterien:
  - System ist erfolgreich am Laufen und die Komponenten können miteinander kommunizieren.

- Hauptschritte:
    1. Vorbereitung der Umgebung: Starten des Tor Browsers und verbinden mit dem Onion Netz
    2. Ausführen der Software und das eingeben der Adresse des Koordinators bei den anderen Komponenten Darunter der Speicherplatz für den Speicher
    3. Test der Verbindungen
- Alternative Wege:
    - Bei Problem beim Starten: Admin überprüft Aufbau und behebt eventuelle Probleme
    - Kommunikation unter den Komponenten funktioniert nicht: Netzwerkeinstellungen überprüfen
    - Tor funktioniert nicht: Admin überprüft Tor Proxy Configuration

Anwendungsfall: Download einer Datei von einem Onion-Server

Durch diese detaillierten Schritte wird sichergestellt, dass das System korrekt läuft. Die Vorbedingungen stellen sicher, dass alle notwendigen Voraussetzungen erfüllt sind, bevor das System mit der eigentlichen Arbeit startet. Dies minimiert das Risiko von Fehlern und Unterbrechungen während der Laufzeit des Systems.

- Ziel: Das System lädt ein Datei oder mehrer Datein von einem Onion-Server herunter und speichert sie lokal ab.
- Akteure:
    - Koordinator
    - Speicher
    - Client
    - Onion Service
    - Tor Proxy

- Vorbedingungen:
  - Das System ist vollständig funktionsfähig und korrekt eingerichtet.
  - Die Onion-URL der gewünschten Datei ist bekannt.
  - Alle Komponenten (Koordinator, Speicher, Client) sind betriebsbereit und verbunden.
- Nachbedingungen/Erfolgskriterien:
  - Die gewünschte Datei ist erfolgreich lokal im Speicher abgelegt.
  - Alle beteiligten Komponenten (Koordinator, Client, Speicher) haben den erfolgreichen Abschluss des Vorgangs bestätigt.
- Hauptschritte:
  1. Auftragserteilung: Der Koordinator weist dem Client den Auftrag zu, die Datei von der angegebenen Onion-URL herunterzuladen.
  2. Auftragsannahme: Der Client nimmt den Auftrag vom Koordinator an und bestätigt die Auftragserteilung.
  3. Download der Datei: Der Client stellt eine Verbindung über den Tor Proxy zum Onion Service her und lädt die gewünschte Datei von der Onion-URL herunter.
  4. Übertragung der Datei: Nach erfolgreichem Download überträgt der Client die Datei an den Speicher.
  5. Speicherung der Datei: Der Speicher empfängt die Datei und speichert sie lokal ab.
  6. Erfolgsmeldung: Der Speicher meldet dem Client und dem Koordinator, dass die Datei erfolgreich gespeichert wurde.
  7. Freigabe des Clients: Der Client meldet sich beim Koordinator als verfügbar für weitere Aufträge.
- Alternative Wege:

  - Download Fehler: Wenn der Client die Datei nicht herunterladen kann, meldet er den Auftrag als fehlgeschlagen beim Koordinator. Der Koordinator teilt den Auftrag neu zu.

Abbau des Systems

Durch diese detaillierten Schritte wird sichergestellt, dass das System korrekt abgebaut wird und minimiert die Fehler beim Abbau des Systems.

- Ziel: Das System wird beendet
- Akteure:
  - Koordinator
  - Speicher
  - Client
  - Tor Proxy
  - Admin
- Vorbedingungen:
  - Das System lauft.
  - Das System ist fertig mit seiner Aufgabe.
- Nachbedingungen/Erfolgskriterien:
  - Alle Komponenten sind beendet
  - Die gewünschten Dateien liegen im lokalen Speicher
- Hauptschritte:
  1. Man erteilt dem Koordinator den Befehl das System zu beenden
  2. Der Koordinator teilt allen bei ihm gemeldeten Komponenten mit sich zu beenden und beendet sich dann selbst.
  3. Alle Komponenten beenden sich nach Erhalt der Nachricht vom Koordinator

4. Manuelles Schießen aller Tor Proxys

- Alternative Wege:
  - Komponenten laufen noch: Manuell im Betriebssystem beenden.

5.3 Das System an sich - Design

5.3.1 Einleitung

Das vorgestellte Programm ist eines um einen Verteilten Download aus dem Darknet, zu realisieren. Der Zweck des Programms besteht darin, Daten aus dem Darknet schneller herunterladen zu können, indem man den Vorgang parallelisiert. Beim Zugriff auf einen Onion Service müssen alle Daten über das Tor-Netzwerk geleitet werden, wo die Bandbreite geringer ist als im Clear Net. Daher soll das Aufteilen des Downloads auf verschiedene Clients mit unterschiedlichen Verbindungen ins Darknet zu einer Effizienzsteigerung führen.

Das Ziel der Arbeit besteht darin, ein Programm zu entwickeln, das die grundlegenden Anforderungen erfüllt. Dazu gehört, dass ein Download auf mehrere Clients aufgeteilt werden kann, die auf verschiedenen Rechnern mit unterschiedlichen Verbindungen zum Onion Server laufen. Die Kommunikation zwischen den einzelnen Komponenten erfolgt über das Clear Net.

Hintergrund dieser Aufgabe ist das die HAW beim Herrunterladen von Daten aus dem Dark net sehr lange dafür gebraucht hat und nach einer Möglichkeit suchen möchte wie das schneller zu gestalten wäre. Die Probleme, die es bei diesem Projekt zu lösen gilt, sind zu einem der zu automatisiernde Download aus dem Onion-Net und was dafür gemacht werden muss um das in einem Programm unterzubringen. Ein weiteres Problem besteht darin, die Kommunikation zwischen den verteilten Clients über das Clear-Net zu ermöglichen und eine geeignete Struktur für das Programm zu wählen.

5.3.2 Programm-Architektur

In diesem Abschnitt wird die Programm Architektur erläutert und begründet. Nach einer Erklärung der groben Architektur des Programms werden die Programmteile an sich erläutert werden.

Ein verteilter Download muss verschiedene Aufgaben erfüllen. Prominent darunter: Die Verteilung der Aufgaben, das Ausführen der Aufgaben und das Zusammenführen von allen Heruntergeladen Dateien an einem Speicher Ort.

Aus diesen Drei Aufgaben ergibt sich auch die Programm Architektur. Es gibt ein Client Programm, das die Daten herunterlädt. Ein Koordinator Programm, was die Verteilung der Aufgaben Koordiniert und ein Speicher Programm, welches dafür sorgt das alle Dateien an einem Ort gespeichert werden.

Dazu gibt es Funktionen die alle Komponenten brauchen. Diese sind in der erstellten Java Bibliothek. Die Komponenten teilen wenn möglich ihren Aufbau so das es eine möglichst große Abstraktion zwischen ihnen gibt und sie möglichst viel Programm Logik Teilen können.

Natürlich gibt es auch einige Bibliothek-Funktionen die alle diese Komponenten brauchen, wie zum Beispiel die Funktionen, um Nachrichten zu schicken.

Nachfolgend sind die Komponenten gelistet.

5.3.3 Koordinator Architektur

Die Komponente Koordinator koordiniert. Ihre Aufgabe ist es den Download der Dateien jeweils pro Datei den einzelne Clients zuzuweisen. Sie sorgt dafür das am Ende der Download vollständig durchgeführt ist. In ihr findet das Zuteilen der Aufgaben an die Clients statt.
In der Abbildung 5.1 ist eine Übersicht der Klasse zu sehen, sie zeigt auf das es im wesentlichen drei Bestandteile im Koordinator gibt. Das Datenbjekt des Koordinators eine

Main Klasse und eine Klasse die sich um alle Kommunikation kümmert.

Abbildung 5.1: Eine Übersicht über die Koordinator Komponente

Der Koordinator kommuniziert sowohl mit der Speicher-Komponente als auch mit den einzelnen Clients. Diese Kommunikationen werden alle erst mal gleich behandelt und auch gleich bearbeitet. Im Abschnitt der Bibliothek findet man mehr über den Aufbau der Nachrichten. Der Koordinator besteht aus drei einzelne Klassen. Einem Daten Objekt wo alle Daten die der Koordinator zur Erfüllung seiner Aufgaben braucht.

Einer Thread Klasse bei der für jede hereinkommende Nachricht ein Thread gestartet wird um die Nachricht zu verarbeiten, diese greifen auf das Daten Objekt zu. Und eine Initialisierungs Klasse wo der, gesamte Koordinator aufgebaut wird.

Der Koordinator verwaltet eine Liste mit allen Aufgabenteilen und welchen Zustand sie gerade haben. Also ob sie bereits einem Client zugeteilt worden sind, bereits fertig oder noch nicht zugeteilt worden sind oder der betreffende Download bereits einmal gescheitert ist.
Des weiteren verwaltet der Koordinator eine Liste von Clients, die sich bei ihm gemeldet haben und in welchem Zustand diese sich gerade befinden.

