Projekt Orion

Jakob Klemm, Dominik Keller

Inhaltsverzeichnis

TODO: Abstract

Vorwort

Die schriftliche Komponente der Maturaarbeit `Projekt Orion` besteht aus drei verschiedenen Teilen. Da die behandelten Themen äusserst komplex und umfangreich sind, verlangen verschiedene Abschnitte der Arbeit verschiedenes Vorwissen und einen verschiedenen Zeitaufwand. Deswegen wurde die schriftliche Komponente in drei Subkomponenten aufgeteilt, wobei sie nach technischem Detailgrad sortiert sind. Wer nur ein oberflächliches Verständnis über die Arbeiten und eine Analyse des Umfelds will, ohne dabei zu technisch zu werden, muss nicht über den Umfang dieses Dokuments hinaus. Aber für vollständigen Einblick in die Errungenschaften und Konzepte muss mit einem grösseren Aufwand gerechnet werden.

- Schriftlicher Kommentar: In diesem Dokument hier findet sich eine klassische Besprechung der Arbeit. Begonnen mit einer Zielsetzung und Besprechung verschiedener Projekte, bis zur Analyse des Produkts und einem Ausblick in die Zukunft gibt dieses Dokument einen guten, aber oberflächlichen Einblick in das `Projekt Orion`. Natürlich wird besonders bei der Analyse der existierenden Projekten und Darlegung des Konzepts gewisses technisches Know-How benötigt, aber es wurde versucht, alle Fachbegriffe zu umschreiben oder zu erklären. Wer nur über die Vision und den aktuellen Stand wissen will muss nicht über dieses Dokument hinaus, aber verschiedene Konzepte und nahezu die gesamte technische Umsetzung befinden sind nicht in diesem Dokument.

- Dokumentation: In dieser alleinstehenden Dokumentation, welche im Detailgrad zwischen dem schriftlichen Kommentar und der Code-Dokumentation steht, werden die Konzepte und Ideen besprochen. Wer die Entstehung und aktuelle Form der Komponenten genauer verstehen will, oder wer von den umgesetzten Funktionen profitieren will, sollte die Dokumentation durcharbeiten. Das Dokument ist eher umfangreich, es kann aber auch gut als eine Art Nachschlagewerk verwendet werden.

- Code: Neben der Dokumentation des Projekts und der Konzepte, existiert eine weitere Form der Dokumentation. Nahezu jede Funktion, jedes Modul und jedes Objekt über die verschiedenen *Crates* sind dokumentiert. Diese Dokumentationen lassen sich nicht in einem klassisch strukturierten Dokument finden. Stattdessen ist die Code-Dokumentation online über automatisch generierte Rust-Dokumentation zu finden. Die Seiten mögen anfangs etwas unübersichtlich wirken, wer aber den Code von `Projekt Orion` verwenden will wird sich dort gut zurecht finden.

1 Vision

1.1 Grenzen

1.2 Inhalte

1.3 Routing

1.4 Zentralisierung

1.5 Orion

2 Projekte

2.1 Kademlia

Wie geht man also gegen die totale Abhängikeit von Internetanbietern und zentralen Routern vor?
Man kann ja nicht einfach seine eigenen Router aufsetzen und einen alternativen Dienst anbieten. Neben den technischen Schwierigkeiten würde ein solcher Schritt auch überhaupt nichts das eigentliche Problem bekämpfen.

Der Trick, der bei Systemen wie `Kademlia` verwendet wird, ist es, Router vollständig zu eliminieren. Dies ist möglich, indem jedes Mitglied des Netzwerks neben seinen normalen Funktionen gleichzeitig auch noch als Router agiert. Strenggenommen werden in `Kademlia` Router also nicht wirklich eliminiert, lediglich zentrale Router fallen weg.

In einem früheren Abschnitt wurden die Probleme der *Zentralrouter* bereits angesprochen. Wenn jetzt aber jedes Mitglied in einem Netzwerk plötzlich als Router agiert und es keine zentrale Instanz gibt, träfe die Problematik der *Zentralrouter* plötzlich auf alle Server zu. Genau da kommt `Kademlia` ins Spiel. Aber was genau ist `Kademlia` eigentlich?

Laut den Erfindern, *Petar Maymounkov* und *David Mazières*, ist es

> ein Peer-to-peer Nachrichten System basierend auf XOR-Metrik.[1]

Was genau bedeutet das und wie lässt sich eine `XOR`-Metrik für verteilte Datensysteme einsetzen?

