

# Project Orion

DOMINIK KELLER, JAKOB KLEMM

---

# Inhaltsverzeichnis

---

<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>1 Einleitung</b>	<b>5</b>
<b>2 Zielsetzung</b>	<b>7</b>
<b>3 Theorie</b>	<b>9</b>
3.1 Adressen	9
3.2 Zentralisierung	11
3.3 Kademlia	12
<b>4 Prozess</b>	<b>17</b>
4.1 Modularität	17
4.2 Präsentation	21
<b>5 Produkte</b>	<b>23</b>
5.1 Actaeon	23
5.2 Orion	30
5.3 Anwendungen	33
<b>6 Fazit</b>	<b>35</b>
<b>7 Anhang</b>	<b>37</b>
7.1 Recherche	37
7.2 Ausblick	40
7.3 Code	41
7.4 Glossar	42
<b>Abbildungsverzeichnis</b>	<b>45</b>
<b>Literaturverzeichnis</b>	<b>47</b>

**Abstract**

Zielsetzung, Schlüsselwörter, Absteigende Komplexität.

## Vorwort

Die schriftliche Komponente der Maturaarbeit Project Orion besteht aus drei verschiedenen Teilen. Da die behandelten Themen äusserst komplex und umfangreich sind, verlangen die Abschnitte der Arbeit verschiedenes Vorwissen und einen unterschiedlichen Zeitaufwand. Deswegen wurde die schriftliche Komponente in drei Subkomponenten aufgeteilt, die 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 Hauptteil dieses Dokuments hinaus lesen. Aber wer vollständigen Einblick in die Errungenschaften und Konzepte erhalten will, muss gewisses Vorwissen und genügend Zeit mitbringen. Die drei folgenden Komponenten sind:

- **Schriftlicher Kommentar:** Im Hauptteil dieses Dokuments findet sich eine klassische Besprechung der Arbeit. Begonnen mit der Zielsetzung und der Vision, bis zur Analyse des Produkts und einem Ausblick in die Zukunft, bekommt man schnell einen guten, aber eher oberflächlichen Einblick in das Projekt. Wer nur über die Vision und den aktuellen Stand wissen will, muss nicht darüber hinaus, aber verschiedene Konzepte sind damit noch nicht abgedeckt.
- **Dokumentation:** Im Anhang und im Kapitel *Theorie* befinden sich zusätzliche Kapitel, welche einen tieferen Einblick in die Gedanken und die schlussendliche Umsetzung der vorgestellten Produkte geben. Für diese Abschnitte wird allerdings gewisses technisches Know-How verlangt.
- **Code:** Neben der schriftlichen Dokumentation des Projekts existiert eine weitere Form der Dokumentation. Nahezu jede Funktion und jedes Modul über die verschiedenen Programme und Bibliotheken sind dokumentiert. Diese Dokumentationen befinden sich nicht in einem klassisch strukturierten Dokument, stattdessen ist die Code-Dokumentation online über automatisch generierte Rust-Dokumentation zugänglich. Wer den Code von Project Orion verwenden will, wird sich dort gut zurecht finden.

Da besonders bei englischen Begriffen das Gendern mit einem Doppelpunkt, Bindestrich oder Stern den Textfluss zu stark beeinflussen könnte, wurde die Entscheidung getroffen, während des gesamten Texts lediglich das generische Maskulin zu verwenden, welches allerdings jedes Mal die inklusiveren Formen repräsentieren soll.

## Kapitel 1

---

# Einleitung

---

Am 29. Oktober 1964 wurde von der Universität Kaliforniens aus folgende Nachricht an das 500 Kilometer entfernte Stanford geschickt: **LO**.

1964 rechnete niemand mit der fundamentalen Änderung unserer Lebensweise, die mit dieser einfachen Nachricht in Bewegung gebracht wurde. Eigentlich hätte die erste Nachricht über das **ARPANET** im Jahre 1964 **LOGIN** heissen sollen, doch das Netzwerk stürzte nach nur zwei Buchstaben ab. Ob dies als schlechtes Omen für die Zukunft hätte gewertet werden sollen, bleibt eine ungeklärte Frage. Aber das Internet ist hier und es ist so dominant wie noch nie zuvor. Jetzt ist es in der Verantwortung jeder neuen Generation auf diesem Planeten, mit den unglaublichen Möglichkeiten richtig umzugehen und die Vielzahl an bevorstehenden Katastrophen und Gefahren zu navigieren.

Ohne das Internet wäre die Welt, wie wir sie kennen, nicht möglich. Unsere Arbeit, Kommunikation und unser Entertainment sind nicht einfach nur abhängig von der enormen Interkonnektivität des Internets, ohne sie würden ganze Industrien und Bereiche unserer Gesellschaft gar nicht erst existieren. Das Internet hatte einen selbstverstärkenden Effekt auf sein eigenes Wachstum. Der um ein Vielfaches schnellere Datenaustausch und die enorme Interkonnektivität führten dazu, dass jede neue Innovation und jede neue Plattform noch schneller noch mehr User erreichte und auf immer unvorstellbarere Grössen anwuchs.

Das ist grundsätzlich nichts Schlechtes. Das Internet hat eine unvorstellbare Menge an Vermögen, Geschwindigkeit und Bequemlichkeit für uns alle geschaffen und wir haben unsere Gesellschaftsordnung daran ausgerichtet. Aber man muss sich fragen, ob wir manche der Schritte nicht doch überstürzt haben. Im Namen des Wachstums und aus **FOMO** (*Fear Of Missing Out*) wurden Technologien für die Massen zugänglich, die eigentlich nie für solche Grössenordnungen entwickelt wurden. Denn sobald die immer höheren Erwartungen an teils unglaublich fragile Systeme nicht mehr erfüllt werden, kommt es schnell zur Katastrophe. Und durch unsere Abhängigkeit von diesen Systemen steht bei einem solchen Szenario nicht nur der Untergang einiger Produkte oder einzelner Firmen bevor, nein, es könnte zum Kollaps ganzer Länder oder Gesellschaften kommen.

Egal wie sicher und zuverlässig unsere *öffentliche* Infrastruktur auch scheinen mag, es lassen sich doch schnell Risse im System erkennen. Oftmals handelt es sich dabei nicht um einfache *Kleinigkeiten*, *Meinungsverschiedenheiten* oder *Kontroversen*, sondern um grundle-

gende Designfehler oder fundamentale Limitierungen. Besonders das Adress-System des Internets ist seinen Grenzen nahe, und die einfachste Lösung scheint immer stärkere Zentralisierung. Doch damit entstehen nebst den ideologischen Bedenken auch technische Probleme, für welche wiederum neue Lösungen gefunden werden müssen. Schlussendlich führt dies zu einer konstant steigenden Komplexität und zu einem System, welches selbst mit dem nötigen Know-How kaum noch durchschaubar ist. Dadurch ist es nicht nur immer schwerer, neue Applikationen zu entwickeln und diese in existierende Systeme zu integrieren, sondern es wird auch mit jeder neuen Abhängigkeit, jedem Protokoll und jeder Abstrahierung schwerer, Aussenstehenden einen Einblick in Errungenschaften und Neuheiten zu geben. Zwar existieren einheitliche, weit verbreitete Lösungen, mit welchen sich einfacher arbeiten lässt, allerdings werden diese meist von grossen Unternehmen angeboten und leiden unter den bereits erwähnten Problemen der Zentralisierung.

Die Situation scheint also ziemlich hoffnungslos. Was lässt sich tatsächlich noch unternehmen, ohne dabei an gigantische Unternehmen, veraltete Standards oder unnötigen Aufwand gebunden zu sein?

Es existieren bereits verschiedene Projekte, die Fortschritte im Zusammenhang mit *dezentralen Datensystemen* machen. Die grundlegende Idee ist dabei immer dieselbe:

Anstelle einer zentralen Instanz übernimmt jedes Mitglied des Netzwerks einen Teil der ganzen Arbeit, die Mitglieder kommunizieren direkt miteinander.

Allerdings werden viele dieser dezentralen Technologien für illegale Zwecke wie Film-Piraterie eingesetzt, weswegen Unternehmungen in diesen Gebieten oftmals einen schlechten Ruf bei uninformierten Personen geniessen. Aber die Einsatzmöglichkeiten für dezentrale Datensysteme gehen weit über *file sharing* hinaus. Wer aber von den fortgeschritteneren Funktionen dezentraler Systeme profitieren will, muss meistens mit mehr Aufwand rechnen und viel Know-How mitbringen.

# Zielsetzung

---

Die Vorteile dezentraler Systeme sind vielseitig, allerdings gibt es, aus offensichtlichen Gründen, wenig Interesse von grossen Firmen oder Initiativen in diesem Gebiet zu arbeiten. Tatsächlich öffnen dezentrale Systeme viele neue Möglichkeiten:

- Moderne Videospiele haben meist starke Limitierungen, wenn es um die Anzahl gleichzeitig aktiver Spieler geht. Diese existieren allerdings hauptsächlich, weil die zentralen Server, welche die Verbindungen zwischen den Spielern möglich machen, nicht mehr Nutzer verarbeiten können. Dezentrale Systeme könnten diese Limits nahezu komplett aufheben.
- Bei den meisten sozialen Interaktionen zwischen Personen über digitale Plattformen geht es eigentlich um den Kontakt. Die Plattform agiert dabei nur als Vermittler und ist nicht zwingend für die Aktivität. Damit wären auch soziale Netzwerke oder Chat-Dienste für dezentrale Alternativen offen.
- Content Addressing erlaubt schnellere, effizientere Möglichkeiten zum Teilen von Dateien, wobei sich die Prinzipien für viel mehr als nur gestohlene Filme einsetzen lassen.
- Auch wenn digitale Dienste oftmals als Methode zum Datenaustausch zwischen Menschen gesehen werden, so kommunizieren schlussendlich nur Maschinen miteinander. Seine es Datenbanken, Austauschplattformen oder eine Vielzahl anderer Möglichkeiten, die Chancen für dezentrale Systeme rund um automatisierte Plattformen und Systeme sind nahezu unbegrenzt.

An Möglichkeiten fehlt es also in diesem Gebiet nicht. Auch existieren bereits wichtige Unternehmungen, welche grosse Fortschritte in verschiedene Richtungen machen. Einige dieser Projekte wurden im Anhang genauer angeschaut. Welche Rolle kann Project Orion übernehmen?

- Da das Entwickeln einer vollständigen, getesteten und integrierten Applikation im gegebenen Zeitrahmen nicht möglich war, sollen stattdessen verschiedene, beispielhafte Anwendungen entwickelt werden. Diese sollen die technischen Vorteile dezentraler Systeme sichtbar machen und dabei einen sinnvollen Nutzen haben. Ziel ist dabei nicht eine fertige Alternative zu Google, Facebook oder ähnlichen, sondern

um die Entwicklung von Prototypen, die die technischen Möglichkeiten aufzeigen sollen.

- Um die Entwicklung verschiedener Applikationen möglichst einfach zu machen, soll ein dezentrales Datensystem als einheitliche Komponente entwickelt und veröffentlicht werden.
- Als Anwendungen sollen die folgenden drei Systeme ein möglichst grosse Auswahl der möglichen Applikationen zeigen:
  - Ein einfacher Chat soll direkte, manuelle Kommunikation zwischen Nutzern erlauben.
  - Mit einer eigenen, spezialisierten Programmiersprache sollen die Möglichkeiten von Maschine-Maschine Interaktionen und allgemeiner Automatisierung gezeigt werden. Dazu soll die Programmiersprache allgemein in das dezentrale Netzwerke eingebunden werden und damit geschriebene Applikationen sollen somit auch direkt mit Nutzern kommunizieren können.
  - Mit einem Videospiel als zentrale Demo soll gezeigt werden, mit wie wenig Aufwand dezentrale Technologien sichtbare Vorteile in Videospielen und anderen Echtzeit-Applikationen bringen können.



# Theorie

---

Da es sich bei dezentralen Datensystemen um ein sehr spezifisches, komplexes Thema handelt, sollen als erstes einige der theoretischen Grundlagen erklärt werden. Dabei soll es nicht nur um dezentrale Systeme an sich gehen, es sollen auch einige der bereits erwähnten Probleme detaillierter angesprochen werden. Dafür müssen einige der technischen Details des aktuellen Internets angesprochen werden. Dabei liegt ein Schwerpunkt auf dem Adress- und Routing-System, welches nicht nur viele Probleme aufweist, sondern zu dem auch verschiedene gute Alternativen zur Auswahl stehen.