Der Koordinator ist dafür verantwortlich das am Ende alle Aufgabenteile auf die Clients aufgeteilt worden sind und auch heruntergeladen worden sind.

5.3.4 Client Architektur

Der Client Programm Teil übernimmt den eigentlichen Download. Seine Aufgabe ist es die ihm zugeteilte Datei Herunterzuladen und dann an den Speicher zu schicken. Er Schickt Updates über seinen momentanen Zustand an den Koordinator sowohl selbständig als auch auf Anfrage um bestmöglich eingesetzt werden zu können.

In Abbildung 5.2 sehen wir den Aufbau der Klasse Client sowie ihre Abhängigkeiten zu den verschiedene Bibliotheks Klassen. Der Client hat ein Data Objekt in dem alle seine Daten gespeichert werden, ein Haupt Objekt und ein Logik Objekt was ausgeführt werden kann.

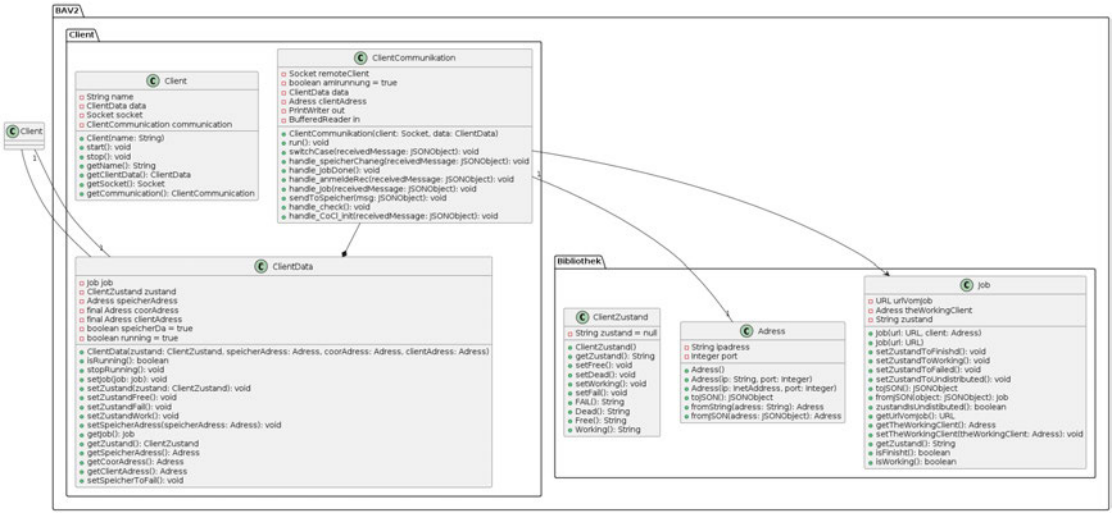

Abbildung 5.2: Ein Diagramm über die Klasse Client

Um eingesetzt werden zu können, muss der Client sich beim Koordinator melden. Dann führt er einen Test durch, ob er die Website erreichen kann und der Speicher für ihn ansprechbar ist bevor er sich zurückmeldet. Dann bekommt er vom Koordinator eine Aufgabe zugeteilt. Wenn er diese erledigt hat schickt er die Daten an den Speicher und

ein Update über seinen Zustand an den Koordinator. Dann bekommt er einen neuen Auftrag. Auf Kontrollnachrichten vom Koordinator antwortet er mit seinem momentanem Zustand.

Der Client hat ein Daten Objekt wo er die Adressen von Koordinator und Speicher sowie ihren momentan Zustand und Job festhält. Beim Empfangen einer Nachricht wird ein Thread geöffnet, der sich um die Bearbeitung dieser Nachricht kümmert.

Der Client speichert die heruntergeladenen Daten nicht extra in einem Objekt ab sondern, er schickt sie nach dem runterladen sofort weiter und behält sie nicht vor.

Der Client ist für das Ausführen der ihm zugeteilten Aufgabe sprich den Download der ihm zugeteilten Datei verantwortlich.

5.3.5 Speicher Architektur

Der Speicher speichert alle ihm zugeschickten Daten an einem Ort.
In Abbildung 5.3 ist ein Klassendiagramm des Speichers zu sehen. Auch der Speicher hat eine Daten Objekt Komponente wo alle Daten gespeichert werden eine Haupt Komponente und eine Thread Komponente.

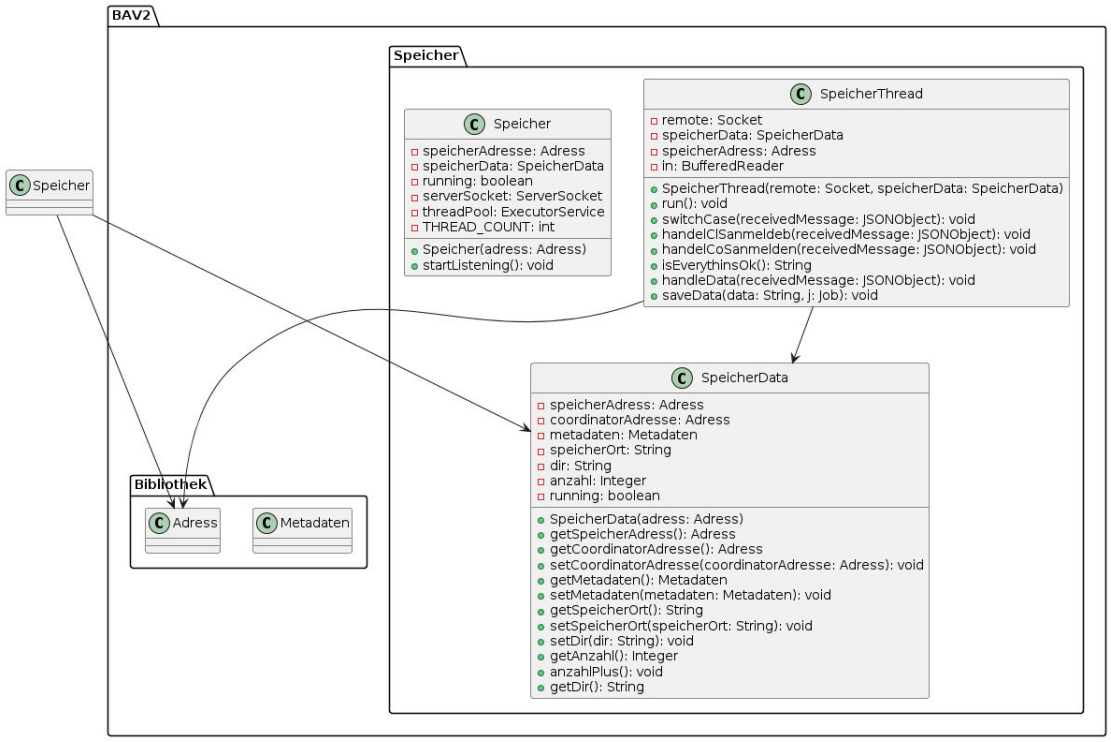

Abbildung 5.3: UML Diagramm des Speichers

Als Erstes wird der Speicher vom Koordinator kontaktiert und bekommt von ihm die Metadaten also eine Liste die alle Jobs/Aufgaben aus denen der Download besteht beinhaltet.
Von den Clients die sich bei ihm anmelden bekommt er die heruntergeladenen Inhalte zugeschickt. Dann meldet er diese Aufgabe beim Koordinator als beendet.

Der Speicher hat ein Daten Objekt in dem er diese Metadaten und den momentan Stand von ihnen speichert. Für Jede Nachricht die der Speicher bekommt wird ein Thread aufgerufen um diese zu bearbeiten.

Der Speicher gibt das Speichern der Dateien an das Betriebs System des Computers ab.

Der Speicher ist dafür verantwortlich das alle ihm zugeschickten Daten im Speicher abgespeichert werden und diese dann auch als gespeichert zu melden damit der Koordinator auch wissen kann das tatsächlich alle Aufgaben abgespeichert worden sind.

5.3.6 Bibliotheks Funktionen

Unter die Bibliotheks Funktionen zählen alle Funktionen die nicht zur Logik der Komponenten gehören.

5.3.7 Die Nachrichten Messages

Alle Nachrichten die in dem System umhergeschickt werden, werden in der Bibliothek Klasse Messages erstellt. Dort befindet sich auch die Methoden um die Daten wieder aus den Nachrichten heraus zu bekommen. So kann man wenn man will, das Nachrichten Prinzip einfach austauschen.

Alle Nachrichten, die im dem System umherum geschickt werden, sind JSONs und alle Nachrichten haben ihren eigenen Namen der in dem JSON Objekt unter ”msgName” gespeichert wird.

5.3.8 Nachrichten verschicken

In der Bibliothek gibt es auch die Klasse Sender. Diese Klasse wird genutzt um Daten an andere Mitglieder zu verschicken. Alle nutzen die selbe klasse.