[1] Kademlia: Whitepaper: `https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf`, heruntergeladen am: 30.05.2020.

Da die einzelnen Server nicht in der Lage sind, Informationen über das komplette Netzwerk zu speichern oder zu verarbeiten, funktionieren Kademlia-Systeme grundlegend anders. Anstelle der hierarchischen Anordnung der Router ist jedes Mitglied eines Systems gleichgestellt. dabei kümmert sich jedes Mitglied auch nicht um das komplette System, sondern nur um sein direktes Umfeld. Während dies für kleinere Systeme gut funktioniert und vergleichbare Geschwindigkeiten liefert, skaliert es nicht so einfach für grosse Systeme. Genau dafür gibt es die XOR-Metrik.

2.1.1 Distanz

Die XOR-Funktion, die in der Informatik an den verschiedensten Orten auftaucht, wird verwendet, um die Distanz zwischen zwei Zahlen zu berechnen. Die Zahlen repräsentieren dabei Mitglieder im Netzwerk und sind je nach Variante im Bereich $0..2^{160}$(*20 Bytes*) oder $0..2^{256}$(*32 Bytes*). Mit einem so grossen Bereich lässt sich auch das Problem der limitierten IP-Adressen lösen. Auch wenn es kein tatsächlich unlimitiertes System ist, so gibt es doch mehr als genug Adressen.

Wenn mit XOR-Funktionen einfach die Distanz zwischen zwei Zahlen berechnet wird, stellt sich die Frage, wieso nicht einfach die Differenz verwendet wird. Um dies zu beantworten, muss man sich die Eigenschaften der XOR-Funktion etwas genauer anschauen:

1. $xor(x, x) = 0$: Das Mitglied mit seiner eigenen Adresse ist zu sich selbst am nächsten. Mitglieder werden hier als Namen für Server in einem Kademlia-System verwendet.

2. $xor(x, y) > 0$ wenn $x \neq y$: Die Funktion produziert nie negative Zahlen und nur mit zwei identischen Parametern kann man 0 erhalten.

3. $xor(x, y) = xor(y, x)$: Die Reihenfolge der Parameter spielt keine Rolle.

4. $xor(x, z) \leq xor(x, y) + xor(y, z)$: Die direkte Distanz zwischen zwei Punkten ist am kürzesten oder gleich kurz wie die Distanz mit einem zusätzlichen Schritt dazwischen, also einem weiteren Sprung im Netzwerk.

5. Für ein gegebenes x und eine Distanz l gibt es nur genau ein y für das $xor(x, y) = l$ gilt.

Aber wieso genau funktioniert dies? Wieso kann man `XOR`, eine Funktion zur Berechnung der Bit-Unterschiede in zwei Binärzahlen, verwenden, um die Distanz zwischen zwei Punkten in einem verteilten Datensystem zu berechnen?

Um dies zu verstehen, hilft es, sich das System als umgekehrten Baum vorzustellen. Untergeordnet zum zentralen Punkt zu oberst stehen alle Mitglieder im System. Mit jeder weiteren Abzweigung zweier Teilbäume halbiert sich die Anzahl. Man wählt am einfachsten zwei Abzweigungen pro Knoten, da sich damit die Werte direkt als Binärzahlen darstellen lassen, wobei jeder Knotenpunkt einfach eine Stelle in der langen Kette aus 0 oder 1 darstellt. Der ganze Baum sieht dann wie folgt aus[2]:

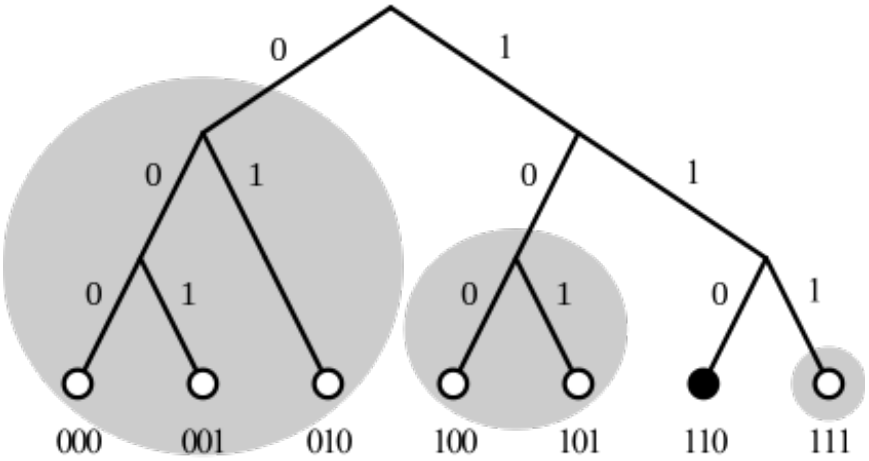

Abbildung 1: Beispielhafte Darstellung eines einfachen Kademlia-Systems

Mit dieser Sicht auf das System beschreibt die `XOR`-Funktion die Anzahl der unterschiedlichen Abbiegungen im Baum und somit die Distanz. Zwar mag es auf den ersten Blick nicht intuitiv wirken, wieso man `XOR` anstatt einfach der Differenz verwendet, allerdings funktioniert die Funktion mit Binärzahlen in einem solchen Baum einiges besser.

2.1.2 Routing-Table

In einem `Kademlia`-System hat jedes Mitglied eine gewisse Anzahl anderer Mitglieder, mit denen es sich verbunden hat. Da `Kademlia` ein sehr umfangreiches, kompliziertes Protokoll und System beschreibt, sollen hier nur

[2]Wikipedia: Kademlia `https://en.wikipedia.org/wiki/Kademlia`, heruntergeladen am: 30.05.2020.

einige zentrale Funktionen besprochen werden, die für diesen ersten Prototypen von `Engine: Orion` relevant sind. Besonders beim `Routing Table` lassen sich einige Abschnitte weglassen, welche zwar für die Optimierung und Skalierung eines Systems wichtig, allerdings für das Analysieren eines einfachen, kleinen Systems wie `Engine: Orion` irrelevant sind.

Einfach formuliert speichert der `Routing Table` die verbundenen Mitglieder. Eine eingehende Nachricht wird dann mithilfe dieser Liste, sowie der `XOR`-Metrik ans richtige Ziel geschickt. Um das System zu optimieren und die Anzahl der benötigten Sprünge klein zu halten, wird ein spezielles System verwendet, um zu entscheiden, welche Mitglieder im `Routing-Table` gespeichert werden sollen:

1. Sehr nahe an sich selbst (in der obigen Darstellung also: wenige Sprünge im Baum) werden alle Mitglieder gespeichert.
2. Je weiter weg sich die Mitglieder befinden, desto weniger werden gespeichert.

Die optimale Anzahl der gespeicherten Mitglieder hängt von den Zielen und Ansprüchen an das System ab. Grundlegend muss man die Frage beantworten, mit wie vielen Unterbäumen Verbindungen gehalten werden sollen. Zwar mag dies etwas abstrakt wirken, es lässt sich aber mit dem eben eingeführten Modell eines umgekehrten Baumes gut erklären:

- In der obersten Ebene trennt sich der Baum in zwei vollständig getrennte Teile, was sich mit jeder weiteren Ebene wiederholt.
- Die einzige Möglichkeit vom einen *Ende* des Baums zum anderen zu kommen, ist über den obersten Knoten. Um also in die andere Hälfte zu kommen, braucht man mindestens eine Verbindungsstelle in der anderen Hälfte.
- Deshalb braucht ein `Routing-Table` nicht nur kurze Verbindungen zu nahen Punkten, sondern auch einige wenige Verbindungen zu Mitgliedern in der anderen Hälfte.
- Mit nur einer weit entfernten Adresse hat man eine Verbindung in *eigene* und die *andere* Hälfte. Hat man zwei solche Verbindungen auf die andere Seite hat man schon Verbindungen in jeden Viertel des Baumes.

- Man muss also entscheiden, wie fein man die andere Hälfte aufteilen will. (Eine genaue Unterteilung bedeutet wenige Sprünge aber grosse `Routing-Tables`, eine grobe Unterteilung genau das Umgekehrte).

Zwar hat ein vollständiges `Kademlia`-System noch komplexere Elemente, wie `k-Buckets`, welche den `Routing-Table` optimieren, allerdings sind diese für die grundlegende Funktionsweise des Systems irrelevant.

Die dynamische Regulation des `Routing-Tables` muss allerdings noch erwähnt werden:

- Sobald die definierte Maximalgrösse erreicht ist, werden keine neuen Verbindungen akzeptiert.
- Zwar können existierende Einträge durch Neue ersetzt werden, allerdings werden dabei nicht alte, sondern inaktive Einträge entfernt. Für ein `Kademlia`-System werden also auch Mechanismen benötigt, um die Zustände aller Verbindungen periodisch zu überprüfen.