### 3.1 Adressen

Das Internet erlaubt einfache, standardisierte Kommunikation zwischen Geräten aller Art. Egal welche Funktion oder Form sie auch haben mögen, es braucht nicht viel, um ein Gerät mit dem Internet zu verbinden. Nebst den benötigten Protokollen, hauptsächlich TCP und UDP, wird eine IP-Adresse als eindeutige Identifikation benötigt. Während vor dreissig Jahren wunderbare Systeme und Standards geschaffen wurden, welche seither die Welt grundlegend verändert haben, gibt es doch einige fundamentale Probleme und Limitierungen.

#### 3.1.1 IP-V4

$4'294'967'296$ . So viele IP-V4-Adressen wird es jemals geben. [1] IP-V4-Adressen werden für jedes Gerät benötigt, das im Internet kommunizieren will und dienen zur eindeutigen Identifizierung. Aktuell wird die vierte Version (V4) verwendet. In einer Wirtschaft, in der unendliches Wachstum als letzte absolute Wahrheit geblieben ist, kann ein solch hartes Limit verheerende Folgen haben. Besonders wenn die limitierte Ressource so unendlich zentral für unser aller Leben ist, wie kaum etwas Anderes. Mit IP-V6 wird zurzeit eine Alternative angeboten, die solche Limitierungen nicht hat. Aber der Wechsel ist eine freiwillige Entscheidung, zu der nicht nur alle Betroffenen bereit sein müssen, sondern für die auch jede einzelne involvierte Komponente die neue Technologie unterstützen muss.

Für jeden Einzelnen kann dies verschiedene Konsequenzen haben:

- Die Preise der Internetanbieter und Mobilfunkabonnemente werden wahrscheinlich langfristig steigen, sobald die erhöhten Kosten für neue Adressen bis zum Endnutzer durchsickern.
- Ein technologischer Wandel wird langfristig von Nöten sein, welcher Jeden dazu zwingt, auf neue Standards umzusteigen. Eine solche Umstellung wird den häufigen Problemen grossflächiger technischer Umstellungen nicht ausweichen können.

### 3.1.2 Routing

Jedes Gerät im Internet ist über Kabel oder Funk mit jedem anderen Gerät verbunden. Da das Internet aus einer Vielzahl von Geräten besteht, wäre es unmöglich, diese direkt miteinander zu verbinden. Daher lässt sich das Internet besser als *umgekehrte Baum-Struktur* darstellen:

- Ganz unten finden sich die Blätter, die Abschlusspunkte der Struktur. Sie stellen die *Endnutzengeräte* dar. Jeder Server, PC und jedes iPhone. Hier ist es auch wichtig festzustellen, dass es in dieser Ansicht des Internets keine magische *Cloud* oder ferne Server und Rechenzentren gibt. Aus der Sicht des Netzwerks sind alle Endpunkte gleich, auch wenn manche für Konsumenten als *Server* gelten.
- Die Verzweigungen und Knotenpunkte über den Blättern, dort wo sich Äste aufteilen, stellen *Router* und *Switches* dar. Hier geht es nicht um Geräte, die sich in einem persönlichen Setup oder einem normalen Haushalt finden. Mit *Switches* sind die Knotenpunkte (*POP-Switches*) der Internet-Anbieter gemeint. Diese teilen eingehende Datenströme auf und leiten die richtigen Daten über die richtigen Leitungen.
- Ganz oben findet sich der Stamm. Während ein normaler Baum natürlich nur einen Stamm hat, finden sich in der Infrastruktur des Internets aus Zuverlässigkeitsgründen mehrere. Von diesen *Zentralroutern* gibt es weltweit nur eine Handvoll und sie sind der Grund für das Problem.

Die Zentralrouter kümmern sich nicht um einzelne Adressen, sondern um Abschnitte von Adressen, auch *Address Spaces* genannt. [2] An den zentralen Knotenpunkten geht es nicht um einzelne Server oder Geräte, zu denen etwas gesendet werden muss, stattdessen wird eher entschieden, ob gewisse Daten beispielsweise von Frankfurt a. M. aus nach Ost- oder Westeuropa geschickt werden.

Im Laufe der Jahre wurden die grossen Abschnitte von Adressen immer weiter aufgeteilt. Internet-Anbieter und grosse Firmen können diese Abschnitte untereinander verkaufen und aufteilen. Und jede Firma will natürlich ihren eigenen Abschnitt, ihren eigenen Address Space. Für die Firmen hat dies viele Vorteile, beispielsweise müssen weniger Parteien beim Finden des korrekten Abschnitts involviert sein. Aber für die Zentralrouter bedeutet es eine immer grössere Datenbank an Zuweisungen. Dieses Problem geht so weit, dass die grossen *Routingtables* inzwischen das physikalische Limit erreichen, das ein einzelner Router verarbeiten kann.

## 3.2 Zentralisierung

Neben den ideologischen Fragen und Sicherheitsbedenken gibt es praktische Probleme in der Art, wie moderne Internet-Dienste implementiert sind. Abhängig von politischen Einstellungen könnte dieser Abschnitt stark in die Länge gezogen werden, allerdings soll der Schwerpunkt hier nur auf zwei Aspekten liegen, nämlich dem Datenschutz und der Abhängigkeit.

### 3.2.1 Datenschutz

*Wenn man nicht für etwas zahlt, ist man das Produkt.* Nach dieser Idee ist man für ziemlich viele Firmen ein Produkt. Doch leider muss man realisieren, dass man selbst bei kostenpflichtigen Diensten als Produkt gesehen wird. Denn das Internet hat einen neuen Rohstoff zur Welt gebracht. Wer viele Daten über Menschen besitzt, bekommt binnen kürzester Zeit Macht. In ihrer einfachsten Funktion werden Daten für personalisierte Werbung eingesetzt. [3] Damit lassen sich Werbungen zielgerichtet an Konsumenten schicken und der Umsatz, sowohl für Firmen als auch für Anbieter, optimieren.

Werbung ist mächtig und hat einen grossen Einfluss auf den Markt. Aber letztendlich lassen sich damit lediglich Konsumenten zu Käufen überzeugen oder davon abbringen. Wenn man dies mit dem tatsächlichen Potential in diesen Daten vergleicht, merkt man schnell, wie viel noch möglich ist. Denn die Daten, die sich täglich über Nutzer im Internet anhäufen, zeigen mehr als unser Kaufverhalten. Von Echtzeit-Positionsupdates, Anrufen und Suchanfragen bis hin zu privaten Chats und unseren tiefsten Geheimnissen, sind Nutzer meist überraschend unvorsichtig im Umgang mit digitalen Werkzeugen.

Während man davon ausgehen muss, dass Firmen, deren Haupteinnahmequelle Werbung ist, unsere Daten sammeln und verkaufen, gibt es eine Vielzahl an anderen Firmen, die ebenfalls unsere Daten sammeln, obwohl man von den meisten dieser Firmen noch nie gehört hat. Die Liste der potentiellen Mithörer bei unseren digitalen Unterhaltungen ist nahezu unendlich: Internet-Anbieter, DNS-Dienstleister, CDN-Anbieter, Ad-Insertion-Systeme, Analytics-Tools, Knotenpunkte & Datencenter, Browser und Betriebssysteme.

Aus dieser Tatsache heraus lassen sich zwei zentrale Probleme formulieren:

- Selbst für die einfachsten Anfragen im Internet sind wir von einer Vielzahl von Firmen und Systemen abhängig. Dieses Problem wird noch etwas genauer im Abschnitt Abhängigkeit besprochen.
- Wir haben weder ein Verständnis von den involvierten Parteien noch die Bereitschaft, Bequemlichkeit dafür aufzugeben.

### 3.2.2 Abhängigkeit

In einem fiktionalen Szenario erklärt *Tom Scott* auf seinem YouTube-Kanal, was passieren könnte, wenn eine einzelne Sicherheitsfunktion beim Internetgiganten Google fehlschlagen würde. [4] In einem solchen Fall ist es natürlich logisch, dass es zu Problemen bei den verschiedensten Google-Diensten kommen würde. Aber schnell realisiert man, auf wie vielen Seiten Nutzer die *Sign-In with Google* Funktion benutzen. Und dann braucht es

nur eine böswillige Person um den Administrator-Account anderer Dienste und Seiten zu öffnen, wodurch die Menge an Sicherheitsproblemen exponentiell steigt.

Aber es muss nicht immer etwas schief gehen, damit die Probleme entstehen. Sei es politische Zensur, *Right to Repair* oder *Net Neutralität*, die grossen Fragen unserer digitalen Zeit sind so relevant wie noch nie.

Während die enorme Abhängigkeit als solche bereits eine Katastrophe am Horizont erkennen lässt, gibt es noch ein konkreteres Problem: Den Nutzern (*den Abhängigen*) ist ihre Abhängigkeit nicht bewusst. Wenn sie sich ihren Alltag ohne Google oder Facebook vorstellen, denken sich viele nicht viel dabei. Weniger *lustige Quizfragen* oder Bilder von Haustieren, aber was könnte den schon wirklich Schlimmes passieren?

Während es verständlich ist, dass das Benutzen von Google von Google abhängig ist, so versteht kaum jemand, wie viel unserer täglichen Aktivitäten von Diensten und Firmen abhängen, die selbst wieder von Google abhängig sind. Seien es die Facebook-Server, durch welche keine Whatsapp-Nachrichten mehr geschickt werden könnten, oder die fehlerhafte Konfiguration bei Google, durch welche manche Kunden die Temperatur ihrer Wohnungen auf ihren Nest Geräten nicht mehr anpassen könnten [5], das Netz aus internen Verbindungen zwischen Firmen ist komplex und undurchschaubar. Und das nicht nur für die Entnutzer, da oftmals die Firmen selbst von kleinsten Problemen anderer Dienste überrascht werden können. Der wirtschaftliche Schaden solcher Ausfälle ist unvorstellbar, aber als noch wichtiger muss die zerstörende Wirkung dieser unvorhergesehenen, scheinbar entfernten Problemen auf Millionen von Menschen angesehen werden.

### 3.3 Kademlia

Tatsächlich existieren bereits Ansätze, um gegen die angesprochen Probleme vorzugehen. Eines dieser Projekte wurde im Laufe der Zeit als vielversprechende Option bekannt. Unter dem Namen *Kademlia* wurde bereits 2002 in einem Whitepaper ein vielseitig einsetzbares System beschrieben, welches seither in verschiedensten Gebieten Verwendung gefunden hat. [6]

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, träge die Problematik der *Zentralrouter* plötzlich auf alle Server zu. Genau da kommt *Kademlia* ins Spiel. Laut den Erfindern, *Petar Maymounkov* und *David Mazières*, ist *Kademlia*

ein Peer-to-peer Nachrichten System basierend auf XOR-Metrik.

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

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 aber nicht um das komplette System, sondern nur um sein direktes Umfeld. Während dies für kleinere Systeme gut funktioniert und Geschwindigkeiten liefert, die mit zentralisierten Systemen vergleichbar sind, skaliert es nicht so einfach für grosse Systeme. Genau dafür gibt es die XOR-Metrik.

### 3.3.1 Distanz

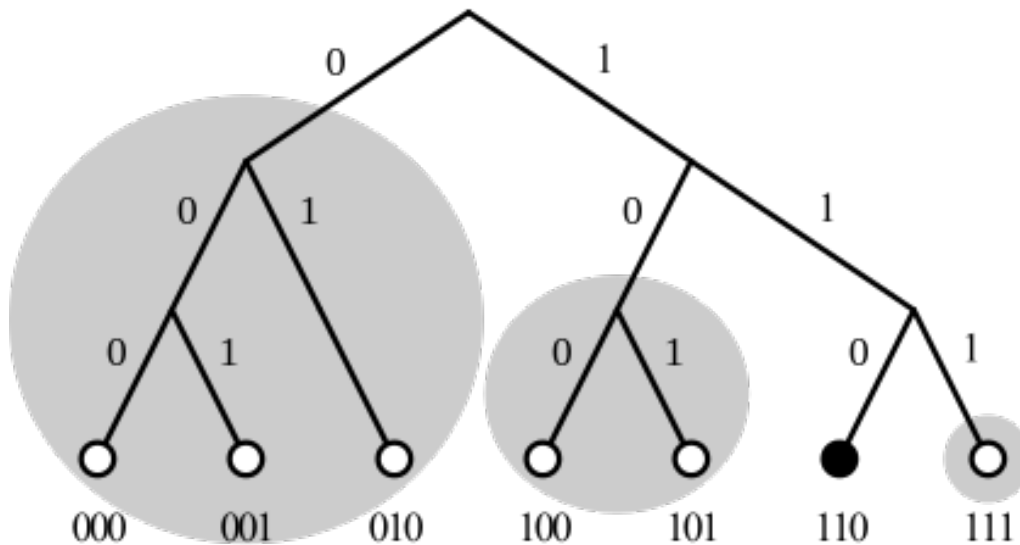
Die XOR-Funktion, die in der Informatik an den verschiedensten Orten auftaucht, wird verwendet, um die Distanz zwischen zwei Zahlen zu berechnen. Die Zahlen repräsentieren dabei Mitglieder im Netzwerk und sind je nach Variante im Bereich  $0..2^{160}$  (20 Bytes) oder  $0..2^{256}$  (32 Bytes). Mit einem 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 kürzer 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 das? 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 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:



Beispielhafte Darstellung eines einfachen Kademlia-Systems, Wikipedia: <https://en.wikipedia.org/wiki/Kademlia>,

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

### 3.3.2 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 einige zentrale Funktionen besprochen werden, die für diesen ersten Prototypen von Project Orion relevant 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 die *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.





