

Project Orion

DOMINIK KELLER, JAKOB KLEMM
KANTI BADEN, G4A

SIMON HALLSTRÖM, MICHAEL SCHNEIDER

23. NOVEMBER 2021

Inhaltsverzeichnis

Inhaltsverzeichnis	2
1 Einleitung	5
2 Zielsetzung	7
3 Theorie	9
3.1 Adressen	9
3.2 Zentralisierung	11
3.3 Kademlia	12
4 Prozess	16
4.1 Modularität	16
4.2 Präsentation	20
5 Produkte	22
5.1 Actaeon	22
5.2 Orion	29
5.3 Anwendungen	32
6 Fazit	34
7 Anhang	36
7.1 Ausblick	36
7.2 Recherche	37
7.3 Code	39
7.4 Glossar	41
Abbildungsverzeichnis	44
Literaturverzeichnis	45

Abstract

Wir leben in einer bequemen Welt. Das wollen wir wenigstens glauben. In Wahrheit leben wir aber in einer idyllischen Illusion. Die unglaublichen Veränderungen unserer Gesellschaft und unseres Alltags hätte sich vor 30 Jahren niemand vorstellen können. Die Geschwindigkeit der Fortschritte hat noch nie gesehene Mengen an Wohlstand und Reichtum geschaffen. Doch im Rausch des Wandels wurden gewisse Entscheidungen, manche bewusst und böswillig, andere aus Not, getroffen, welche uns in unserer modernen, vom Internet vollständig abhängigen Gesellschaft in eine prekäre Situation bringen. Die tiefsten Fundamente der Welt, wie wir sie kennen, sind dem totalen Zusammenbruch gefährlich nahe, und was das System am Laufen hält, sind oftmals monopolistische Megaunternehmen und ihre ungewählten CEO's, die nur von immer mehr Macht, mehr Einfluss und mehr Nutzern träumen.

Es ist offensichtlich, dass eine Alternative her muss. Aber während verschiedenste Projekte bereits grosse Fortschritte in den Bereichen der dezentralen Kommunikation und Datenspeicherung machen, bleiben oftmals die Nutzer aus. Denn trotz aller Innovation und aller Vorteile geht es hier um äusserst abstrakte, komplexe Themen, welche Unmengen an Vorwissen und Interesse benötigen, um sie genügend zu verstehen. *Project Orion* versucht genau da anzusetzen: Ein dezentrales Kommunikationssystem soll verschiedene, praktisch einsetzbare Anwendungen dezentraler Systeme in die Hände der Endnutzer bringen und dort einen sichtbaren, verständlichen Unterschied machen. Besonders dafür wurde ein 3D Multiplayer Videospiel entwickelt, welches basierend auf einem eigenen dezentralen Kommunikationsprotokoll, die gewohnte Erfahrung bekannter Videospiele liefern soll, ohne dabei in die Fänge der Megaunternehmen zu geraten.

Zusätzlich werden die entwickelten Werkzeuge mit ihrer Dokumentation und unseren Erfahrungen und Lektionen öffentlich zur Verfügung gestellt, sodass zukünftige Projekte davon profitieren können und die Einstiegshürde für das Gebiet der dezentralen Daten-systeme hoffentlich gesenkt wird.

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 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 Abschnitte, 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. Im Anhang wird zusätzlich ein Ausblick in die Zukunft des Projekts gegeben.
- **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 äusserst 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 eher 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 und die Mitglieder kommunizieren direkt miteinander.

Allerdings werden viele dieser dezentralen Technologien für illegale Zwecke wie 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 fortschrittlicheren 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 beispielsweise 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 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, werden stattdessen verschiedene, beispielhafte Anwendungen entwickelt. Diese werden 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 die Entwicklung von Prototypen, die die technischen Möglichkeiten aufzeigen.

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

Theorie

Da es sich bei dezentralen Datensystemen um ein sehr spezifisches, komplexes Thema handelt, werden als erstes einige der theoretischen Grundlagen erklärt. Dabei geht es nicht nur um dezentrale Systeme an sich, es werden auch einige der bereits erwähnten Probleme detaillierter angesprochen. 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 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 Baumstruktur* 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 Internetanbieter 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 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. Internetanbieter und grosse Firmen können diese Abschnitte untereinander verkaufen und aufteilen. Und jede Firma will 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 liegt der Schwerpunkt hier auf zwei Aspekten, 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 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 Echtzeitpositionsupdates, 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: Internetanbieter, 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, die gewonnene 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 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, trafe 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 der 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 Kademliasysteme 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