5.4 Abläufe bei verschiedenen Tätigkeiten im System

In diesem Abschnitt werden wir die Abläufe im System durchsprechen und begründen.

5.4.1 Initialisierung

Um das System zu starten braucht es die URL auf der die ganzen herunterzuladenden Dateien liegen. Das heißt der Koordinator wird gestartet und baut eine Verbindung zu dem Hidden Service auf und holt sich die angegeben Website. Von dieser Website erstellt er die Metadaten in diesem Fall alle auf dieser Website verlinkten URLs als Liste an Aufträgen.

Dann baut der Koordinator eine Verbindung zum Speicher auf fragt bei ihm nach ob alles in Ordnung ist. Dafür das der Koordinator sich mit dem Speicher verbinden kann muss natürlich der Speicher vorher von dem Benutzer manuell gestartet und eingerichtet werden.

Dann wartet der Koordinator darauf, das sich bei ihm Clients melden, damit er ihnen Aufgaben zuteilen kann. Die einzelne Clients werden von dem Benutzer auf ihren eignen Systemen gestartet und bekommen dort dann auch die Adresse des Koordinators mitgeteilte.

Dies ist auch in Abbildung 5.4 zu sehnen.

Alle Fehler die in diesem Bereich auftreten werden nicht vom System selber gelöst sondern brauchen menschliche Eingriffe.
Es gibt keine automatisierte Lösung dafür, wenn im Aufbau des System ein Fehler unterläuft.

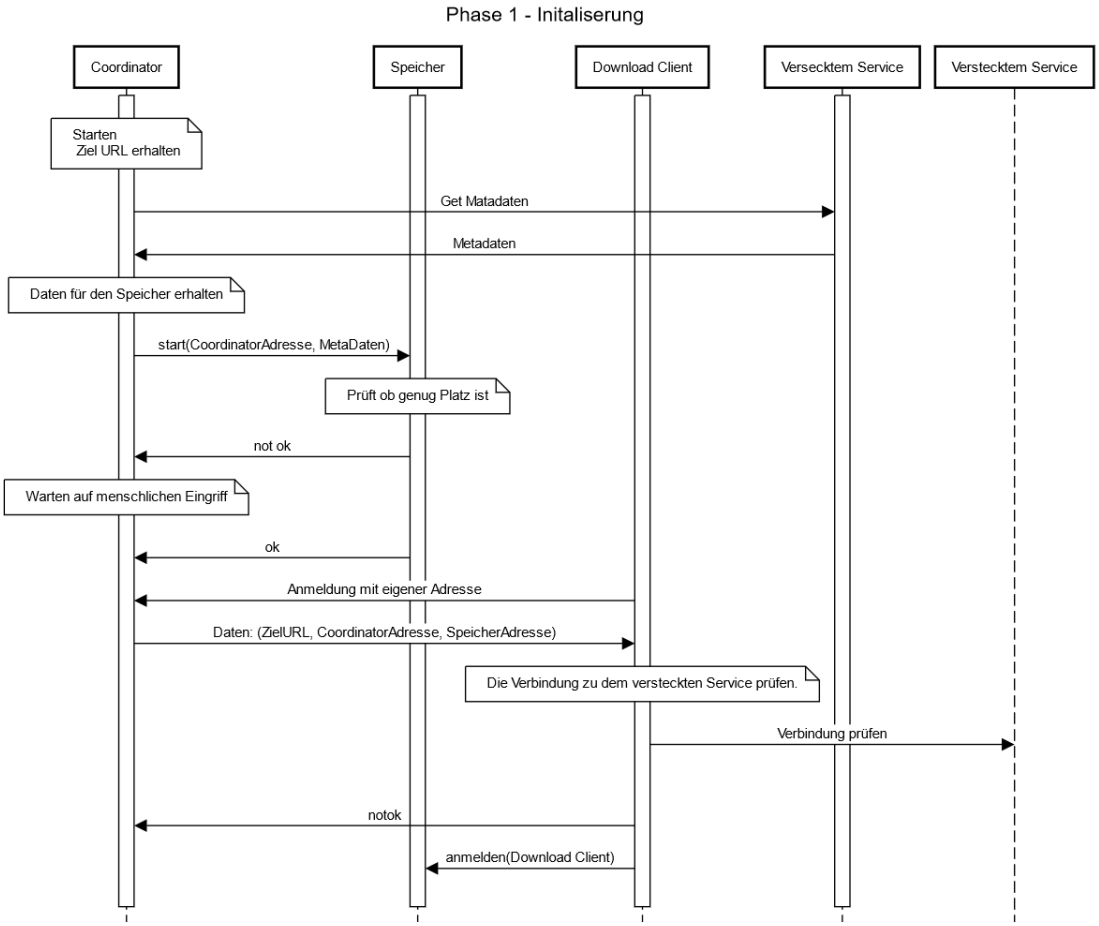

Abbildung 5.4: Sequenz Diagramm zum Aufbau des Systems

5.4.2 Laufender Betrieb

Im laufendem Betrieb geht es wie folgt voran. Ein Client meldet sich beim Koordinator. Dieser gibt ihm einige Daten darunter unter welcher Adresse er den Speicher finden kann. Der Client setzt sich auf und überprüft das er alles funktioniert. Darunter auch ob er Zugriff auf das Onion Netz hat und den Onion Server erreichen kann.

Dann meldet er sich bei dem Koordinator als einsatzbereit. Dieser teilt ihm Darauf hin eine Aufgabe zu.
Der Client lädt seine Aufgabe herunter und überträgt sie dann an den Speicher damit dieser sie lokal abspeichern kann.

Wenn er dann seine Aufgabe abgeschlossen hat meldet er sich erneut beim Koordinator das er frei ist und eine nächste Aufgabe bekommen kann.

Der Koordinator checkt regelmäßig ob die Clients noch leben. Wenn einer der Clients nicht mehr antwortet, gibt er die ihm zugeteilte Aufgabe frei und vergibt sie erneut.

Wenn ein Client keine Verbindung mehr zum Onion Server aufbauen kann meldet er sich beim Koordinator als Fail und wird ebenfalls aus dem Pool der zu Verfügung stehenden Clients entfernt.

Wenn der Client keine Verbindung mehr zum Speicher erhalten kann meldet er sich als Fail dem Koordinator und wird aus dem Pool an guten Clients genommen.

Stürzt der Koordinator im laufenden Betrieb ab so soll der Speicher noch alle Daten von allen Clients entgegennehmen.

Bekommt der Speicher von zwei Clients die gleiche Datei wird die Alte überschrieben. Da wir das speichern an sich an das Betriebs System abgeben kommt es da aber natürlich trotzdem auf das Betriebs System an.

Wenn in dem lokalen speicher nicht mehr genug Platz vorhanden ist, meldet der Speicher sich beim Koordinator das etwas nicht geklappt hat. Worauf hin der Koordinator das Herunterladen abbrechen wird und auf Eingriff durch den Betreiber gewartet wird.

Ein Sequenz Diagramm ist dazu in Abbildung 5.5 zu sehen.

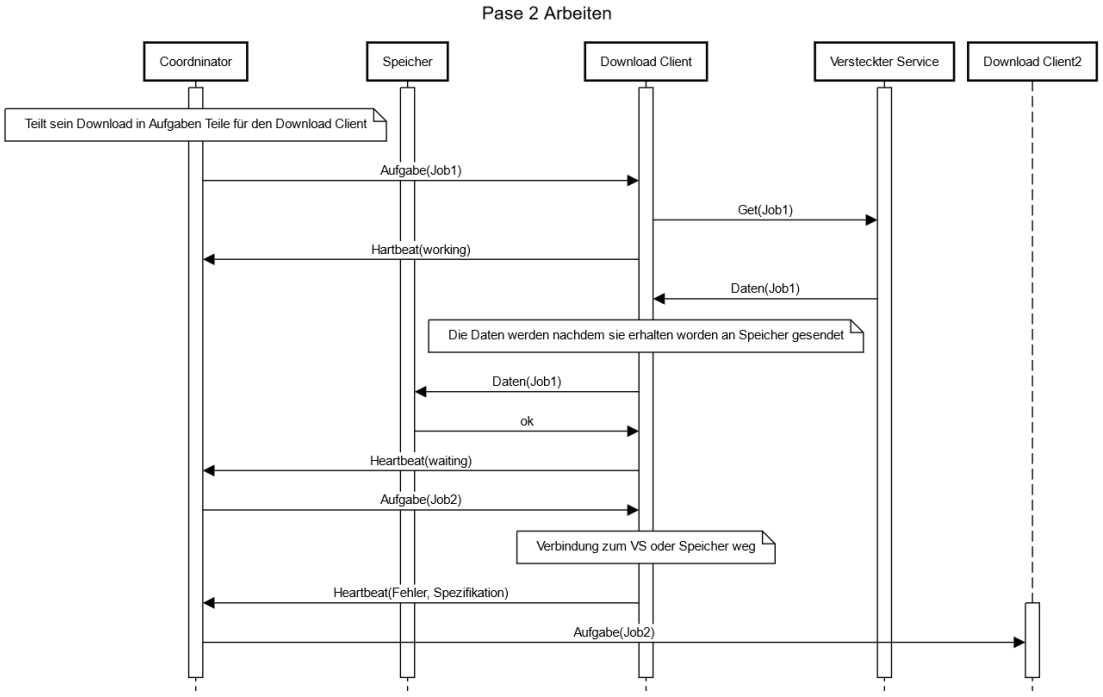

Abbildung 5.5: Sequenz Diagramm zum Arbeiten des Systems

5.4.3 Das Ende

Hat der Koordinator keine weiteren Aufgaben mehr zu Verteilen. So teilt er den Clients mit das sie nicht mehr gebraucht werden und diese beenden sich von selbst.
Wenn der Speicher alle Daten hat und er sich vom Koordinator verabschiedet hat beendet er sich ebenfalls.
Dies ist auch in Abbildung 5.6 zusehen.