# Prozess

---

Tatsächlich erstreckt sich dieses Projekt über zwei Entwicklungsphasen, wobei sich die Methoden und Ziele von der ersten zur zweiten Phase teilweise verändert haben. Besonders die erste Entwicklungsphase soll hier genau beschrieben werden, für die zweite Phase sollen lediglich die Unterschiede angesprochen werden, die tatsächlichen Applikationen werden dann im Kapitel *Produkte* genauer erklärt. Da während der ersten Entwicklungsphase viele wichtige Erkenntnisse entstanden, 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 aber lassen sich für die aktuelle Zielsetzung vollständig übernehmen. Die beiden Phasen werden ab hier einfach *Modularität* und *Präsentation* genannt, da dies die zwei zentralen Eigenschaften der beiden Phasen sind.

### 4.1 Modularität

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.

- **Modularität**  
Wie der Name bereits verrät, ging es in erster Linie um die Modularität. Ziel war also, eine Methode zur standardisierten Kommunikation zu entwickeln, 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. Als zentrale Komponente dieses Systems gibt es ebenfalls ein dezentrales Datensystem, welches die Kommunikation zwischen verschiedenen Geräten übernimmt.
- **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 dieselbe: *Die Entwicklung eines dezentralen, vielseitig einsetzbaren Kommunikationsprotokolls*. Da allerdings keine einzelne Anwendung angestrebt wurde, ging es stattdessen um die Entwicklung eines vollständigen Ökosystems und allgemein einsetzbarer Komponenten.

In den folgenden Abschnitten sollen einige dieser Komponenten und die Entscheidungen, die zu ihnen geführt haben, beschrieben werden. In einem weiteren Abschnitt sollen dann die Lektionen und Probleme dieser erster 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 sollten 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 welcher es beliebigen andere Instanzen beitreten konnte.

Sobald eine Nachricht im System am Ziel angekommen war, wurde sie über einen *Unix-Socket* an den nächsten Komponenten im System weitergegeben. Dies geschah nur, wenn das einheitlich verwaltete Registrierungssystem für Personen und Dienste, eine Teilfunktion von *Hunter*, ein 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. Jedes beliebige Programm, unabhängig von Programmiersprache und internen Strukturen, müsste dann also nur das verhältnismässig Protokoll implementieren und wäre damit in der Lage, mit allen anderen Komponenten zu interagieren. 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 läuft 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 müssen. Deshalb muss Hunter als erstes gestartet werden, wobei das Programm intern für jede Verbindung einen dedizierten Thread startet.

Diese klare Trennung der Aufgaben und starke Unabhängigkeit der einzelnen Komponenten erlaubt ein einheitliches Nachrichtenformat, da die einzelnen Komponenten kein Verständnis andere Komponenten oder die Verbindungen haben müssen.

#### 4.1.3 NET-Script

Eine weitere zentrale Komponente des Systems ist eine eigens dafür entwickelte Programmiersprache, welche mit starker Integration in das restliche System das Entwickeln neuer Mechanismen und Komponenten für das System offener machen sollte. Eine einfache lisp-ähnliche Syntax sollte das Entwickeln neuer Programme einfach und vielseitig einsetzbar machen.

#### 4.1.4 Demo

Da am Ende dieser ersten Entwicklungsphase eine Präsentation stand, musste eine Applikation entwickelt werden, welche einen möglichst einfachen, grafischen Einblick in die entstandenen Komponenten liefert. Eine Chat-Demo bietet dabei ein paar wichtige Vorteile:

- Sie zeigt verständlich die Kommunikation zwischen zwei oder mehr Personen.
- Chat Nachrichten sind technisch für das System nicht anspruchsvoll und daher gut für eine Live-Demo geeignet.

Allerdings ist es hierbei wichtig zu bedenken, dass alle Komponenten des damaligen Systems nur auf Servern lief und das Datensystem eine Schnittstelle zwischen verschiedenen Servern ermöglichte. Der Chat müsste aber für Endnutzer zugänglich sein, weswegen eine eigene Komponente dafür benötigt wurde: Websocket.or konnte wie andere Komponenten auch an das restliche System angeschlossen werden und präsentierte dann eine nach Aussen erreichbare Chat-Webseite. Das Design dieser Website wird wohl kaum viele Preise gewinnen, liefert aber trotzdem eine schlichte, minimalistische Chat-Umgebung für Nutzer.

## Engine: Orion - Chat Demo

Dezentrale Chat Demo  
Hello World  
Modularer Chat Client des Orion Systems

Message SEND

Engine: Orion Chat-Website, intern verbunden mit einem dezentralen Kommunikationssystem.

Nutzer waren mit dieser Applikation in der Lage, mit beliebigen anderen Nutzern zu kommunizieren, solange sie die Adresse von einem Chat-Server kannten, der das passende Interface anbot.

#### 4.1.5 Probleme

Die oben beschriebene Architektur hat viele verschiedene Vorteile, allerdings ist sie nicht ohne Probleme. Grundsätzlich geht es bei jedem Programm darum, Probleme zu lösen. Eine der zentraler Ideen war die Modularität, welche es Nutzern erlauben soll, die verschiedenen Komponenten des Systems einfach zu kombinieren. Und auch wenn dieses Ziel auf einer technischen Ebene erfüllt wurde, so ist die Umsetzung alles andere als *einfach*. Die Anzahl möglicher Fehlerquellen steigt mit jeder eingebundenen Komponente exponentiell an, und wenn mindestens vier der Komponenten selbst für die einfachsten Demos benötigt werden, kann nahezu alles schiefgehen. Dazu kommt, dass viele Fehler nicht richtig isoliert und verarbeitet würden, weswegen sich die Probleme durch das System weiter verbreiten würden. Während die Umsetzung also ihre eigentlichen Ziele erfüllt hatte, war sie noch weit davon entfernt, für tatsächliche Nutzer einsetzbar zu sein.

Trotzdem wurden die beschriebenen Komponenten vollständig entwickelt, getestet und vorgeführt. Zwar war es umständlich und nur bedingt praktisch einsetzbar, trotzdem war es aber eine technisch neuartige, funktionsfähige Lösung für komplexe und relevante Probleme. Nachdem die erste Entwicklungsphase erfolgreich abgeschlossen wurde, kam allerdings noch ein weiteres Problem auf, welches die folgenden Entscheidungen stark beeinflusst hat. Es ist ein Problem, welches sich auf die grundlegende Natur der Informatik zurückführen lässt:

Anders als in nahezu allen Studienrichtungen, Wissenschaften und Industrien, werden in der Informatik die gleichen Werkzeuge verwendet und entwickelt. Wer die Werkzeuge der Informatik verwenden kann, ist gleichzeitig in der Lage (zumindest bis zu einem

gewissen Grad) neue Werkzeuge zu entwickeln. Diese Eigenschaft erlaubt schnelle Iterationen und viele fortschrittliche Werkzeuge, kommen gleichzeitig neue Probleme auf:

- Neue Methoden und Werkzeuge werden mit unglaublicher Geschwindigkeit entwickelt und verbreitet. Wer also nicht mit den neusten Trends mithält, kann schnell abgehängt werden. Dies macht auch das Unterrichten besonders schwer.
- Natürlich werden die Werkzeuge meistens immer besser und schneller, allerdings kommt es oftmals auch zu einer Spezialisierung. Dies führt schnell zu immer spezielleren, exotischeren Lösungen und unzähligen Unterbereichen und immer kleineren Gebieten. So ist beispielsweise der Begriff *dezentrale Datensysteme*, der zwar ein einzelnes Gebiet genau beschreibt, für Aussenstehende mehrheitlich bedeutungslos und sorgt für mehr Verwirrung als Aufklärung.
- Die immer neuen Gebiete und Gruppen können auch schnell zu Elitismus führen, wodurch es für Anfänger schwer sein kann, Zugang zu finden.

Diese Eigenschaften, besonders bei unseren sehr neuartigen Ideen und Mechanismen, machten es schwer, Aussenstehenden die Funktionen und Konzepte zu erklären. Ohne Vorkenntnisse über Netzwerke und Kommunikationssysteme war es nahezu unmöglich, auch nur die einfachsten Ideen zu erklären oder den Inhalt dieser Arbeit darzulegen. Und selbst mit grossem Vorwissen liessen sich nur die absoluten Grundlagen innerhalb absehbarer Zeit erklären. Das Erklären der theoretischen und technischen Grundlagen würde Stunden in Anspruch nehmen.

Da am Ende dieser Arbeit zwingend eine zeitlich begrenzte Präsentation vor einem technisch nicht versierten Publikum steht, mussten nach dieser ersten Entwicklungsphase gewisse Aspekte grundlegend überarbeitet werden, diesmal mit einem besonderen Fokus auf die *Präsentierbarkeit*.

## 4.2 Präsentation

Auch wenn von der ersten Entwicklungsphase viele Konzepte und sogar einige Umsetzungen übernommen werden konnten, gab es grundlegende Probleme, welche nicht ignoriert werden konnten. Es wurde schnell klar, dass unabhängig von allen technischen Fortschritten eine bessere Art der Präsentation gefunden werden musste. Dabei war es wichtig, die technischen Neuerungen und Besonderheiten nicht zu vergessen. Die Umsetzung der ersten Entwicklungsphase, wie innovativ und attraktiv sie auch wirken mag, ist noch weit davon entfernt, von Endnutzern verwendet oder gar angepasst zu werden. Auch wenn manche der Ideen hier wieder aufgegriffen werden, musste doch ein grösserer Fokus auf die *Präsentierbarkeit* der Fortschritte gelegt werden. Daher wurde die Entscheidung getroffen, die Entwicklung in zwei Bereiche zu unterteilen:

- Ein möglichst vielseitig einsetzbares Nachrichtensystem basierend auf den bereits bekannten Prinzipien wird als Bibliothek für die Anwendungen öffentlich angeboten. Entwickelt in Rust wird Geschwindigkeit und Sicherheit garantiert und es lassen sich möglichst viele Möglichkeiten finden, Integrationen in andere Projekte und Applikationen zu ermöglichen..

- Aufbauend auf diesem Datensystem sollen mit verschiedenen Anwendungen die Vorteile und vielseitige Einsatzmöglichkeiten gezeigt werden. Auch wenn damit die weltverändernde Revolution noch nicht direkt gestartet wird, so wird ein Aspekt angesprochen, welcher in technischen Kreisen oftmals vergessen geht, nämlich die Frage, wie man komplexe Themen und Programme einfachen Nutzern näher bringt.

# Produkte

---

Nun ist es an der Zeit, die Errungenschaften und Produkte des Projektes anzuschauen. Eigentlich müssten auch hier die Errungenschaften der ersten Entwicklungsphase besprochen werden, dies geschah allerdings bereits grösstenteils während der Analyse des Arbeitsprozesses. Wer trotzdem noch genaueres über die erste Entwicklungsphase erfahren will, liest sich am besten den dazugehörigen Bericht durch, oder schaut sich direkt die dabei entstandenen Programme an. [7]

Dieses Kapitel ist in drei Abschnitte geteilt:

- `Actaeon` beschreibt das dezentrale Datensystem und die `Rust`-Bibliothek mit der es sich verwenden lässt. Dazu werden einige der technischen Details angesprochen.
- `Anwendungen` beschreibt die Integration mit der eigenen Programmiersprache sowie die Chat-Anwendung.
- `Orion` beschreibt das Videospiel, welches als zentrale Demo den Titel des gesamten Projekts und einen eigenen Abschnitt erhalten hat.