2.2 BitTorrent

Dezentralisierung hat viele Vorteile und muss langfristig flächendeckend eingesetzt werden. Aktuell sind die meisten Industrien und Produkte noch nicht so weit. Trotzdem gibt es einige Anwendungen und Gruppen bei denen solche Systeme bereits seit Jahren Verwendung finden.

Beispielsweise im Zusammenhang mit *(mehr oder weniger legalen)* Verbreiten von Materialien wie Filmen oder Musik wird eines der grössten global verteilten Systeme eingesetzt. Natürlich gibt es hunderte von verschiedenen Programmen, Ideen und Umsetzungen, wobei die meisten Nachfolger von `Napster` sind.

Im preisgekrönten Film *The social network* erhält man Einblick in den Lebensstil von `Sean Parker`, einem der Gründer von `Napster`. Es mag überraschen, wie jemand wie Parker, der nur wenige Jahre zuvor mit `Napster` die komplette Musikindustrie in Unruhe gebracht hatte, eine so zentrale Rolle in `Facebook`, einem der zentralisiertesten Megaunternehmen der Welt, einnehmen konnte.

Auch wenn es noch nicht *vollständig* dezentralisiert ist, erlaubte es `Napster` Nutzern, Musik über ein automatisiertes System mit anderen Nutzern zu teilen und neue Titel direkt von den Geräten anderer Nutzer herunterzuladen.

Dabei gab es allerdings immer noch einen zentralen Server, der die Titel sortierte und indizierte.
Napster musste am Ende abgeschaltet werden, nachdem die Klagen der Musikindustrie zu belastend wurden. Auch wenn das Produkt abgeschaltet wurde, liess sich nichts mehr gegen die Idee unternehmen.

Über viele Iterationen und Generationen hinweg wurden die verteilten Systeme immer weiter verbessert, jegliche zentrale Server entfernt und in die Hände immer mehr Nutzer gebracht. Heute läuft ein Grossteil des Austauschs über BitTorrent.
BitTorrent baut auf der gleichen grundlegenden Idee wie Napster auf: Nutzer stellen ihren eigenen Katalog an Medien zur Verfügung und können Inhalte von allen anderen Mitgliedern im System herunterladen. Anders als Napster gibt es bei BitTorrent keine zentrale Komponente, stattdessen findet selbst das Indizieren und Finden von Inhalten dezentralisiert statt[3]. Dafür wird über das Kademlia-System aktiv bekannt gegeben, wer welche Inhalte zur Verfügung stellt, wobei einzelne Mitglieder speichern können, wer die gleichen Inhalte anbietet. Neben Dezentralisierung und Sicherheit lassen sich über BitTorrent tatsächlich gute Geschwindigkeiten erreichen, da sich Inhalte von mehreren Anbietern gleichzeitig herunterladen lassen. Da es sich bei BitTorrent eigentlich um ein grosses Dateisystem handelt, lassen sich direkt die SHA1-Hashwerte der Inhalte als Kademlia-Adressen verwenden.

2.3 Tox

Im Sommer 2013 veröffentlichte Edward Snowden schockierende Geheimnisse über massive Spionage Prgogramme der NSA, mit welchen nahezu aller digitaler Verkehr, ohne Rücksicht auf Datenschutz oder Privatsphären mitgelesen, ausgewertet und gespeichert wurde. Nahezu jede Person mit war betroffen und das genaue Ausmass ist bis heute noch schwer greiffbar. Vielen wurde aber klar, dass sichere, verschlüsselte Kommunikation nicht mehr nur etwas für Kriminelle und *Nerds* war, sondern dass jeder Zugang zu verschlüsselter, sicherer und dezentraler Kommunikation haben sollte. In einem Thread auf 4chan kamen wurden viele dieser Bedenken gesammelt und es kam die Idee auf, selbst eine Alternative zu herrkömmlichen Chat Programmen wie Skype zu entwickeln. Aus dieser Initiative heraus entstand Tox, wobei die Namen vieler der ursprünglichen Entwickler bis heute unbekannt

[3]BitTorrent: Mainline DHT: https://www.cs.helsinki.fi/u/lxwang/publications/P2P2013_13.pdf, heruntergeladen am: 4.06.2020.

sind. Damals war das Ziel die Entwicklung einer sicheren Alternative zu Skype zu entwicklen, allerdings hat sich der Umfang des Projekts inzwischen ausgeweitet. Im Zentrum der Arbeiten steht das `Tox Protocol`, welches dann von verschiedenen, unabhängigen Programmen umgesetzt wird. Zwar ist Chat weiterhin eine zentrale Funktion, es wird aber auch Video- und Audiokommunikation sowie Filesharing gearbeitet.