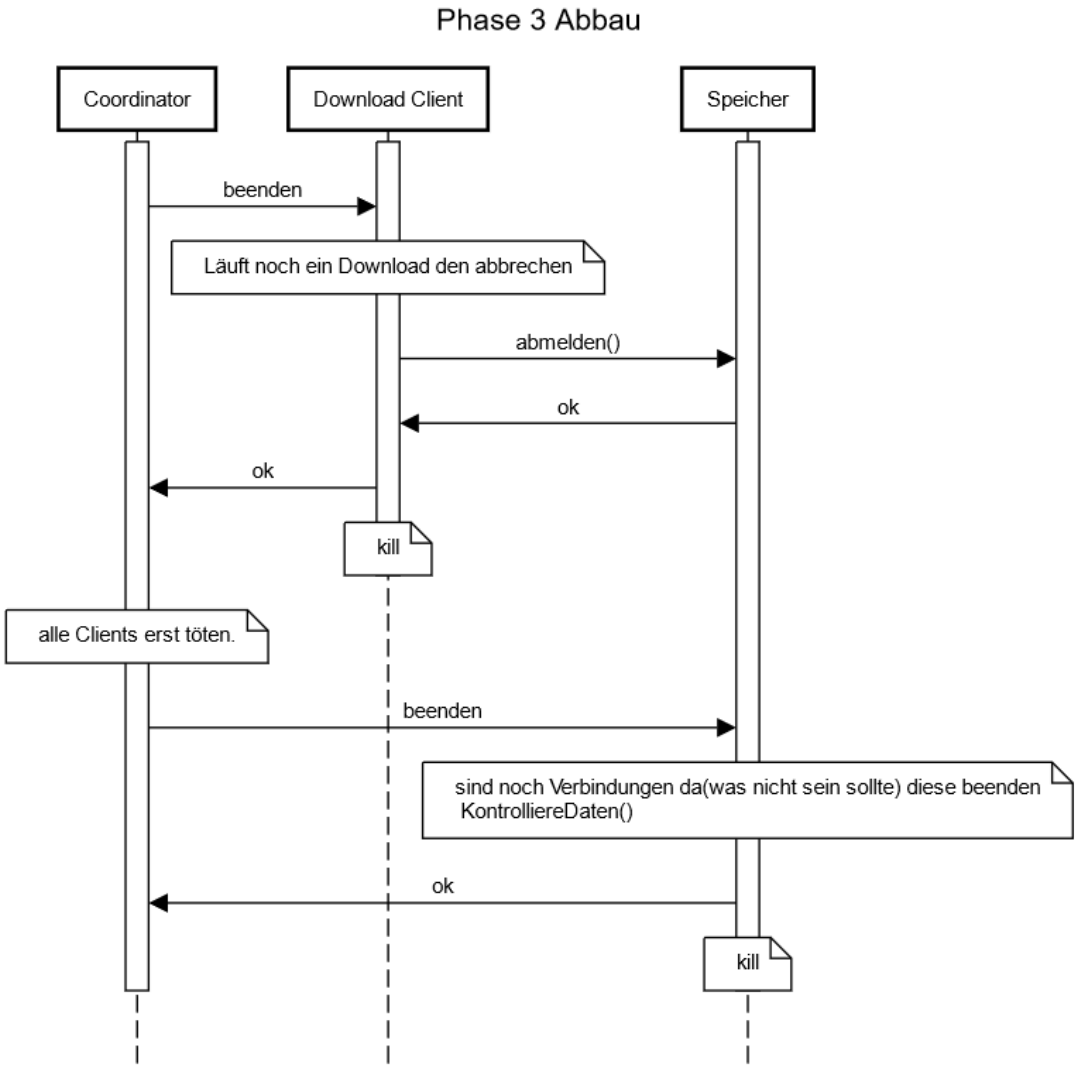

Abbildung 5.6: Sequenz Diagramm zum Abbau des Systems

5.5 Das System Umsetzung Anmerkungen

Die Anwendung ist in Java geschrieben und nutzt die org.json Bibliothek.

Die Abbildung 5.7 zeigt die wesentlichen Klassen und ihre Aufteilung in der Anwendung. Dort kann man sehen das es insgesamt drei wesentliche Komponenten sowie die Bibliothek gibt. In der Bibliothek befinden sich die Dinge die von allen Komponenten gebraucht

[h]

Abbildung 5.7: UML-Diagramm mit allen wesentlichen Klassen

werden während die Komponenten überwiegend nur ihre Logik enthalten.

5.5.1 Einbindung des Tor Cleints

Damit das System auf das Onion Network zugreifen kann muss es den Tor Clients benutzen. Bei mir läuft das System unter Windows und dort ist die Einbindung relativ simple.

Man lädt den Tor Browser runter. Damit das System auf das Netzwerk zugreifen kann muss man dann natürlich auch den Tor Browser öffnen und ihn verbinden. Tor läuft Standard mäßig auf Port 9050. Wenn man also bei laufendem Tor Proxy diesen Port nutzt agiert man über das Tor Netzwerk.
Bei Linux ist die Installieren von Tor ebenfalls kein großes Problem und man lädt einfach Tor ganz normal runter.

5.5.2 Kommunikation zwischen den Komponenten

Die Komponenten kommunizieren miteinander über folgendes Protokoll:

Die Nachrichten sind im JSON-Format strukturiert, wobei alle Nachrichten den selben Aufbau haben. Das Fel "msgname" enthält den Namen der Nachricht.
Die Namen folgen einem bestimmten Format: Zuerst steht "MSG", gefolgt von einem Unterstrich, dann kommt eine Bezeichnung, die angibt, von wem die Nachricht gesendet wird ("Co" für den Koordinator, "Cl" für den Client und "S" für den Speicher), gefolgt von einer beschreibenden Bezeichnung für die Nachricht.

Ein Beispiel wäre "MSG_CoS_update",was eine Benachrichtigung vom Speicher an den Koordinator darstellt, dass ein weiterer Teil des Auftrags einen neuen Zustand erreicht hat. Die weiteren Inhalte in der JSON Nachricht sind in diesem Fall dann "JobID" und "JobZustand", damit der Koordinator den neuen Zustand des Jobs richtig einpflegen kann.

Es folgt eine Tabelle 5.1 mit allen verwendeten Nachrichten:

Die Übertragung dieser Nachrichten erfolgt mithilfe von Java Sockets. Java Socktes sind eine einfache Möglichkeit eine bidirektionale Verbindung zwischen den Komponenten zu Implementieren. Sie brauchen keine Plattform oder sonstige Abhängigkeit um zu funktionieren, und können auf einer Vielzahl von Systemen ausgeführt werden, ohne das man sie an das Betriebssystem auf dem sie sind anpassen muss.
Wenn eine unbekannte Nachricht ankommt so wird diese einfach verworfen und auf gültige Nachrichten gewartet.

5.5.3 Ausführen des Programms

Das System kann derzeit über eine Java-JAR-Datei gestartet werden. Bei der Ausführung der Datei hat der Benutzer die Möglichkeit, auszuwählen, welche Komponente gestartet werden soll. Die Eingaben, die von den Komponenten benötigt werden, erfolgen über die Konsole. Alle Log-Nachrichten werden in eine lokale Logdatei geschrieben.