## 5.1 Actaeon

`Actaeon` stellt das Herzstück des gesamten Projekts dar. Als `Rust` Bibliothek ist es in alle anderen Anwendungen eingebunden und ermöglicht dezentrale Kommunikation unabhängig von der Anwendung.

In der griechischen Mythologie ist Actaeon (auch Aktaion) ein Jäger, der mit seinen Hunden durch den Wald streift und die Göttin der Jagt beim Bad in einer Quelle überrascht. Die Göttin verwandelt Actaeon in einen Hirsch, der daraufhin von seinen eigenen Hunden in Stücke gerissen und gefressen wird. Der Philosoph Peter Sloterdijk, der sich dabei auf Giordano Bruno bezieht, deutet die Geschichte von Actaeon als Gleichnis für die menschliche Suche nach Wahrheit. Ein Blick auf die ganze, göttliche Wahrheit ist für einen Menschen nicht möglich. Wer das versucht, wird vom Jäger zum Gejagten, wird von der Vielzahl alles Seienden, den Hunden, verschlungen. Die Wahrheit ist nur zu finden als Teil einer Einheit, die die gesamte Natur in ihrer ganzen Vielfalt umfasst. So ist Actaeon als Namensgeber für ein Projekt der dezentralen Kommunikation bestens geeignet. [8]

Nahezu alle Applikationen in Project Orion sind in der Programmiersprache Rust geschrieben. Als zentrales Verbindungsstück zwischen allen Applikationen hängen alle Programme von Actaeon ab. Um die Verwaltung der internen Abhängigkeiten einfacher zu machen, wurde Actaeon mit dem offiziellen Rust *Package-Manager* als `crate` veröffentlicht. Dies erlaubt es jedem Rust Entwickler, Actaeon einfach in eigene Programme zu integrieren, dafür ist lediglich eine Zeile Code nötig:

```
actaeon = "0.2.1"
```

Damit wird Actaeon als Abhängigkeit definiert und der Nutzer hat nun vollen Zugang zu allen Funktionen, die Actaeon zu bieten hat. Da es sich hier um eine Bibliothek handelt, welche dann in andere Programme eingebunden werden soll, ist es nicht von Nöten, die interne Funktionsweise vollständig zu verstehen. Hier soll ein Überblick über die Verwendung und die Funktionen gegeben werden. In den folgenden Abschnitten werden dann zusätzlich noch technische Details zur Bibliothek angesprochen, diese sind aber nicht notwendig, um die Bibliothek zu verwenden.

Es lohnt sich aufzulisten, welche Funktionen die Bibliothek übernimmt, und welche weiterhin vom Nutzer verlangt werden. Actaeon übernimmt das folgende:

- Verbindungen (stateful & stateless): Das System entscheidet intern selbständig, wie Nachrichten versendet werden sollen. Entweder werden einzelne Nachrichten an andere Mitglieder geschickt, oder es wird eine langlebige Verbindung aufgebaut, welche für mehrere Nachrichten verwendet werden kann.
- Adressen: Anstelle von IP-Adressen oder UUIDs verwendet das System ein eigenes, vielseitiges Adresssystem. Dieses wird auf verschiedene Arten eingesetzt:
  - Identifikation: Jede Adresse identifiziert ein Objekt (Nutzer oder Thema) eindeutig. Dabei gibt es zwar keinen Mechanismus, um dies zu garantieren, die kleine Wahrscheinlichkeit einer Kollision über die 32 Bytes reicht allerdings aus.
  - Routing: Mit den Adressen lässt sich ebenfalls rechnen, wobei es hauptsächlich um die Kademia XOR-Operation geht. Damit werden Distanzen und die Wege sowie Orte für Nachrichten bestimmt.
  - Verschlüsselung: Dank der Familie der NaCl-Verschlüsselungsbibliotheken ist es möglich, öffentliche und private Schlüssel mit einer Länge von nur 32 bytes zu haben. Damit lassen sich im System Nachrichten automatisch End-zu-End verschlüsseln, ohne einen dedizierten Mechanismus zum Austausch von öffentlichen Schlüsseln zu benötigen.
- Signaling: In Netzwerken ist es immer eine wichtige Frage, wie neue Nutzer beitreten können. Actaeon verlangt dabei lediglich die Kontaktdaten von einem bekannten Mitglied des Systems und kümmert sich dann um den Rest.
- Form: Durch regelmässige Abfragen und Überprüfungen können Mitglieder automatisch neue Mitglieder finden und sie in ihren lokalen Routing-Table einbauen.

Der Nutzer muss sich um einige Aspekte selber kümmern:



- Die Verbindungsdaten für ein beliebiges Mitglied eines Clusters müssen bekannt sein, sodass das System eine erste Verbindung aufbauen kann. Auch wenn gewisse Automatisierung noch möglich ist, wird dieser Schritt niemals vollständig aus den Händen der Nutzer genommen werden können, ohne nicht dabei einen zentralen Registrierungspunkt einzubauen.
- Auch wird aktuell noch von Nutzern verlangt, dass sie öffentlich erreichbare Verbindungsdetails angeben. Das bedeutet, Nutzer müssen sich selbst um Portforwarding und ihre öffentlich zugängliche Adresse kümmern.

In Code ausgedrückt sieht das dann wie folgt aus: (Ein minimales Beispiel für eine Applikation, die Actaeon verwendet.)

```
use actaeon::{
    config::Config,
    node::{Center, ToAddress},
    Interface,
};
use sodiumoxide::crypto::box_;

fn main() {
    let config = Config::new(20, 1, 100, "example.com".to_string(), 4242);
    let (_, secret) = box_::gen_keypair();
    let center = Center::new(secret, String::from("127.0.0.1"), 1234);

    let interface = Interface::new(config, center).unwrap();

    let mut topic = interface.subscribe(
        &"example"
            .to_string()
            .to_address()
            .unwrap()
    );

    let _ = topic.broadcast("hello world".as_bytes().to_vec());
}
```

Nebst der Konfiguration, welche sowohl direkt erstellt, als auch von einer Datei geladen werden kann, muss nur ein `Interface` erstellt werden. Bei diesem Schritt werden intern verschiedene Threads gestartet, und die Initialisierung des Routing-Tables (Bootstrapping) beginnt. Der Nutzer muss sich dabei um nichts kümmern, ausser potentiell aufkommende Fehler handhaben. Das zweite wichtige Element ist dabei das `Topic`, denn die meisten Interaktionen des Nutzers finden über ein solches Thema statt. Für genauere Informationen empfiehlt sich die technische Dokumentation. Als einfache Zusammenfassung lässt sich sagen: Unter einem Thema werden Nachrichten gesammelt, welche laut dem Nutzer zusammengehören. Dabei ist wichtig festzustellen, dass die Wahl der

Themen vollkommen in der Hand des Nutzers liegt, die Bibliothek hat keinen Einfluss darauf. Sobald aber ein solches Thema erstellt wurde, gibt es eigentlich nur zwei wichtige Aktionen:

- Senden: Schickt eine Nachricht an alle Nutzer, welche ebenfalls ein Thema mit gleicher Adresse erstellt haben.
- Empfangen: Hört auf einkommende Nachrichten von beliebigen anderen Mitgliedern zu diesem Thema.

Nur mit diesen beiden Operationen ist die Menge an potentiellen Applikationen nahezu unbegrenzt, denn nahezu alle Interaktionen, Programme und Netzwerke lassen sich mit diesen beiden Befehlen umsetzen. In den nächsten Abschnitten sollen nun einige der technischen Konzepte angesprochen werden. Dabei geht es allgemein um die Funktionen eines Pub/Subs, möglichen Verschlüsselungsmethoden sowie um einige der Besonderheiten der technischen Umsetzung für die Actaeon-Bibliothek.

### 5.1.1 Verschlüsselung

Zwar ist es bei weitem nicht so, dass alle moderne dezentrale Systeme das Internet oder ein ähnliches Austauschsystem als Grundlage verwenden, allerdings ist dies in nahezu allen Fällen, besonders bei den beliebten und weit verbreiteten Fällen. Das Internet ist für fehlende Sicherheit und Gefahren bekannt, daher ist es von Nöten, dass sich jede Applikation selbst um Verschlüsselungen und Sicherheit kümmert. Allerdings ist es wichtig, *die passende Verschlüsselung* zu wählen. Hier wird nun ein Ansatz beschrieben, welcher sich besonders gut für gewisse Kademia-inspirierte Systeme eignet. Dieser Ansatz beruht auf der Verschlüsselungs-Bibliothek NaCl, beziehungsweise deren modernen Abzweig `libsodium`. Während klassische Verschlüsselungsmethoden sehr lange Schlüssel benötigen, gibt es gewisse Kombinationen von Algorithmen, welche mit sehr kurzen Schlüsseln Sicherheit gewährleisten können. Besonders geht es dabei um `curve25519xsalsa20poly1305`, einer Kombination aus den Algorithmen Curve25519, Salsa20 und Poly1305. Während das innere Zusammenspiel dieser Algorithmen sehr komplex ist, ist das Resultat ein Algorithmus, wessen Schlüssel jeweils nur eine Länge von 32 *bytes* haben.

Eigentlich beschreibt Kademia Adressen mit einer Länge von 160 *bits* (20 *bytes*), allerdings ist es ohne grosse Probleme möglich, die Adressen Länge auf 32 *bytes* zu verlängern. Dies erlaubt es uns, die Verschlüsselung und das Adresssystem direkt miteinander zu verbinden. Das mag auf den ersten Blick etwas umständlich und unlogisch wirken, es erlaubt allerdings, ohne weitere Operationen, verschlüsselte Nachrichten an einen Nutzer zu schicken, wobei lediglich dessen Adresse bekannt sein muss. Insgesamt fällt damit viel Komplexität weg und macht das Erstellen, Verifizieren und Finden von Adressen viel einfacher.

### 5.1.2 PubSub

Ein einfaches, aber vielseitig einsetzbares Nachrichtenübermittlungsmuster ist die Idee eines Publish/Subscribe Systems. Ein solches System lässt sich mit nur zwei Aktionen beschreiben:

- Nutzer können ein gewisses Thema abonnieren, das bedeutet, sie folgen einem gewissen Schlüssel und erhalten Nachrichten von diesem.
- Jeder Nutzer kann dann in den Themen, die er abonniert hat Nachrichten schicken. Diese werden dann automatisch an alle abonnierten Nutzer verteilt.

Mit diesen beiden Mechaniken lassen sich die meisten Funktionen in modernen Applikationen beschreiben. Sei es ein Chat- oder Emailsysteem, ein komplexer Datenverarbeitungsmechanismus oder ein Datennetzwerk, alle lassen sich relativ einfach mit diesen beiden Funktionen modellieren.

1. Dezentraler PubSub Offensichtlich kann selbst die beste, fehlerfrei optimierte Implementierung der oben beschriebenen Prinzipien nicht gegen die bereits angesprochenen, fundamentalen Probleme lokal gebundener Programme vorgehen. Daher ist es in einem nächsten Schritt von Nöten, die Ideen hinter dezentralisierten PubSub-Systemen anzuschauen. Das mag im ersten Moment komplex klingen, ist aber tatsächlich unglaublich einfach. Man muss sich lediglich einen PubSub als zwei getrennte Unterkomponenten vorstellen:

- Themen lassen sich vereinfacht als Einträge in einer Datenbank beschreiben. Die Identifikation der Themen ist dabei der Schlüssel, wobei die Abonnenten als dazugehörige Felder ausgedrückt werden können. Oben wurde bereits ein System beschrieben, welches zuverlässig dezentral Daten speichern kann. Wenn man in der beschriebenen Kademia Implementierung die Checksumme des Inhalts mit der Checksumme des Themenschlüssels ersetzt, lässt sich Kademia ohne weitere Veränderungen für einen dezentralen PubSub einsetzen.
- Danach bleibt natürlich noch das Problem der Nachrichtenverbreitung. Dafür gibt es verschiedene Möglichkeiten:
  - Die Nachrichten werden direkt an das zuständige Mitglied gesendet, von dort werden sie weitergeleitet. Vorteilhaft an diesem Konzept ist natürlich, dass die Verwender des Systems unglaublich einfach gehalten werden können. Sie müssen lediglich Nachrichten an eine Adresse schicken, das System kümmert sich dann von alleine um die Weiterverbreitung. Damit geben die Nutzer allerdings auch eine gewisse Kontrolle auf, denn sie können nicht direkt einsehen, an wen ihre Nachrichten verteilt werden. In einer solchen Situation gibt es zusätzlich noch gewisse technische Bedenken im Zusammenhang mit der Verschlüsselung.
  - Andererseits dazu können auch die Aktionen des Abonnierens und Deabonnierens als Nachrichten im System verbreitet werden. Jeder abonnierte Nutzer wird somit also über neue Abonnenten informiert und speichert deren Details lokal. Zwar erhöht dies die Komplexität enorm, erlaubt

aber schnellere Übertragung und genauer Kontrolle über die Auswahl der Abonnenten.