Basierend auf der bekannten `NaCl`-Bibliothek[4] wird die gesamte Kommunikation über das `Tox Protocol` [5] zwingend End- zu Endverschlüsselt. Intern wird ein dezentrales Routing System basierend auf Kademila verwendet, mit welchem Kontakt zwischen Nutzern (Freunden) aufgebaut wird. Während im Kademila Whitepaper Addressen mit einer Länge von 20 Bytes definiert werden, nutzt `Tox` 32 Bytes. Dies vereinfacht die Verschlüsselung stark, da `NaCl` Schlüssel verwendet, welche ebenfalls 32 Bytes lang sind. Nebst der eingesparten Verhandlung von Schlüsseln und der zusätzlichen Kommunikation bindet diese Idee die Verschlüsselung direkt stärker in das Routing System ein, denn es werden keine zusätzlichen Informationen zum Verschlüsseln einer Nachricht gebraucht und sie kann direkt mit der Addresse des Ziels verschlüsselt werden.

Es ist allerdings wichtig festzustellen, dass `Tox` Kademila lediglich als Router verwendet. Kontakt zwischen zwei Nutzern wird komplett dezentral hergestellt, sobald diese sich allerdings gefunden haben wechseln zu einer direkten Kommunikation über UDP. Zwar erlaubt diese zweiteilung der Kommunikation schnellen Datenverkehr sobald sich zwei Nutzer gefunden haben (so ist beispielsweise Video- und Audiokommunikation möglich), es kommen aber auch einige neue Probleme auf:

- Anders als beispielsweise im Darknet über das Onion-Routing von Aussen klar erkennbar, mit wem jemand kommuniziert. Natürlich ist der Inhalt weiterhin verschlüsselt, aber ein solches System setzt in erster Linie auf Sicherheit und Geschwindigkeit und nicht auf Anonymität.

- Auch muss man bedenken, dass nicht jedes Gerät im Internet in der Lage ist direkte Verbindungen mit jedem anderen Gerät aufzubauen. Besonders Firewalls können schnell zu Problemen führen. Um den

[4] `NaCl` Verschlüsselungs Bibliothek: `https://nacl.cr.yp.to/`, heruntergeladen am: 22.09.2021.

[5] Tox Protokoll Spezifikationen: `https://toktok.ltd/spec.html`, heruntergeladen am: 22.09.2021.

Aufwand für die Nutzer zu minimieren wird `UDP hole punching` [6] verwendet, allerdings existieren auch dafür gewisse Kriterien und Probeleme.

Das `Tox Protocol` bietet eine einheitliche Spezifikation mit der eine grosse Auswahl an Problemen gelöst werden können. Wer eine sichere, dezentrale Alternative zu Whatsapp sucht könnte an `Tox` gefallen finden. Seit einigen Jahren gibt es aber Bedenken über die Sicherheit und aktuelle Richtung des Projekts, sowie Berichte von internen Konflikten, besonders im Zusammenhang mit Spendengeldern.

3 Konzept

Tatsächlich wird hier hauptsächlich der zweite Teil einer grösseren Arbeit beschrieben. Bereits im Abschnitt **Modularität** wurde der erste Teil dieser Arbeit analysiert. Aus den Erfahrungen und Ideen während der ersten Entwicklungsphase wurden in dieses, verbesserte Konzept eingearbeitet. Eine der zentralsten Feststellungen ist die Frage der Komplexität:

- In der Welt der angewandten Wissenschaften geniesst die Informatik einen besonderen Platz. Während die Ingeneurswissenschaften oftmals mit der Informatik verglichen werden, so gibt es doch eine zentrale Unterscheidung:

4 Prozess

4.1 Modularität

Tatsächlich beschreibt dieses Dokument bereits den zweiten Versuch, die zweite Iteration eines dezentralen Datensystems. Da während dieser ersten Entwicklungsphase viele Lektionen gelernt wurden, ist es wichtig die Ideen und die Umsetzung genau zu analysieren. Zwar unterscheiden sich die Ziele und Methoden der beiden Ansätze stark, gewisse Konzepte und einige Programme lassen sich für die aktuelle Zielsetzung genau übernehmen.

Als erstes ist es wichtig, die Zielsetzung des Systems, welches hier einfach als "Modularer Ansatz" bezeichnet wird, zu verstehen und die damit entstandenen Probleme genau festzuhalten.