Tabelle 5.1: Beschreibung der Nachrichten im Kommunikationsprotokoll

Name	**Beschreibung**	**Detail**
MSG_CoS_init	Metadaten vom Download an Speicher	{msgname, Coor Adresse, Metadaten}
MSG_CoS_init2	Antwort auf init, ob alles funktioniert hat	{msgname, Status Speicher}
MSG_CoS_update	Update Nachricht von Speicher an Koordinator	{msgname, JobID, JobZustand}
MSG_CoCl_anmelden	Der Client meldet sich beim Koordinator an	{msgname, Client}
MSG_CoCl_init	Client bekommt Parameter zugeschickt	{msgname, URL-Standard, Speicher-Adresse}
MSG_CoCl_init2	Client gibt zurück, ob alles funktioniert	{msgname, Status, Adresse}
MSG_CoCl_update	Client gibt Koordinator ein Update	{msgname, Status, Adresse}
MSG_CoCl_Job	Koordinator teilt Client einen Job zu	{msgname, Job}
MSG_CoCl_check	Koordinator checkt den Client	{msgname}
MSG_ClS_anmelden	Client meldet sich beim Speicher an	{msgname, Client}
MSG_ClS_anmdenrec	Antwort auf Anmelden, ob alles geklappt hat	{msgname, Status}
MSG_ClS_Data	Die Datei	{msgname, Job, Data}
MSG_ClS_erhalten	Nachricht, ob die Datei erhalten wurde	{msgname, Job}
MSG_ClS_done	Nachricht an Client, dass er mit seinem Auftrag fertig ist	{msgname, Job}

5.6 Auswertung

Das entwickelte System ermöglicht einen verteilten Download von einem Hidden Server. Durch die Verteilung der Download-Aufgaben auf verschiedene Clients kann die Effizienz des Downloads verbessert werden. Die Implementierung basiert auf einer klaren Programm Architektur, die aus den drei Hauptkomponenten - Koordinator, Client und Speicher - besteht.

Ein Download über Tor ist immer langsamer als über das normale Netzwerk. Dies liegt hauptsächlich am erhöhten Aufwand im Bereich der Verschlüsselung und Weiterleitung. In der Grafik 5.9 ist dies deutlich erkennbar. Dort wurden über einen Zeitraum von sechzig Minuten Messwerte während eines Downloads aufgenommen. Die Messreihe mit den Downloads aus dem normalen Netzwerk ist die stabilste und auch die schnellste, was zu erwarten war. Ein Download über das Tor-Netzwerk auf einen Server im Clearnet benötigt wesentlich mehr Zeit, und ein Download von einem Hidden Server ist in dieser Messung sogar doppelt so ineffizient.

Um das System zu bewerten, habe ich folgendes Experiment durchgeführt: Der gleiche Download wurde mit dem System drei Mal durchgeführt. Einmal mit nur einem Client, einmal mit zwei Clients und einmal mit drei Clients.
Der Download wurde von dem System auf einen Hidden Server durchgeführt, den wir selbst gehostet haben.
Das Ergebnis dieser Downloads ist in Abbildung 5.8 zu sehen. Der Download mit einem Client war erwartungsgemäß der längste und dauerte dreimal so lang wie der mit zwei Clients. Mit drei Clients war die Verbesserung dann nicht mehr ganz so stark, aber dennoch signifikant.

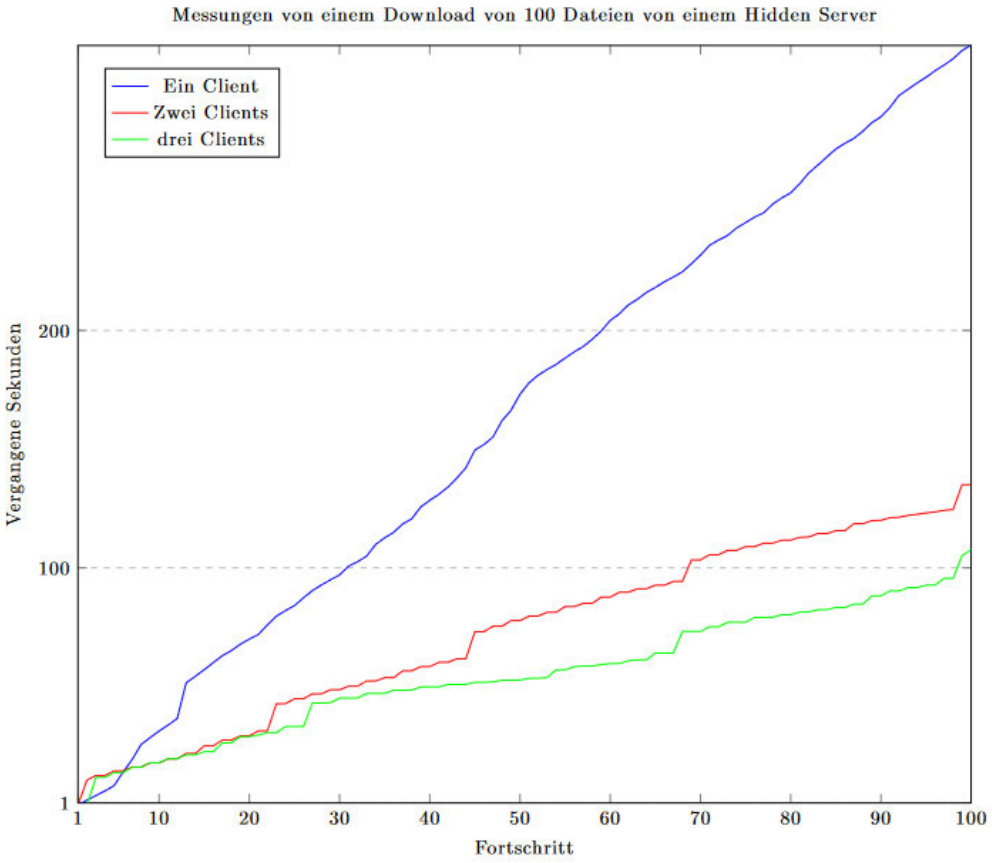

Abbildung 5.8: Abbildung der Messung eines Downloads mit unterschiedlich vielen Clients.

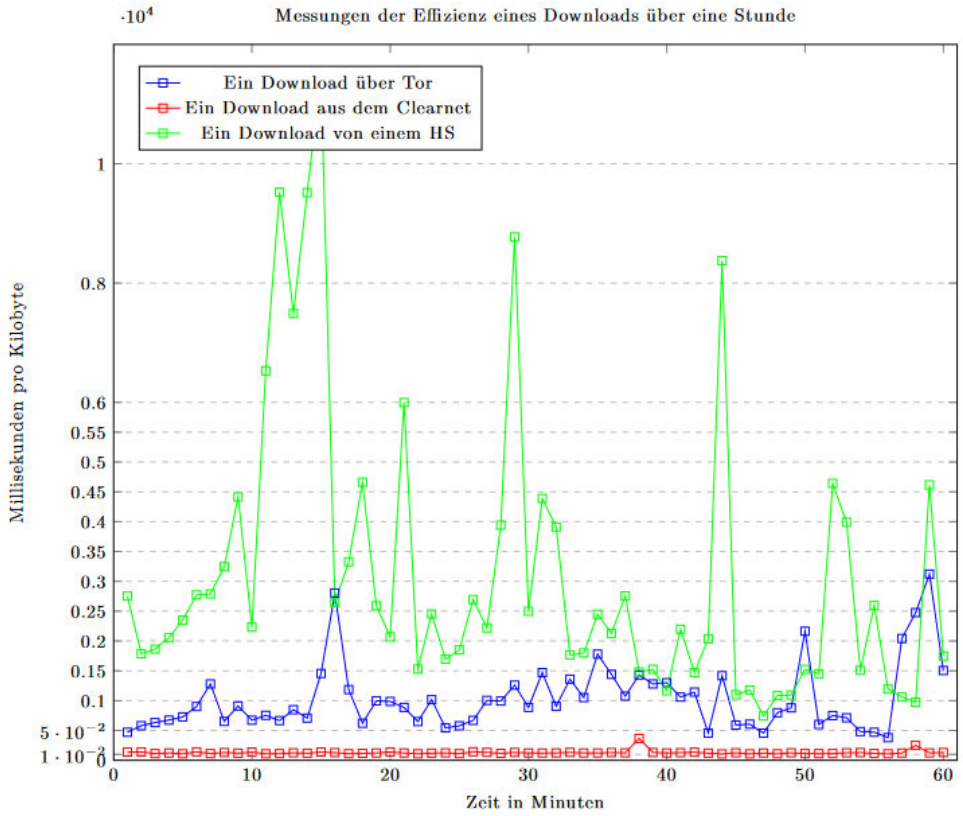

Abbildung 5.9: Effizienz der Netzwerkverbindungen im Download. Drei Messreihen. Normales Internet, Verbindung über Tor zu einem normalen Server und Verbindung über Tor zu einem Hidden Server.

6 Fazit

Die vorliegende Bachelorarbeit beschäftigte sich mit der Entwicklung einer verteilten Anwendung zum zeitoptimierten Download aus dem Darknet. In diesem Zusammenhang wurden verschiedene existierende Verfahren zum Download beleuchtet, um die Grundlagen für die Entwicklung der verteilten Anwendung zu legen. Durch die Implementierung der vorgestellten verteilten Anwendung wurde ein wichtiger Schritt in Richtung effizienter und sicherer Datenübertragung aus dem Darknet unternommen.