2. Probleme Zwar gibt es viel Gutes über PubSubs als Systemkonzept zu sagen, allerdings müssen auch einige Probleme angesprochen werden:

- Wie bereits eben angesprochen kann es zu gewissen Unklarheiten und Problemen im Zusammenhang mit der Verschlüsselung der Nachrichten in dezentralen Systemen kommen. Da Nachrichten meist über das offene Internet übertragen werden und daher Verschlüsselung nahezu zwingend benötigt wird, muss man sie auch hier berücksichtigen. Wie bereits im Abschnitt zur Verschlüsselung angesprochen, sollen Nachrichten mit dem öffentlichen Schlüssel des Empfängers, welcher auch gleichzeitig die Adresse darstellt, verschlüsselt werden. Beim Durchgehen der oben beschriebenen Architekturen wird ein Problem offensichtlich: Wenn ein Thema ein normaler Empfänger im System ist, muss seine öffentliche Adresse verwendet werden. Allerdings wurde definiert, dass die Adresse eines Themas die Checksumme eines bekannten Schlüssels darstellt. Die Adressen in einem solchen System lassen sich allgemein aber durch einen gegebenen privaten Schlüssel ableiten. Umgekehrt ist das natürlich nicht möglich. Ein geheimer Schlüssel lässt sich nicht aus dem öffentlichen Schlüssel errechnen. Hier wird also eigentlich ein System verlangt, bei welchem der private Schlüssel durch eine Checksumme errechnet wird, wobei der daraus entstehende öffentliche Schlüssel als Adresse verwendet wird. Gleichzeitig darf der private Schlüssel nicht bekannt sein, sonst wäre die gesamte Verschlüsselung sinnlos. Es wird schnell offensichtlich, dass solche Bedingungen nie erfüllt werden können.
- Da es sich bei der Liste der Abonnenten um Daten handelt, welche während der Laufzeit gespeichert und verwaltet werden müssen, bringt man plötzlich eine Vielzahl neuer Probleme ins Spiel. So müssen die Daten repliziert und gesichert werden, da ein einzelnes Mitglied jederzeit unerreichbar sein könnte. Sie müssen verifiziert werden, da man kein Vertrauen in die Mitglieder des Systems haben darf und sie müssen trotz häufiger Änderungen konstant gehalten werden. Das Gebiet der dezentralen oder verteilten Datenbanken alleine ist sehr gross und komplex, wenn man also plötzlich nebst einem Nachrichtensystem ohne verteilte Zustände auch noch eine verteilte Datenbank verwalten muss, übersteigt dies meist die erhoffte Komplexität vieler Projekte.

### 5.1.3 Router

Als zentrales Herzstück des gesamten Actaeon Systems übernimmt der Router zwei wichtige Aufgaben gleichzeitig:

- Das Speichern der bekannten Nodes im System.
- Funktionen zum berechnen der nächsten Ziele für Nachrichten.

Um die Funktionen des Routers zu verstehen, lohnt es sich, den Abschnitt zu Kademlia, oder besser noch das tatsächliche Kademlia Whitepaper zu lesen, denn der Router ist der

Punkt im System, der die meisten der kademlia-spezifischen Funktionen übernimmt. Als erstes ist es wahrscheinlich wichtig zu verstehen, welche Daten im Router überhaupt gespeichert werden. Grundsätzlich wird im Router jede Node gespeichert, von der Daten empfangen werden, oder die durch eine Anfrage gefunden wurde. Allerdings definiert Kademlia gewisse Regeln, nach welchen neu gefundene Nodes zum Routing Table hinzugefügt werden:

- Grundsätzlich werden alte, bereits seit langem bekannte Nodes über neuere bevorzugt, selbst wenn diese ansonsten bessere Eigenschaften (kleine Distanz) hätten. Damit soll verhindert werden, dass das Netzwerk durch eine Vielzahl böswillig generierter Nodes einfach angegriffen werden kann.
- Für jede Node wird ein Status gespeichert, welcher angibt, ob die Node aktuell erreichbar ist. Da sich dies während der Laufzeit ändern kann muss der Status regelmäßig überprüft werden.
- Die Anzahl der gespeicherten Nodes ist umgekehrt proportional zum Abstand zwischen der Node und dem relativen Zentrum (also sich selbst).

Um diese Regeln möglichst einfach umzusetzen, wird der Router intern als binärer Baum dargestellt. Jedes Blatt stellt dabei ein k-Bucket dar, in welchem die tatsächlichen Nodes liegen. Sobald ein solcher Bucket voll ist, kann er halbiert werden und es entstehen zwei neue Blätter. Dies ist allerdings nur auf der, dem Zentrum (also der eigenen Adresse) nahen Seite möglich. Auf der *fernen* Seite können Buckets nicht geteilt werden. Damit wird direkt die Regel, dass die Häufigkeit der Nodes mit steigendem Abstand abnimmt direkt umgesetzt. Sollte es nicht möglich sein, einen Bucket zu teilen, werden neue Nodes entweder vergessen, oder bereits vorhandene Nodes werden überschrieben. Auch ist es wichtig zu verstehen, dass innerhalb der tatsächlichen Buckets nicht nach Distanz, sondern nach Zeit sortiert wird. Da Nachrichten meistens an mehrere Ziele gleichzeitig geschickt werden, um die Auslieferung zu garantieren, wird versucht, zuerst möglichst viele Nodes aus dem gleichen Bucket zu verwenden. Nur wenn diese nicht reichen dürfen Nodes aus anderen Buckets verwendet werden.

Der Router bietet alle nötigen Funktionen direkt in einer öffentlichen API an, in der tatsächlichen Applikation wird er allerdings etwas anders eingesetzt:

- Um die Menge an internen Nachrichten und Abhängigkeiten zu reduzieren, haben verschiedene Punkte im System Zugang zum gleichen Router Objekt.
- Da verschiedene Threads Zugang zu den gleichen Daten brauchen, muss das eigentliche Router Objekt hinter einem Mutex liegen und darf immer nur von einem Thread gleichzeitig bearbeitet werden.

#### 5.1.4 Speicher

Leider ist es meistens nicht möglich, ein so komplexes, umfangreiches Kommunikationssystem aufzubauen, ohne dabei nicht geteilte Zustände zu benötigen. Dabei geht es um Daten, die

- nicht nur für ein Mitglied des Systems relevant sind wie beispielsweise die Router Daten,
- länger Leben müssen als ein Mitglied online sein könnte, daher eine ändernde Netzwerkstruktur überstehen müssen.

Es ist offensichtlich, dass man solche Daten um jeden Preis vermeiden sollte. Da es sich im Actaeon-System *nur* um ein Nachrichtensystem und keine dezentrale Datenbank handelt, existieren tatsächlich nahezu keine geteilten Daten. Als einzige Ausnahme allerdings werden im System `Topic Records` verwendet, auf welche genau diese Problematik zutrifft.

Wenn ein Mitglied ein Thema abonniert, muss der Kontakt mit allen bisherigen Abonnenten hergestellt werden. Dies könnte theoretisch ohne Struktur durch ein einfaches Netzwerkfluten gemacht werden, ein solcher Ansatz wäre aber sehr ineffizient. Actaeon, sowie die meisten Systeme dieser Art, verwendet einen kleinen Eintrag im System, der den neuen Abonnenten über existierende informiert und umgekehrt. Dieser Eintrag kann aber nicht einfach irgendwo liegen, sondern muss genau bei dem Mitglied zu finden sein, zu welchem seine Adresse den kleinsten Abstand hat. Dieses Mitglied muss sich dabei um nichts kümmern, das System verwaltet diese Einträge automatisch. Man muss sich nun aber um die Möglichkeit kümmern, dass das zuständige Mitglied nicht mehr erreichbar ist. Die einfachste Lösung dafür und die, welche aktuell von Actaeon verwendet wird, ist einfach mithilfe einer hohen Replikation dafür zu sorgen, dass hoffentlich immer mindestens ein Mitglied verfügbar ist.

Allerdings kann dies trotzdem zu Problemen führen. Besonders wenn sich die Netzwerkstruktur schnell stark ändert, kann es dazu kommen, dass gewisse Teile des Netzwerks Themen an der falschen Stelle suchen. Im aktuellen Actaeon-System existiert noch keine Lösung für dieses Problem, aber eine mögliche Lösung existiert und deren Umsetzung ist geplant.

## 5.2 Orion

Orion ist die zentrale Anwendung, welche die Möglichkeiten von Actaeon aufzeigen soll. Orion ist ein 3D Spiel, in welchem man ein Flugzeug fliegt und versucht mehr Punkte zu machen als der Gegner. Punkte werden vergeben, wenn das Flugzeug durch Ringe gesteuert wird. Diese Demo trägt denselben Namen wie das Projekt, da es die zentrale Demo ist.

Orion ist komplett in der Rust Programmiersprache geschrieben und baut auf dem Bevy Framework auf. Bevy ist ein sehr neues Framework, welches die 3D programmierung vereinfacht, da es grundlegende Komponente wie Oberflächen und Formen zur verfügung stellt.

### 5.2.1 Integration von Actaeon

Die Integration von Actaeon in Orion stellte sich als ein kleines Problem heraus, da die Daten von Bevy verwaltet werden. Dies ist vonnöten, da die Funktionen, welche das Game definieren von Bevy aufgerufen werden. Damit dies aber mit den Daten von Actaeon funktioniert, muss Actaeon Eigenschaften besitzen, welche das Crate nicht hat. Die Lösung für dieses Problem ist ein Wrapper um Actaeon, welcher die Daten isoliert. Dieser Wrapper kann nun auch die Eigenschaften haben, um sie von Bevy verwalten zu lassen.

### 5.2.2 Bevy

Bevy ist eine neue und intuitive Bibliothek, die besonders für die Programmierung von Games ausgelegt ist. Bevy nimmt einen anderen Ansatz als andere Bibliotheken, denn normalerweise ist der Entwickler dafür zuständig, dass die Daten richtig verwaltet und die Funktionen richtig aufgerufen werden. Diese Last nimmt Bevy dem Programmierer ab und erleichtert es neue Funktionen hinzuzufügen.

Dieser neue Ansatz nennt Bevy ECS, was kurz für Entity Component System steht. Die Daten sind die Komponenten und die Funktionen sind die Systeme. Sie beide werden von Bevy verwaltet und aufgerufen. Dieser Ansatz hat mehrere Vorteile.

1. Die Daten sind global verfügbar. Jede Funktion, welche von Bevy aufgerufen wird, hat theoretisch die Möglichkeit auf die Daten zuzugreifen.
2. Es ist einfacher den Code lesbar und erweiterbar zu halten, da bei gebrauch einfach eine neue Funktion an Bevy übergeben werden kann.
3. Durch die Trennung der einzelnen Funktionen und Daten kann Bevy viele Funktionen gleichzeitig ausführen und somit effizienter die Ressourcen der Hardware ausnutzen.

Bevy stellt nur wenig zur Verfügung, aber durch ein einfaches Plugin System können Bibliotheken von anderen einfach in andere Projekte integriert werden.

Um ECS ein bisschen besser zu beleuchten, zeigt das folgende Beispiel, wie einfach die Daten- und Funktionsverwaltung in Bevy ist.

Diese Datenstruktur ist wie man an der ersten Codezeile erkennen kann eine Komponente. Mit dem `derive(Component)` wird Bevy mitgeteilt, dass die folgende Datenstruktur an einem gewissen Punkt von Bevy verwaltet wird. Die Daten können über die Commands an Bevy übergeben werden. In dieser Funktion/System macht Bevy Platz um eine Instanz des `Personenstructs` zu speichern. Das bedeutet allerdings, dass der Programmierer nicht weiß, wo sich diese Daten befinden. Die Daten werden komplett von Bevy verwaltet. Die Daten können allerdings wieder angefordert werden.