[6] Wikipedia: UDP Hole punching: `https://en.wikipedia.org/wiki/Hole_punching_(networking)`, heruntergeladen am: 24.09.2021.

- Modularität
  Wie der Name bereits verrät, ging es in erster Linie um die Modularität. Ziel war also eine Methode zur standardisierten Kommunikation, durch welche dann beliebige Komponenten an ein grösseres System angeschlossen werden können. Mit einigen vorgegebenen Komponenten, die Funktionen wie das dezentrale Routing und lokales Routing abdecken, können Nutzer für ihre Anwendungszwecke passende Programme integrieren.

- Offenheit
  Sobald man den Nutzern die Möglichkeit geben will, das System selbst zu erweitern und zu bearbeiten, muss man quasi zwingend open-source Quellcode zur verfügung stellen.

Die grundlegende Idee war die selbe: *Die Entwicklung eines dezentralen vielseitig einsetzbaren Kommunikationsprotokoll.* Da allerdings keine einzelne Anwendung angestrebt wurde, ging es stattdessen um die Entwicklung eines vollständigen Ökosystems und allgemein einsetzbare Komponenten.

Im nächsten Abschnitt sollen einige dieser Komponenten und die Entscheidungen die zu ihnen geführt haben beschrieben werden. In einem weiteren Abschnitt sollen dann die Lektionen und Probleme dieses erster ersten Entwicklungsphase besprochen werden.

4.1.1 Shadow

Zwar übernahm die erste Implementierung des verteilten Nachrichtensystems, Codename `Shadow` weniger Funktionen als die aktuelle Umsetzung, für das System als ganzes war das Programm aber nicht weniger wichtig. Der Name lässt sich einfach erklären: für normale Nutzer sollte das interne Netzwerk niemals sichtbar sein und sie sollte nie direkt mit ihm interagieren müssen, es war also quasi *im Schatten.* Geschrieben in `Elixir` und mit einem `TCP`-Interface konnte `Shadow` sich mit anderen Instanzen verbinden und über eine rudimentäre Implementierung des `Kademlia`-Systems Nachrichten senden und weiterleiten. Um neue Verbindungen herzustellen wurde ein speziell Entwickeltes System mit so genannten *Member-Files* verwendet. Jedes Mitglied eines Netzwerks konnte eine solche Datei generieren, mit welchen dann beliebige andere Instanzen beitreten konnten.

Sobald eine Nachricht im System am Ziel angekommen war, wurde sie über einen `Unix-Socket` an den nächsten Komponenten im System, meistens

also `Hunter` weitergegeben. Dies geschah nur, wenn das einheitlich verwaltete Registrierungssystem für Personen und Dienste, eine Teilfunktion von `Hunter`, ein treffendes Resultat lieferte. Ansonsten wurde der interne Routing-Table verwendet. Dieser bestand aus einer Reihe von Prozessen, welche selbst auch direkt die `TCP`-Verbindungen verwalteten.

4.1.2 Hunter

Während `Shadow` die Rolle des verteilten Routers übernimmt, ist `Hunter` der lokale Router. Es geht bei `Hunter` also nicht darum, Nachrichten an andere Mitglieder des Netzwerks zu senden, sondern sie an verschiedene Applikationen auf der gleichen Maschine zu senden. Anders als `Shadow` wurde `Hunter` komplett in Rust entwickelt und liess sich in zwei zentrale Funktionen aufteilen:

- Zum einen diente das Programm als Schnittstelle zu einer einfachen *Datenbank*, in diesem Fall eine `JSON-Datei`. Dort wurden alle lokal aktiven Adressen und die dazugehörigen Applikationen gespeichert. Ein Nutzer der sich beispielsweise über einen Chat mit dem System verbindet wird dort mit seiner Adresse oder seinem Nutzernamen und dem Namen des Chats eingetragen. Wenn dann von einem beliebigen anderen Punkt im System eine Nachricht an diesen Nutzer kommt, wird der passende Dienst aus der Datenbank gelesen. All dies lief durch ein *Command Line Interface*, welches dann ins Dateisystem schreibt.
- Das eigentliche Senden und Weiterleiten der Nachrichten war nicht über ein kurzlebiges Programm möglich, da dafür längere Verbindungen existieren müssten.

4.1.3 NET-Script

4.1.4 Interface

4.1.5 Probleme

4.2 Präsentation

5 Produkt

6 Ausblick

6.1 Verifizierung

6.2 Balance