Die entwickelte verteilte Anwendung bietet eine solide Basis für weitere Entwicklungen und Erweiterungen. Insbesondere könnten zukünftige Arbeiten darauf abzielen, die Leistungsfähigkeit und Skalierbarkeit der Anwendung weiter zu verbessern, um den Anforderungen eines breiteren Anwenderkreises gerecht zu werden.

Ein wichtiger Aspekt, der im Rahmen dieser Arbeit festgestellt wurde, ist die Einschränkung der Verteilung aufgrund der verwendeten Technik. Aktuell ist es nur möglich, die verteilte Anwendung innerhalb eines Netzwerks zu betreiben. Dies stellt eine Herausforderung dar, die in zukünftigen Forschungsarbeiten adressiert werden sollte, um die Reichweite und Zugänglichkeit der Anwendung zu erweitern.

Insgesamt liefert diese Arbeit einen Beitrag zum Verständnis und zur Entwicklung von Werkzeugen zur effizienten Datenübertragung aus dem Darknet. Sie legt den Grundstein für weitere Forschung und Entwicklung auf diesem Gebiet und zeigt auf, dass trotz der aktuellen Beschränkungen viel Potenzial für die Verbesserung und Weiterentwicklung der Technologie besteht.

6.1 Ausbau Möglichkeiten des Tools

Es gibt mehrere Ansätze, wie das System verbessert werden könnte:

Das aktuelle Kommunikationsverfahren basiert auf Java Sockets, die in ihrem ursprünglichen Zustand nur für die Kommunikation innerhalb desselben Netzwerks geeignet sind. Um die verschiedenen Teile des Systems in unterschiedlichen Netzwerken betreiben zu können, wäre es notwendig, auf eine andere Art der Kommunikation umzusteigen, die auch über das Internet hinweg funktioniert. Möglichkeiten könnten die Verwendung von HTTP/HTTPS, WebSockets oder spezialisierte Messaging-Protokolle sein.

Eine weitere Möglichkeit wäre, das System zu einem echten Peer-to-Peer-Netzwerk auszubauen. In einem solchen Netzwerk hätte jeder Teilnehmer dieselben Rechte und Pflichten, was zu einer besseren Skalierbarkeit und Redundanz führen könnte. Dies würde auch die Abhängigkeit von einem zentralen Koordinator verringern und das System robuster machen.

Die Implementierung einer automatischen und dynamischen Anpassung der aktiven Clients könnte dazu beitragen, die Belastung des Quellservers zu minimieren. Durch eine intelligente Steuerung könnte das System in Echtzeit auf die Serverkapazität reagieren und sicherstellen, dass dieser nicht überlastet wird, während gleichzeitig die Effizienz des Downloads maximiert wird.

6.2 Mögliche weiter Forschungsansätze

Es gibt mehrere interessante Forschungsrichtungen, die auf dieser Arbeit aufbauen könnten:

Innerhalb des Umfangs dieser Arbeit wurden keine umfassenden Messungen der Download-Performance durchgeführt. Eine detaillierte Untersuchung der Leistungssteigerung durch Verteilung und Parallelisierung der Downloads wäre eine lohnenswerte Untersuchung. Hier könnte verglichen werden, wie viel effizienter der verteilte Ansatz im Vergleich zu

einem traditionellen Downloadverfahren ist.

Die Auswirkungen eines verteilten, parallelisierten Downloads auf den Quellserver sind ebenfalls ein interessantes Forschungsgebiet. Es wäre wichtig zu untersuchen, ob die Last hauptsächlich auf die Tor-Relays oder den Hidden Service selbst fällt. Diese Untersuchungen könnten helfen, die Belastungsgrenzen besser zu verstehen und Strategien zur Lastverteilung zu entwickeln, die sowohl den Server als auch das Netzwerk schonen.

Eine weitere Forschungsrichtung könnte die Analyse der Netzwerkdynamik und der Sicherheitsaspekte umfassen. Wie verändert sich die Effizienz des Systems unter unterschiedlichen Netzwerkbedingungen? Welche Sicherheitsrisiken bestehen bei der Nutzung eines verteilten Systems im Darknet, und wie können diese minimiert werden?

Diese Erweiterungen und Forschungsansätze bieten viele Möglichkeiten, das System weiterzuentwickeln und gleichzeitig neue Erkenntnisse im Bereich verteilter Systeme und der Datenübertragung im Darknet zu gewinnen.

Literaturverzeichnis

[1] Cyber-Angriffe weltweit 2022, KonBriefing.com

[2] Methodically Modeling the Tor Network. In: *5th Workshop on Cyber Security Experimentation and Test (CSET 12)*. Bellevue, WA : USENIX Association, August 2012. – URL https://www.usenix.org/conference/cset12/workshop-program/presentation/Jansen

[3] : *Tor metrics*. 2024. – URL https://metrics.torproject.org/networksize.html. – Zugriffsdatum: 2024-05-18

[4] : *TorProjekt Support*. 2024. – URL https://support.torproject.org/about/change-paths/. – Zugriffsdatum: 2024-05-18

[5] Chen, Haitao ; Gong, Zhenghu ; Huang, Zunguo: Parallel Downloading Algorithm for Large-volume File Distribution, 01 2005, S. 745–749

[6] Cohen, Bram: Incentives build robustness in BitTorrent. In: *Workshop on Economics of PeertoPeer systems* 6 (2003), 06

[7] Dingledine, Roger ; Mathewson, Nick ; Syverson, Paul: Tor: The Second-Generation Onion Router. In: *Paul Syverson* 13 (2004), 06

[8] Dischinger, Marcel ; Haeberlen, Andreas ; Gummadi, Krishna P. ; Saroiu, Stefan: Characterizing residential broadband networks. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. New York, NY, USA : Association for Computing Machinery, 2007 (IMC '07), S. 43–56. – URL https://doi.org/10.1145/1298306.1298313. – ISBN 9781595939081

[9] Funasaka, Junichi: Evaluation on parallel downloading method using HTTP over UDP, 04 2009, S. 1 – 6

[10] Funasaka, Junichi ; Kawano, A. ; Ishida, K: Implementation Issues of Parallel Downloading Methods for a Proxy System, 07 2005, S. 58– 64. – ISBN 0-7695-2328-5

[11] Funasaka, Junichi ; Kawano, Atsushi ; Ishida, Kenji: Adaptive Parallel Downloading Method for Proxy Systems. In: *IEICE Transactions* 90-B (2007), 04, S. 720–727

[12] FZE, Tonec: InternedDownloadManager, April 2024

[13] Hoek: *Downloading big files from Tor*. 2024. – URL https://0ut3r.space/2022/09/30/big-files-from-tor/. – Zugriffsdatum: 2024-05-18

[14] Liu, Yong ; Gong, Weibo ; Shenoy, P.: On the impact of concurrent downloads. In: *Proceeding of the 2001 Winter Simulation Conference (Cat. No.01CH37304)* Bd. 2, 2001, S. 1300–1305 vol.2

[15] Locher, Thomas ; Moore, Patrick ; Schmid, Stefan ; Wattenhofer, Roger: Free Riding in BitTorrent is Cheap. In: *5th Workshop on Hot Topics in Networks (HotNets)*, URL http://eprints.cs.univie.ac.at/5696/, 2006

[16] Øverlier, Lasse ; Syverson, Paul: Locating Hidden Servers. In: *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, IEEE CS, May 2006

[17] Prabhu, Vibha ; Raghuraman, Krishna P. ; Ragulnaath, V. ; Selvaraj, Dharani: Development of Download Manager using Distributed Approach, URL https://api.semanticscholar.org/CorpusID:212547603, 2017

[18] Priyanka, Miss ; Pursuing, Sahu ; Shri, M ; Patel, Mr: Delay Reduction In File Download Through Parallelization. In: *International Journal on Computer Science and Engineering* 4 (2012), 06

[19] Suleman, Hussein: Parallelising harvesting. In: *Proceedings of the 9th International Conference on Asian Digital Libraries: Achievements, Challenges and Opportunities*. Berlin, Heidelberg : Springer-Verlag, 2006 (ICADL'06), S. 81–90. – URL https://doi.org/10.1007/11931584_11. – ISBN 3540493751

[20] Vakali, A. ; Pallis, G.: Content delivery networks: status and trends. In: *IEEE Internet Computing* 7 (2003), Nr. 6, S. 68–74

[21] Wright, Matthew ; Adler, Micah ; Levine, Brian N. ; Shields, Clay: Defending Anonymous Communication Against Passive Logging Attacks. In: *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003, S. 28–43