Diese Funktion/System nimmt ein `Query struct` als Parameter, welches Bevy signalisiert, dass alle Personen, die Bevy registriert hat von dieser Funktion gebraucht werden. Es

ist allerdings nicht klar, wieviele Personen registriert sind, weshalb das Query struct in einen Iterator verwandelt werden kann. Das heisst, dass man durch alle Einträge geht und diese einzeln verarbeitet.

Bisher wurden ausschliesslich Funktionen und Daten definiert, aber noch keine Hauptfunktion, welche die Daten und Funktionen auch braucht. Hierbei unterscheidet sich Bevy von anderen Frameworks, denn die Funktionen werden nicht vom Programmierer aufgerufen, sondern der Entwickler übergibt die Funktionen an Bevy und dieses ruft die Funktionen auf.

### 5.2.3 Performance

Orion ist ein Prototyp. Die Performance lässt zum aktuellen Zeitpunkt zu wünschen übrig. Eine angenehme Sichtweite ist nur mit starker Hardware zu erreichen.

Performanceprobleme lassen sich auf ein zentrales Problem reduzieren. Das Chunksystem. Die Welt ist in verschiedene Teile unterteilt, welche nach und nach geladen werden. Welche Chunks geladen sind hängt von der Spielerposition ab. Wenn sich der Spieler allerdings bewegt, heisst das, dass andere Teile der Welt geladen werden müssen.

In einer perfekt optimierten Version wäre jeder Chunk ein Mesh. Das heisst, dass der Chunk für die Hardware ein Teil ist und auch als solches behandelt werden kann. Die Hardware kann allerdings nur mit Dreiecken umgehen, weshalb der quadratische Chunk mit seinen Unebenheiten aus hunderten bis tausenden Dreiecken bestehen muss. Nebst dem Mesh braucht der Chunk ausserdem noch eine Oberfläche. Die Oberfläche besteht aus verschiedenen Farben, weshalb die Textur erstellt werden muss und anschliessend auf das Mesh gezogen werden muss. Wenn die Welt vorgefertigt wäre und nicht prozedural generiert, könnte man die Texturen generieren und anschliessend, wenn es gebraucht wird, von einer Datei geladen werden.

Die aktuelle Implementierung nutzt fast diese Technik, mit dem Unterschied, dass zwei Dreiecke zu einem Quadrat zusammengesetzt werden. Diese Dreiecke können nun im Raum platziert werden und mit einer einzigen Farbe ausgefüllt werden.

Die Performanceprobleme treten allerdings erst auf, wenn ein Chunk entladen und ein anderer geladen werden muss. Da ein Chunk aus mehreren hunderten bis tausenden Teilen besteht, muss jedes einzelne Objekt separat entladen werden. Der neue Chunk verschlechtert die Performance zusätzlich, da dieser aufgebaut werden muss und hunderte neue Objekte im Raum platzieren.



### 5.2.4 Zeitmanagement

Ursprünglich war definiert, dass das 3D Game ohne Game Engine entstehen soll. Es stellte sich allerdings schnell heraus, dass das in dieser Zeit nicht möglich war.

Die heutigen 3D Pipelines sind sehr komplex, was zur Folge hat, dass man ein breites Wissen und viel Erfahrung haben muss um ein solches Vorhaben umzusetzen.

Das finale Resultat nutzt eine 3D Engine, allerdings versucht es diese so wenig wie möglich zu nutzen und die wichtigsten Funktionen ohne diese zu implementieren.

## 5.3 Anwendungen

In den folgenden Abschnitten sollen die beiden restlichen Demos angesprochen werden. Während Actaeon als zentrales Produkt und Orion als zentrale Demo für dieses Produkt den Hauptteil dieser Arbeit ausmachen, so war es trotzdem eine Priorität, andere Einsatzmöglichkeiten beispielhaft zu zeigen.

### 5.3.1 Chat

Wie bereits in der ersten Entwicklungsphase wurde während der zweiten Entwicklungsphase basierend auf dem Actaeon-System ein Chat-Programm entwickelt. Dieses ist allerdings nicht die zentrale Demo für das gesamte Projekt, sondern lediglich ein technischer Test, mit dem die Fähigkeiten des Actaeon-Systems gezeigt werden sollen. Allerdings gibt es einige wichtige Unterschiede zur Chat Demo der ersten Entwicklungsphase, welche hauptsächlich durch eine andere Umsetzung des Datensystems entstanden:

- Der Chat ist nicht mehr die primäre Demo, weswegen das Design und die *Präsentierbarkeit* weniger wichtig wurden. Deshalb ist der aktuelle Chat nur noch als Konsolen-Applikation verfügbar und nicht mehr als Webseite.
- Das tatsächliche Datensystem, das sich um den Austausch der Nachrichten mit anderen Nutzern kümmert, läuft nun auf den Geräten der Endnutzer, weswegen ein dedizierter Webserver und eine Webseite nicht mehr nötig sind oder sogar unpraktisch wären.
- Der aktuelle Chat ist weniger ein Produkt, als eine technische Demo, mit welcher gezeigt werden soll, mit wie wenig Aufwand eine Applikation basierend auf dem Actaeon-System gebaut werden kann.

Die Chat Demo ist auf Github als Beispiel für die Actaeon Bibliothek verfügbar. Im Anhang dieses Dokuments ist der gesamte Code der Demo ebenfalls abgedruckt und wird dort auch kurz erklärt.

### 5.3.2 Arrow

Arrow ist eine weitere Demo, die aus der Anfangszeit des Projektes stammt. Arrow ist eine Lisp ähnliche Sprache, welche es erlaubt mit einfachen interpretierten Befehlen das

Actaeon Framework zu nutzen. Wie auch die anderen Projekte ist Arrow auch komplett in Rust geschrieben und hat mit der Zeit einige Veränderungen durchgemacht.

Arrow kann über eine interaktive Shell direkt genutzt werden, oder durch das angeben einer Datei auch Programme ausführen.

Wie alle anderen Teile von Project Orion kann man Arrow über Github beziehen. Um Arrow zu nutzen braucht man Rustup und Git. Danach findet alles im Terminal statt:

1. Implementierung Arrow ist wie alle anderen Teile der Arbeit in Rust implementiert. Arrow ist eine Lisp ähnliche Sprache, welche direkten Zugriff auf die Actaeon Bibliothek hat.

Arrow nimmt einen String entgegen und baut daraus einen AST (Abstract Syntax Tree). Anschliessend wird aus jedem einzelnen Knoten des AST ein LispObject generiert. Diese Lispobjekte werden anschliessend in der gleichen Struktur untereinander verbunden. Somit wird aus vielen komplexen ein Objekt, über welches das Programm ausgeführt werden kann.

Wenn das Programm ausgeführt wird, wird der erste Knoten aufgerufen. Wenn dieser abhängig ist von Daten, die dieser nicht lokal hat, ruft er seine Kinder auf und bekommt Daten von diesen. Wenn ein Knoten alle Daten hat gibt er Daten zurück, was sich anschliessend den Baum wieder nach oben arbeitet. Schlussendlich hat auch der letzte Knoten alle Daten und es wird ein Rückgabewert an den Aufrufer zurückgegeben.

2. Beispiel Das folgende ist ein Beispiel für ein Arrow Programm: Hier produziert Arrow drei Objekte. Das erste ist ein message Objekt und ist ein Kind von dem defun Objekt. Das letzte ruft das defun Objekt auf. Das defun Objekt ruft seine Kinder auf, was in diesem Fall das message Objekt ist. Das message Objekt hat keine Kinder, weshalb es seine Funktion ausführt. Das heisst, dass den ersten Parameter "Hello World" in der Konsole ausgibt. Da message nichts zurückzugeben hat, aber verpflichtet ist etwas zurückzugeben, gibt es nil zurück. Wenn man es mit anderen Programmiersprachen vergleichen will ist nil eine Kombination aus null und false.

# Fazit

---

Verteilte und dezentrale Datensysteme sind trotz ihrer Wichtigkeit nur ein kleines Nischengebiet in der Welt der Informatik. Und obwohl unsere Abhängigkeit von Netzwerken und Kommunikationssystemen grosse Probleme für alltägliche Situationen und Menschen mit sich bringt, lassen sich die Wenigsten auf ein tiefgreifendes Gespräch über die Probleme der Zentralrouter ein. Doch wenn diese Debatte nur in den Büros der Megaunternehmen geführt wird, könnten sich die Fortschritte des Internets schnell in eine finstere Dystopie verwandeln. Nebst den offensichtlichen Problemen und Gefahren übermässiger Zentralisierung und der damit verbundenen Abhängigkeit, darf nicht vergessen werden, dass kollaborative, dezentrale Systeme Unmengen von Vorteilen mit sich bringen. Seien es zensursichere Speicher- und Nachrichtensysteme oder die schiere Grösse von beispielsweise Videospielen, die ohne einen zentralen Knotenpunkt möglich werden.

Project Orion hat weder alle der genannten Probleme gelöst, noch das volle Potential dezentraler Datensysteme ausgenutzt und eine umfangreiche Alternative entwickelt. Stattdessen entstanden verschiedene *relativ* einfache, verständliche und nutzbare Demos, welche nicht nur reale Probleme lösen, sondern dazu einen Einblick in die Welt der dezentralen Datensysteme bieten. Mithilfe von umfangreicher Dokumentation und einfachen Erklärungen ist der Einstieg in die Welt der dezentralen Kommunikationssysteme einfacher gemacht worden. Da alle entstandenen Programme öffentlich zugänglich sind und unter freien Lizenzen weiter verbreitet werden können, ist es möglich, dass das Ökosystem in Zukunft durch weitere Applikationen erweitert wird. Mit insgesamt etwa 12000 Zeilen Code über 350 Commits sind mit diesem Projekt verschiedene, umfangreiche Programme entstanden, welche realen Nutzen bieten und vielseitig einsetzbar sind.



# Anhang

---

## 7.1 Recherche

In diesem Abschnitt sollen einige Projekte in ähnlichen Gebieten genauer angeschaut werden. Dabei soll es lediglich um technisch ähnliche Arbeiten gehen, die Welt der Video-spiele ist gross und vielseitig. Es wurde im Rahmen dieser Recherche kein Projekt gefunden, welches eine identische Zielsetzung oder die gleichen Produkte aufweist. Um aber die Wahl der Projekte richtig zu verstehen, muss man die Vision hinter Project Orion im Blick behalten. Denn es wird schnell klar, dass es für nahezu alle beschriebenen Probleme entweder temporäre Lösungen oder einzelne Projekte zur Umgehung der Probleme gibt. Daneben existieren auch grundlegendere Neuentwicklungen bekannter Systeme, welche einzelne Probleme lösen, meist aber andere Ziele haben.

### 7.1.1 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 der (*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, aber die meisten sind Nachfolger von Napster.

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 bei 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. [9] 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.

### 7.1.2 Tox

Im Sommer 2013 veröffentlichte Edward Snowden schockierende Geheimnisse über massive Spionage Programme der NSA, mit welchen nahezu aller digitaler Verkehr, ohne Rücksicht auf Datenschutz oder Privatsphären mitgelesen, ausgewertet und gespeichert wurde. Nahezu jede Person in der westlichen Welt war betroffen und das genaue Ausmass ist bis heute noch schwer greifbar. Vielen wurde aber klar, dass sichere, verschlüsselte Kommunikation nicht nur etwas für Kriminelle und *Nerds* ist, sondern dass jeder Zugang zu verschlüsselter, sicherer und dezentraler Kommunikation haben sollte. In einem Thread auf 4chan wurden viele dieser Bedenken gesammelt und es kam die Idee auf, selbst eine Alternative zu herkömmlichen Chat Programmen wie Skype zu entwickeln. Aus dieser Initiative heraus entstand Tox, wobei die Namen vieler der ursprünglichen Entwickler bis heute unbekannt sind. Damals war das Ziel die Entwicklung einer sicheren Alternative zu Skype, allerdings hat sich der Umfang des Projekts inzwischen ausgeweitet. Im Zentrum der Arbeiten steht das Tox Protocol, welches von verschiedenen, unabhängigen Programmen umgesetzt wird. Zwar ist Chat weiterhin eine zentrale Funktion, es wird aber auch mit Video- und Audiokommunikation sowie Filesharing gearbeitet.

Basierend auf der bekannten NaCl-Bibliothek wird die gesamte Kommunikation über das Tox Protocol [10] zwingend end- zu endverschlüsselt. Intern wird ein dezentrales Routing System, basierend auf Kademlia verwendet, mit welchem Kontakt zwischen Nutzern (Freunden) aufgebaut wird. Während im Kademlia Whitepaper Adressen 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 Kommunika-

tion bindet diese Idee die Verschlüsselung stärker in das Routing System ein, denn es werden keine zusätzlichen Informationen zum Verschlüsseln einer Nachricht gebraucht, und sie kann direkt mit der Adresse 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 gefunden haben, wechseln sie 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 ist 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.

Das Tox Protocol bietet eine einheitliche Spezifikation, mit der eine grosse Bandbreite an Problemen gelöst werden kann. 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 Ausrichtung des Projekts, sowie Berichte von internen Konflikten, besonders im Zusammenhang mit Spendengeldern.

### 7.1.3 CJDNS

Die grundlegenden Ideen und Lösungsansätze die im CJDNS-Whitepaper besprochen werden ähneln in vielerlei Hinsicht den Ideen und Prinzipien hinter Project Orion. CJDNS, das sich selbst als

an encrypted IPv6 network using public-key cryptography for address allocation and a distributed hash table for routing

beschreibt, ist ein beliebtes open-source Projekt mit mehr als 180 Mitentwicklern und tausenden von Nutzern. Es basiert ebenfalls auf Kademlia und ähnlich wie bei BitTorrent wird grosser Wert auf Sicherheit und Verschlüsselung gelegt.

Wer den Code von CJDNS etwas genauer anschaut realisiert schnell, welche grundlegenden Ziele CJDNS verfolgt. Tatsächlich soll mit CJDNS langfristig ein physikalisch unabhängiges Netzwerk entstehen. Das Whitepaper redet von der Freiheit der Nutzer, eigene Kabel und eigene Infrastruktur zu verlegen.

So etwas mag nach digitaler Isolation und Abschottung aussehen, aber wer tatsächlich konsequent alle Probleme von Grund auf angehen will, muss sich auch Gedanken über die darunterliegende physikalische Infrastruktur machen.

## 7.2 Ausblick

Auch wenn sind die geschaffenen Produkte und Programme ihren Zweck erfüllen und eine nutzbare Demo möglich machen, so gibt es einige Punkte, die noch nicht umgesetzt wurden.

- Actaeon:  
Wie in nahezu jedem Software Projekt ist auch Actaeon nicht ohne Bugs oder Probleme. Nebst diesen stehen allerdings auch noch einige andere Schritte an, welche die Nutzung von Actaeon verbessern könnten:
  - Anstelle des aktuellen Threading-Models muss der Wechsel auf die neuen Rust `async-await` Funktionen gemacht werden. Zwar kommen damit mehr Abhängigkeiten und mehr Komplexität, allerdings sollte sich dieser Schritt lohnen, da eine solche Umsetzung signifikant besser laufen sollte.
  - Als weiteres grosses Feature soll *Record-Republishing* eingebaut werden. Damit sollte das Netzwerk auch bei abrupten, grossflächigen Änderungen der Netzwerkstruktur funktionieren, was aktuell noch zu Problemen führen kann.
- Orion:  
Natürlich sind bei einem Videospiel die Möglichkeiten nahezu unlimitiert, allerdings existieren auch hier einige interne, technische Probleme. Nebst der offensichtlichen Frage der Performance, gibt es auch ein grundlegenderes Problem: Auch wenn ein *offizieller* Client existiert, hält einen Entwickler nichts davon ab, eine alternative Umsetzung zu erstellen. Daran ist an sich nichts falsch, allerdings kann eine solche Offenheit schnell ausgenutzt werden, um beispielsweise unfaire Vorteile im Spiel zu erhalten. Wer also das Spiel als einfachen Zeitvertreib ausprobieren will oder etwas Spass sucht, wird zufriedengestellt sein. Als seriöses Spiel, vergleichbar mit kommerziellen Alternativen, ist es allerdings noch nicht geeignet.
- Weiteres:  
Der Umfang des Möglichkeiten beschränkt sich nicht nur auf Videospiele oder Chat Programme. Eine Vielzahl anderer Anwendungen lässt sich mit der actaeon Bibliothek bauen. Dank einer offenen Lizenz gibt es ebenfalls die Hoffnung, dass andere Personen in der Zukunft Programme für das Ökosystem bauen.



## 7.3 Code

### 7.3.1 Chat Demo

Der gesamte Code der Chat Demo sowie eine kurze Erklärung:

```
use actaeon::{
    config::Config,
    node::{Center, ToAddress},
    topic::Topic,
    Interface,
};

use sodiumoxide::crypto::box_;
use std::io;
use std::sync::mpsc;

fn main() -> io::Result<()> {
    let config = Config::new(20, 1, 100, "example.com".to_string(), 4242);
    let (_, secret) = box_::gen_keypair();
    let center = Center::new(secret, String::from("127.0.0.1"), 4242);
    let interface = Interface::new(config, center).unwrap();
    let (s, r) = mpsc::channel();
    std::thread::sleep(std::time::Duration::from_millis(125));
    let mut buffer = String::new();
    let stdin = io::stdin();
    stdin.read_line(&mut buffer)?;
    let topic = buffer.to_address();
    let topic = interface.subscribe(&topic);
    receiver(topic, r);

    loop {
        let mut buffer = String::new();
        let stdin = io::stdin();
        stdin.read_line(&mut buffer)?;
        let message = buffer.as_bytes().to_vec();
        let _ = s.send(message);
    }
}

fn receiver(mut topic: Topic, recv: mpsc::Receiver<Vec<u8>>) {
    std::thread::spawn(move || loop {
        if let Some(msg) = topic.try_recv() {
            let body = msg.message.body.as_bytes();
            let message = String::from_utf8_lossy(&body);
            let from = &msg.source().as_bytes()[0];
            println!("{}", from, message);
        }
    })
}
```

```

    }
    if let Ok(bytes) = recv.try_recv() {
        let _ = topic.broadcast(bytes);
    }
});
}

```

Eine kurze Erklärung was mit diesem Code erreicht wird:

1. Als erstes wird das Actaeon-System konfiguriert und gestartet.
2. Das Thema (in diesem Falle der Chatraum) wird als Input vom Nutzer genommen.
3. Die daraus errechnete Adresse wird Actaeon übergeben, welches dann mit anderen Mitgliedern des Systems kommuniziert und die nötigen Verbindungen herstellt.
4. Da Rust sehr strikte Regeln im Umgang mit Threads und nebenläufigen Programmen hat, müssen die Funktionen auf zwei Threads aufgeteilt werden:
  - Einer kümmert sich um Eingaben vom Nutzer,
  - der andere um Nachrichten vom System.

Sollte eine Applikation eine andere Methode verwenden, Eingaben vom Nutzer zu erhalten, ist es natürlich nicht nötig, einen eigenen Thread zu starten und Actaeon würde sich automatisch im Hintergrund darum kümmern.

5. Beide Threads laufen nebenläufig in einer unendlichen Schleife und warten auf Ereignisse.

Der Code, der tatsächlich für das Actaeon Datensystem relevant ist beschränkt sich allerdings auf die folgenden Zeilen:

```

{
    let config = Config::new(20, 1, 100, "example.com".to_string(), 4242);
    let (_, secret) = box_::gen_keypair();
    let center = Center::new(secret, String::from("127.0.0.1"), 4242);
    let interface = Interface::new(config, center).unwrap();
    let topic = buffer.to_address();
    let topic = interface.subscribe(&topic);
    let _ = topic.broadcast(vec![]);
}

```

Man muss nicht jede Zeile dieses Beispiels exakt verstehen. Was hier schlussendlich wichtig ist, ist die Feststellung, dass nur wenige Zeilen Code nötig sind, um eine beliebige Applikation an ein dezentrales Datensystem anzuschliessen.

## 7.4 Glossar

In diesem Abschnitt sollen alle Begriffe, welche im Text mindestens einmal `monospaced` geschrieben wurden, kurz erklärt werden.

- **Project Orion:** Maturaarbeit zum Thema *Dezentrale Kommunikations- und Daten-systeme* von Dominik Keller und Jakob Klemm, Kanti-Baden, 2021.
- **ARPANET:** Vorgängermodell des Internets, entwickelt vom amerikanischen Verteidigungsministerium.
- **FOMO:** Fear of missing out: Die Angst, Innovationen und Veränderungen zu verpassen.
- **File sharing:** Austausch von Dateien direkt zwischen Personen oder öffentliches Verbreiten. Kann sowohl über dezentrale Programme wie Torrents, im direkten Austausch per E-Mail oder über einen dritten Dienst wie OneDrive oder Google Drive geschehen.
- **Content Addressing:** Domänen und IP-Adressen beschreiben Wo etwas zu finden ist, nicht um was es sich tatsächlich handelt. Würde man direkt nach dem passenden Inhalt fragen, beispielsweise über einen Hash-Wert, könnte man nicht nur die Abhängigkeit von zentralisierten Diensten umgehen, sondern könnte unter Umständen sogar bessere Geschwindigkeiten und weniger Datenverkehr erhalten.
- **IP-Adresse:** Eindeutige Identifizierung für Geräte im Internet. Werden von Internetanbietern ausgestellt und sind meistens nur eindeutig für einzelne Netzwerke, nicht Geräte, da diese ein unabhängiges IP-Adress-System verwenden.
- **TCP:** Transmission Control Protocol: Dominantes Protokoll um Daten im Internet zu versenden. Grundlage für bekanntere Protokolle wie HTTP.
- **UDP:** User Datagram Protocol: Simpleres und schnelleres Protokoll als TCP, ist allerdings meist weniger zuverlässig.
- **IP-V4:** Standardisierte Version der IP-Adressen mit einer Länge von 32 bit.
- **IP-V6:** Neuste Version des Internet Protokolls mit einer Adresslänge von 128 bit, allerdings inkompatibel mit IP-V4.
- **POP-Switch:** Point of Presence: Von einem Internetanbieter kontrollierter Knotenpunkt, der ein kleineres *Subnet* abgrenzt.
- **Zentralrouter:** Grössten und wichtigsten Knotenpunkte für den globalen Internetverkehr, der grösste befindet sich in Frankfurt.
- **Address Spaces:** Um Speicherplatz und Rechenleistung zu sparen, werden komplette IP-Adressen an Firmen oder Gebiete vergeben.
- **Shadow:** Orion Netzwerk-Router der ersten Entwicklungsphase, geschrieben in Elixir.
- **Elixir:** Programmiersprache für nebenläufige Programme, besonders Netzwerke, basierend auf Erlang.
- **Kademlia:** Peer to Peer Nachrichtensystem basierend auf der XOR-Metrik.
- **UNIX-Socket:** System zum Datenaustausch zwischen verschiedenen laufenden Applikationen auf der selben Maschine.

- **Hunter**: Lokaler Router der ersten Orion Entwicklungsphase.
- **JSON**: Menschenlesbares Datei- und Nachrichtenstruktur für Objekte und Schlüssel-Wert-Paare.
- **Websocket.or**: Chat Interface für das erste Orion Datensystem.
- **Actaeon**: Dezentrales Nachrichtensystem der zweiten Orion Entwicklungsphase.
- **Rust**: Moderne Programmiersprache von Mozilla als Alternative zu C & C++.
- **Crate**: Rust Terminologie für, von Nutzern entwickelte, Bibliotheken.
- **XOR**: Logische Operation auf Binärdaten errechnet den Unterschied zwischen zwei Eingaben.
- **NaCl**: Verschlüsselungsbibliothek für Netzwerkprogramme.
- **Interface**: Zentrale Zugangsstelle und Interaktionspunkt für die Actaeon Applikation. Siehe Code Dokumentation für mehr Details.
- **Topic**: Actaeon Datenstruktur die ein, im Netzwerk registriertes Thema zu einem gewissen Schlüssel repräsentiert.
- **CJDNS**: Initiative für ein technisch und physikalisch unabhängiges Netzwerk als Alternative zum aktuellen Internet.
- **Routing Table**: Gespeicherte Daten über bekannte Nodes in einem Netzwerk zu denen Verbindungen aufgebaut werden können.
- **k-Bucket**: Kademlia unterteilt den gesamten Routing Table in kleinere Stücke, wobei alle Nodes in einem solchen Bucket in einen gewissen Anteil ihrer Adresse übereinstimmen.
- **Napster**: Eine der ersten Methoden illegal Musik direkt zwischen Nutzern zu teilen.
- **BitTorrent**: Aktuell beliebteste Methode um Medien und Daten aller Art ohne jegliche Zentralisierung auszutauschen.
- **SHA1**: Hash Algorithmus der einen Schlüssel mit einer Länge von 160 bit produziert. Ursprünglich von der US-Regierung entwickelt.
- **libsodium**: Moderne Umsetzung der NaCl Standards und Ideen.
- **Publish/Subscribe & PubSub**: Prinzip der indirekten Nachrichtenübermittlung via Themen mit gemeinsamen Schlüsseln.
- **Topic Record**: Topic Objekt an der rechnerisch korrekten Stelle in einem Actaeon Cluster. Kümmt sich um das weiterverbreiten neuer Abonnenten.