A Anhang

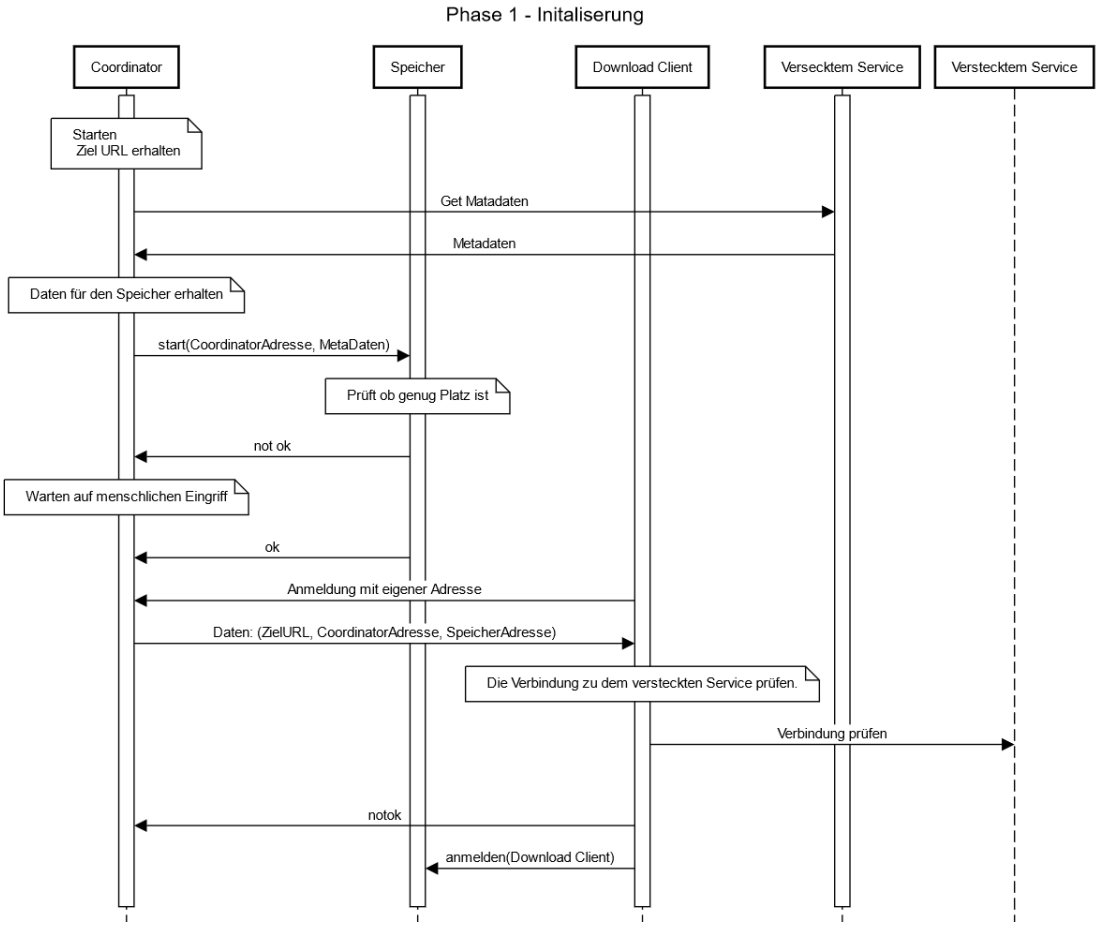

Abbildung A.1: Sequenz Diagramm für den Aufbau des Systems

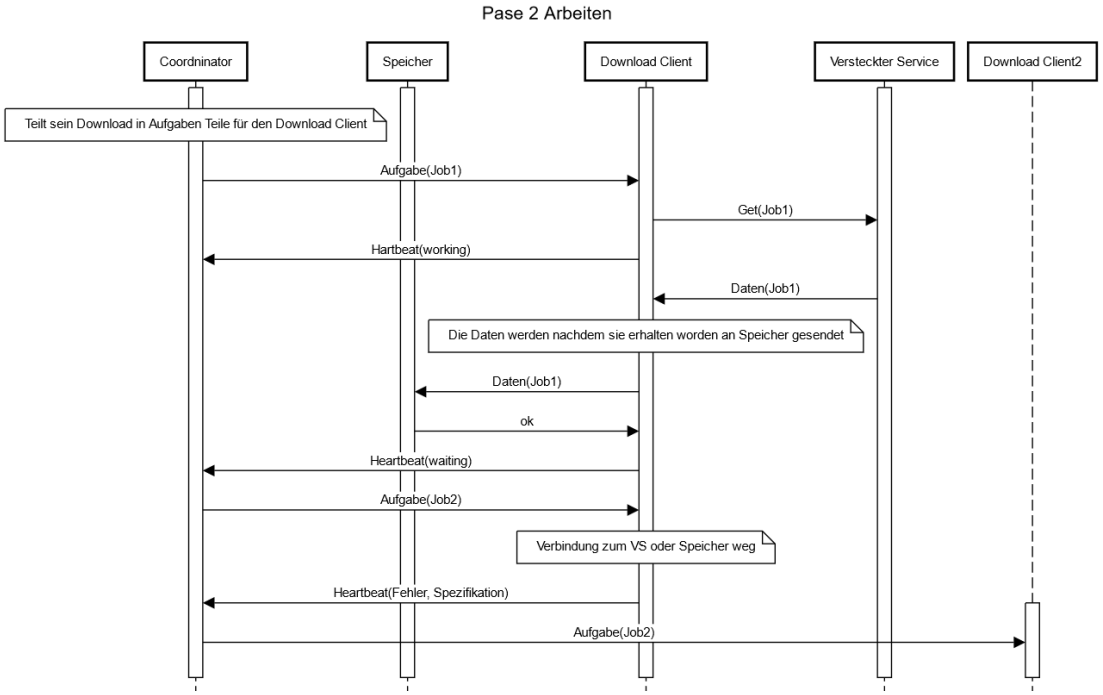

Abbildung A.2: Sequenz Diagramm für den betrieb des Systems

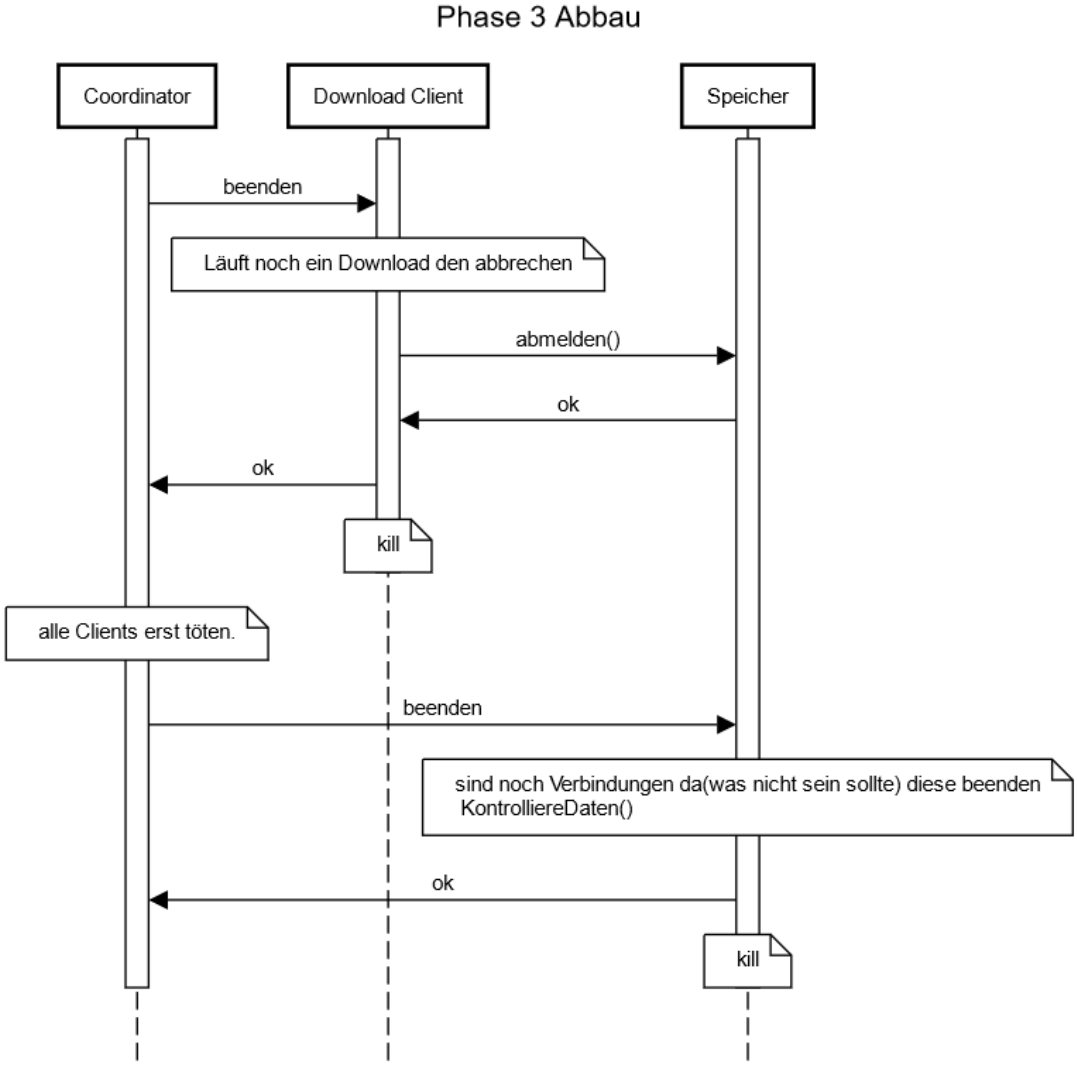

Abbildung A.3: Sequenz Diagramm für den Abbau des Systems

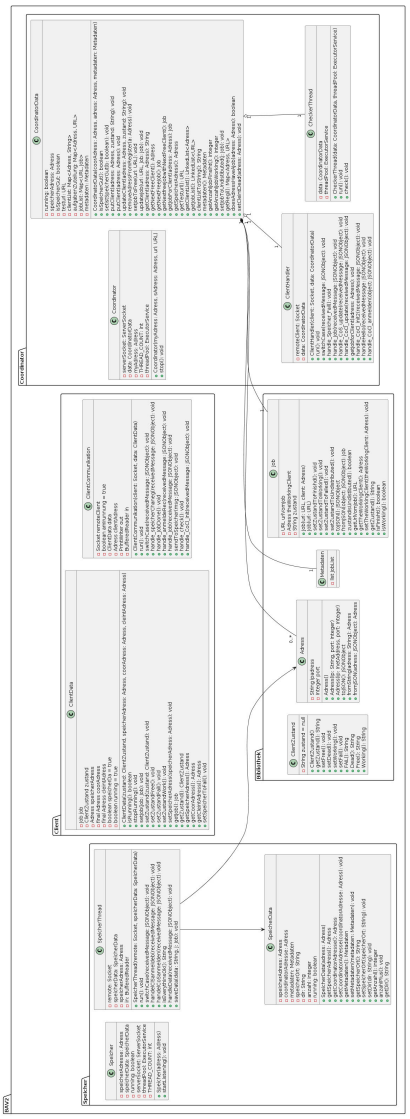

Abbildung A.4: Ein Überblick über alle Klassen im System

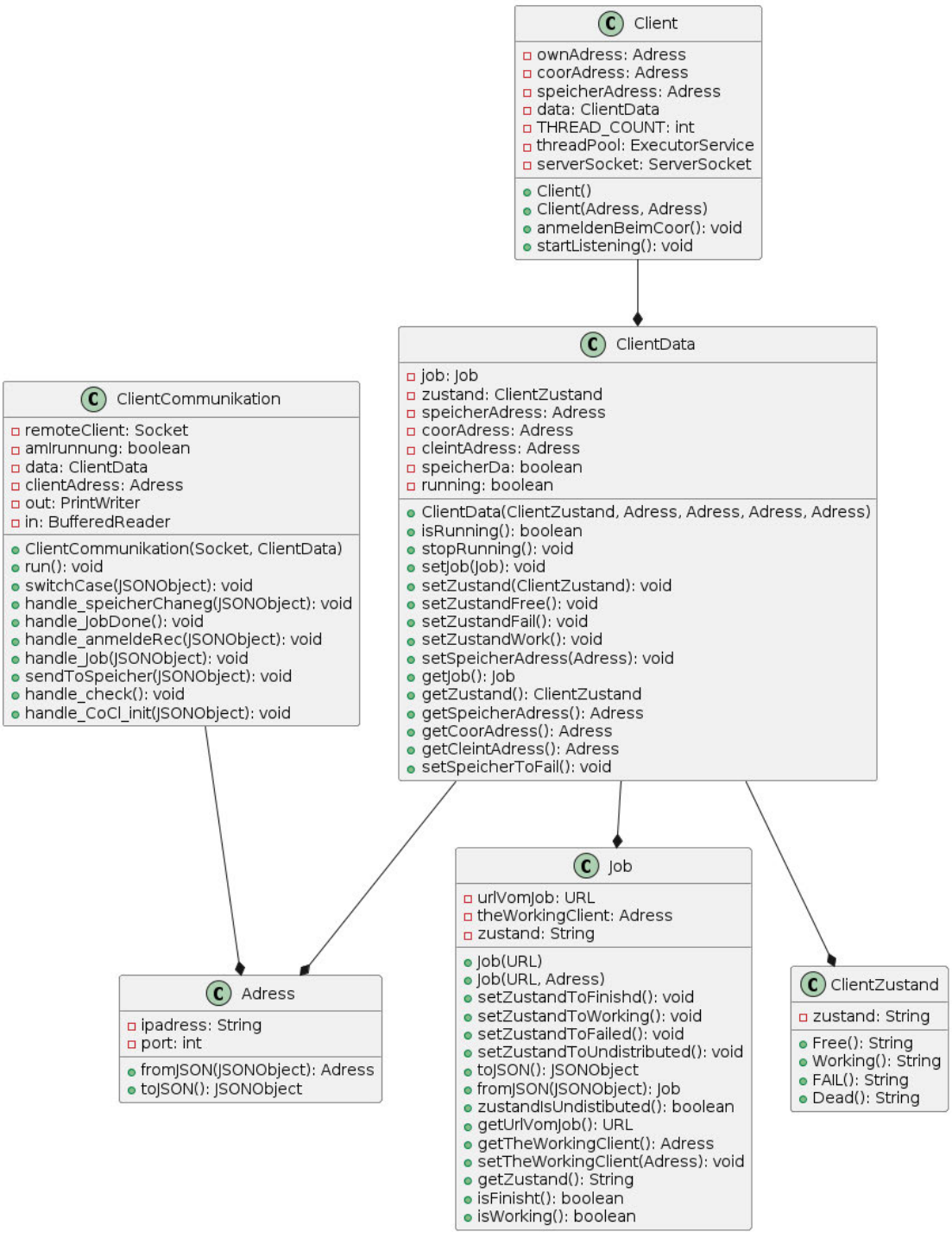

Abbildung A.5: Einen Überblick über die Client Komponente

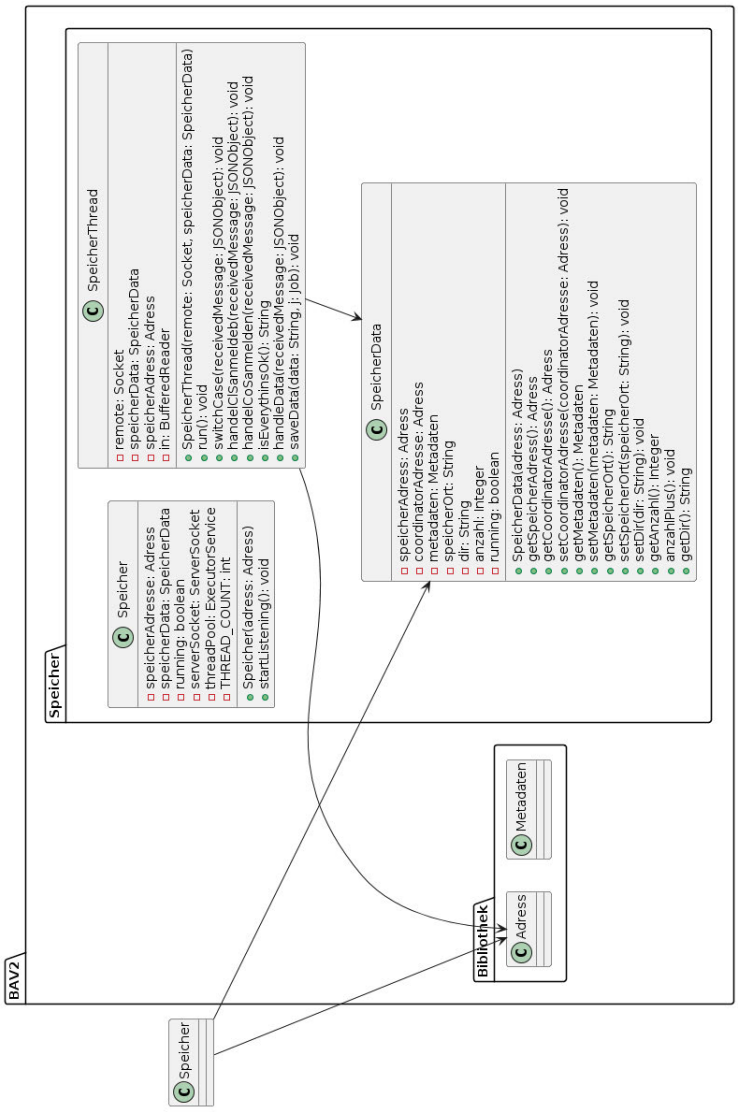

Abbildung A.6: Ein Überblick über die Speicher Komponente

Abbildung A.7: Ein Überblick über die Koordinator Komponente des Systems

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original