---

# Abbildungsverzeichnis

---

- 3.1 Beispielhafte Darstellung eines einfachen Kademlia-Systems, Wikipedia: <https://en.wikipedia.org/wiki/Kademlia>, 14
- 4.1 Engine: Orion Chat-Website, intern verbunden mit einem dezentralen Kommunikationssystem. 20



---

## Literaturverzeichnis

---

- [1] Jon Postel. Internet protocol. STD 5, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc791.txt>.
- [2] Lawrence E. Hughes. Ipv4 and ipv6 address spaces. <https://thirdinternet.com/ipv4-and-ipv6-address-spaces/>, 2019.
- [3] Google. Google privacy terms. <https://policies.google.com/privacy>.
- [4] Tom Scott. Single point of failure: The (fictional) day google forgot to check passwords. [https://www.youtube.com/watch?v=y4GB\\_NDU43Q](https://www.youtube.com/watch?v=y4GB_NDU43Q), 2014.
- [5] Michael Grothaus. That major google outage meant some nest users couldn't unlock doors or use the ac. <https://www.fastcompany.com/90358396/that-major-google-outage-meant-some-nest-users-couldnt-unlock-doors-or-use>
- [6] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. Technical report, 2002.
- [7] Dominik Keller and Jakob Klemm. Engine: Orion. <https://github.com/EngineOrion/kommentar/blob/54489464214ed7833f182df09aab29eae4a591e4/EngineOrion.pdf>, 2021.
- [8] Peter Sloterdijk. *Den Himmel zum Sprechen bringen*. 2020.
- [9] Liang Wang and Jussi Kangasharju. Measuring large-scale distributed systems: Case of bittorrent mainline dht. Technical report, 2013.
- [10] Tox specs. <https://toktok.ltd/spec.html>.

## Bestätigung

Ich/Wir erkläre/-n hiermit, dass meine/unsere Maturaarbeit von mir/uns verfasst oder entwickelt und nicht als Ganzes oder in Teilen kopiert wurde.

Aus Quellen übernommene Teile sind – nach den entsprechenden Regeln – als Zitate erkennbar gemacht. Alle Informationsquellen sind in einem Literaturverzeichnis aufgeführt.

Vorname/-n, Name/-n:

Dominik Keller, Jakob Klemm

Abteilung/-en:

G4a

Maturaarbeit:

Project Orion

Ort, Datum:

Baden, Schweiz, 7. 11. 2021

Unterschrift/-en:

D. Keller

J. Klemm

Eine Kopie der Bestätigung geben Sie – mit Originalunterschrift versehen – bei der Schlusspräsentation im November ab.



# Vertrag Maturaarbeit 2021

Dieser Vertrag definiert das in Angriff genommene Projekt und die sich daraus ergebende Maturaarbeit. Der Vertrag legt die Zeit- und Kostenplanung der Projektarbeit sowie die Bewertungskriterien für den Arbeitsprozess, die Präsentation und das Produkt fest.

## Studierende

	Name, Vorname	Abteilung
1	Keller, Dominik	G3a
2	Klemm, Jakob	G3a

## A Projektbeschreibung

1. Titel der Arbeit: *provisorisch*

Project Orion

2. Projektthema: *Inhalt, Problem- oder Fragestellung*

Dezentralisierte Datenstrukturen & Praktische Anwendungen.

3. Projektform: *Vorgehensweisen, verwendete Methoden und Techniken*

Entwicklung eines dezentralisierten Nachrichtensystems sowie verschiedene Anwendungen.

4. Produkt:

Minimalziel zum Zeitpunkt der Zwischenpräsentation:

Bis zur Zwischenpräsentation sollen die drei zentralen Komponenten grundlegend funktionsfähig sein:

- Rudimentäre Kommunikation verschiedener Server in einem dynamisch entstehenden verteilten Datensystem.
- Dateneingabe & Kommunikationsportal der verschiedenen Komponenten.
- Einfache Befehls Eingabe und Exekution in NET-Script.

Ziele bis zur Abschlusspräsentation:

- Dezentrales Nachrichten-System und automatische Verwaltung der Verbindungen basierend auf Kademia.
- Chat-Anwendung: Terminal-basierendes Chat-System.
- NET-Script-Anwendung: Integration von NET-Script (Teilweise entwickelt im ersten Abschnitt) und Netzwerk, Ausführen von Code mit Nachrichten.
- Emacs-Anwendung: Kollaboration für Emacs durch das Netzwerk.
- Game-Anwendung: Einfacher Multiplayer Flugsimulator ohne Game-Engine auf einer prozeduraler Welt mit grundlegenden Funktionen.

- Corona-Anwendung: Auf ähnlichen Prinzipien wie der Chat lässt sich auch ein rudimentäres Contact-Tracing-System entwickeln, wieder ohne grossen Fokus auf Design.

Schriftlicher Bericht:

- Erklärung der technischen Funktionsweise.
- Besprechung der verschiedenen Anwendungen & Beispiele.
- Analyse / schriftliche Erklärung der Implikationen & Potential / Relevanz.
- Vergleich mit anderen Produkten & Systemen.
- Zukunftsaussicht & Verbesserungsmöglichkeiten
- („Getting started Guide“)
- (Rudimentäre Tests und Messungen über Effizienz und Skalierbarkeit.)

5. Relevanz: *Zielgruppe und Wirkung*

Unsere Abhängigkeit von digitalen Plattformen ist höher als je zuvor. Mit immer besseren Funktionen und Fähigkeiten steigt auch die Überwachung und Regulation. Während die Zahl der Seiten und Apps stetig steigt, sind inzwischen nur noch eine kleine Anzahl Firmen im Besitz des gesamten Markts. Auch wenn dezentrale Applikationen existieren, benötigt man meist viel technisches Knowhow oder man verliert wichtige Funktionen. Für gezielte Anwendungen sollte es aber möglich sein, dezentrale Funktionen ohne extra Aufwand einzubauen.

6. Notwendige/verfügbare Ressourcen:

-

## B Zeit- und Kostenplan

Datum	Abgeschlossene Teilschritte
17. März	
24. März	
31. März	
7. April	
10. April bis 24. April	Frühlingsferien = Erster Prototyp der einzelnen Komponenten
28. April	Begin: Ausarbeitung & Integration
5. Mai	
12. Mai	
19. Mai	
26. Mai	End: Ausarbeitung & Integration
2. Juni	Dokumentation, Darstellung, Verifizierung, Vorbereitung der Präsentation
9. Juni	Abgabe der Arbeiten
16. Juni	Zwischenpräsentationen
23. Juni	
23. Nov.	Abgabe der Maturaarbeiten
30.11. 1.12./ 2.12.	Schlusspräsentationen der Maturaarbeiten

### Kostenabschätzung

Ausgabe	CHF

## C Bewertungskriterien

### 1. Arbeitsprozess: *Gewichtung 20%*

- a) methodisches Vorgehen
  - Strukturierung des Projektes (Teilfragen und Teilschritte)
  - Planung der Teilschritte (Lösungsoptionen, Tests, Auswahl)
  - Kontinuierliche Standortbestimmung (Analyse der erzielten Teilresultate)
  - Reflexion der Vorgehensweise (Zielführung)
- b) inhaltliche und formale Fortschritte
  - Erarbeitung von Fachwissen (Inhaltsrecherche und Auswertung)
  - Erarbeitung fachlicher Fertigkeiten (Technikrecherche und Anwendung)
  - Erarbeitung fachlicher Urteilsfähigkeit (Qualitätskriterien)
- c) Arbeitsorganisation
  - Realistische Zeitplanung (Gesamtprojekt und Teilschritte)
  - Arbeitsaufteilung in Gruppenarbeiten (Ausgewogenheit und Verbindlichkeit)
  - Kommunikation inhaltlicher, formaler und organisatorischer Probleme
  - Integration von Expertenwissen (externe und schulinterne Fachpersonen)
  - Logbuch (Struktur, Übersichtlichkeit, Reflexionsgehalt)

Bewertungsgrundlagen sind das Logbuch und die Betreuungsgespräche.

### 2. Präsentation: *Gewichtung 30%*

- a) inhaltliche Qualität
  - Vorführung des Produktes (Beschaffenheit, Bestandteile, Funktion, Wirkungsweise)
  - Darlegung der wichtigsten Produktionsschritte (Proben, Entscheidungen Ausführungen)
  - Darlegung fachlicher Komponenten (Fachwissen, Verarbeitungstechniken)
- b) formale Qualität
  - Struktur und Ablauf (Übersichtlichkeit und Schlüssigkeit)
  - Publikums- und fachgerechte Sprache und Visualisierung (Allgemeinverständlichkeit)
  - Einnehmender Auftritt, sprachliche Korrektheit, geeigneter Medieneinsatz

Bewertungsgrundlagen sind die Produktpräsentationen anlässlich der Zwischen- bzw. Schlussbewertung. Das Infoposter und das Abstract können anlässlich der Schlussbewertung einbezogen werden.

### 3. Produkt: *Gewichtung 50%*

Zentrales Bewertungskriterium ist das Erreichen der in der Projektbeschreibung festgelegten Produktziele. Spezifische Produktqualitäten sind nachfolgend zu definieren.

Zeitpunkt Zwischenpräsentation:

- Verteiltes Datensystem:

Kommunikation zwischen mindestens zwei Servern.

Dynamische Cluster Formen, Dynamischen Hinzufügen einzelner Server.

Verteiltes, indirektes Routing, Daten werden von Server zu Server ohne zentralen Router übertragen.

- Portal:

Manuelle Dateneingabe ins System.

CLI Client für Kommunikation zum Testen und Vorführen.

Zuweisung der einzelnen Nachrichten an passende Container.

- Container & NET-Script:

Initialisierung & Konfiguration einzelner Container.

Empfangen und verarbeiten rudimentärer Nachrichten.

Ausführen einfacher NET-Script Befehle.

Zeitpunkt Abschlusspräsentation:

Netzwerk:

- Zwei Geräte können Nachrichten austauschen, ohne dabei direkt von Hand verbunden worden zu sein (Datenaustausch über ein drittes Gerät oder über eine automatisch aufgebaute Verbindung).

- Netzwerk soll in andere Projekte eingebunden werden können oder alleine~verwendbar sein.

- Der Code ist intern (Kommentare) und extern (Wiki / Guides) dokumentiert und ist für Aussenstehende verwendbar.

- Neben der technischen Umsetzung erfüllt das Netzwerk auch die ideologischen Ziele, es ist also möglich ein dezentrales, unabhängiges und spezialisiertes Cluster zu starten.

Anwendungen:

- Mindestens 3 der 5 geplanten Anwendungen sind benutzbar und der Aufwand zur Verwendung ist vergleichbar mit ähnlichen, nicht dezentralen Programmen.

- Das Entwickeln von Anwendungen ist gut dokumentiert und lässt sich ohne genaues Verständnis der Funktionsweise aller Komponenten möglich.



Plagiate, Teilplagiate und das Verschweigen von Quellen werden als Betrugsversuch gewertet und haben eine Note 1 und die Zurückweisung der Arbeit zur Folge.

Zum Zeitpunkt der Zwischenpräsentation und der Schlusspräsentation wird eine Bestätigung verlangt, dass die Arbeit selbst entwickelt und verfasst wurde.

Die Beurteilung nach der Zwischenpräsentation wird von den Betreuungspersonen schriftlich verfasst und ist zugleich für die weitere Arbeit an der Maturaarbeit wegweisend. Die Beurteilung nach der Schlusspräsentation erfolgt mündlich.

Sollten die Betreuungspersonen den Eindruck gewinnen, dass die Zusammenarbeit der einzelnen Mitglieder der Projektgruppe wesentliche Mängel aufweist, sind sie berechtigt Einzelnoten anstatt einer Gruppennote zu setzen.

**D Unterschriften****Studierende**

	Name, Vorname	Datum, Unterschrift
1	Keller, Dominik	 9.4.2021
2	Klemm, Jakob	 9.4.2021

**Betreuungsperson**

Name	Datum, Unterschrift
Hallström, Simon	