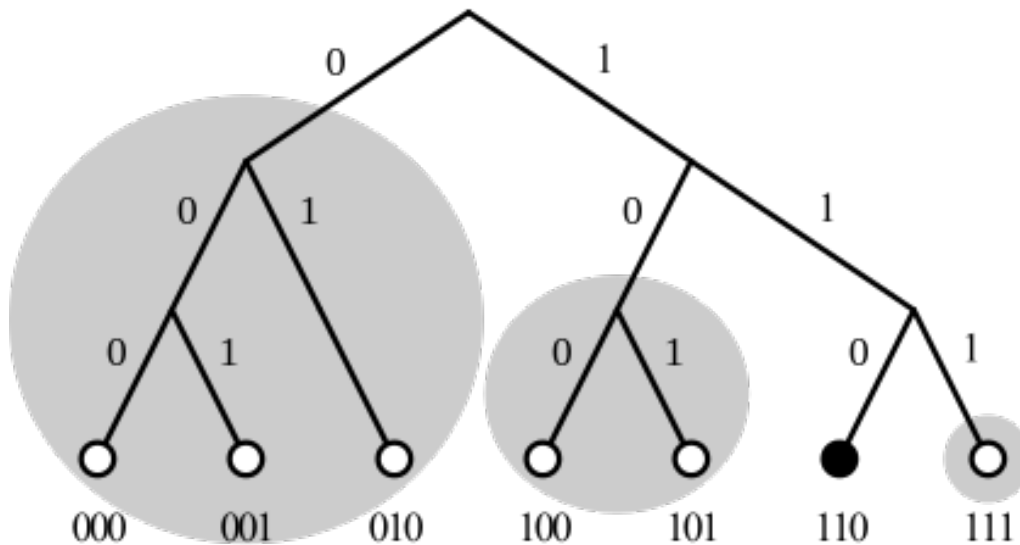
Die XOR-Funktion, die in der Informatik an den verschiedensten Orten auftaucht, wird verwendet, um die Distanz zwischen zwei Zahlen zu berechnen. Die Zahlen repräsentieren dabei Mitglieder im Netzwerk und sind je nach Variante im Bereich $0..2^{160}$ (20 Bytes) oder $0..2^{256}$ (32 Bytes). Mit einem 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 einiges mehr Adressen als beispielsweise IP-V4.

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. $xor(x, x) = 0$: Die Adresse eines Mitglieds ist zu sich selbst am nächsten. Mitglieder werden hier als Namen für Objekte Server in einem Kademliasystem 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 Kademliasytems, 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 Kademliasytem hat jedes Mitglied eine gewisse Anzahl anderer Mitglieder, mit denen verbunden ist. Da Kademlia ein sehr umfangreiches, kompliziertes Protokoll und System beschreibt, werden hier nur einige zentrale Funktionen besprochen.

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:

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, in wie viele 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.
- 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 Kademliasystem 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 Kademliasystem 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 wird hier genau beschrieben, für die zweite Phase sollen lediglich die Unterschiede angesprochen werden. Die tatsächlichen Applikationen und Resultate 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, gewisse Konzepte und einige Programme aber lassen sich für die neuen 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 nebst den allgemeinen Ideen dezentraler Systeme 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 wichtigste Komponente dieses Systems gibt es ein dezentrales Datensystem, welches die Kommunikation zwischen verschiedenen Geräten steuert.
- **Offenheit**
Sobald man den Nutzern die Möglichkeit geben will, das System selbst zu erweitern und zu bearbeiten, muss man quasi zwingend open source Quellcode zur Verfügung stellen.

Die grundlegende Idee war die Entwicklung eines dezentralen, vielseitig einsetzbaren Kommunikationsprotokolls. Da allerdings keine konkrete Anwendung angestrebt wurde, ging es stattdessen um die Entwicklung eines vollständigen Ökosystems und allgemein einsetzbarer Komponenten.

In den folgenden Abschnitten werden einige dieser Komponenten und die Entscheidungen, die zu ihnen geführt haben, beschrieben. In einem weiteren Abschnitt werden dann die Lektionen und Probleme dieser erster Entwicklungsphase besprochen.

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 *Shadow* 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, kann *Shadow* sich mit anderen Instanzen verbinden und über eine rudimentäre Implementierung des Kademliasystems Nachrichten senden und weiterleiten. Um neue Verbindungen herzustellen, wurde ein speziell entwickeltes System mit so genannten *Member Files* verwendet. Jedes Mitglied eines Netzwerks kann eine solche Datei generieren, mit welcher es beliebigen andere Instanzen beitreten konnte.

Sobald eine Nachricht im System am Ziel angekommen ist, wird sie über einen `Unix-Socket` an die nächste Komponente im System weitergegeben. Dies geschieht 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, muss dann also nur das verhältnismässig kleine Protokoll implementieren und ist 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 zuerst Hunter 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 haben müssen.

4.1.3 NET-Script

Eine weitere wichtige 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 und einfacher macht. Eine einfache lisp ähnliche Syntax sollte das Entwickeln neuer Programme einfach und vielseitig einsetzbar machen. Dieses Produkt wurde bereits während der ersten Entwicklungsphase nahezu fertig gestellt und konnte komplett für die zweite Phase übernommen werden.

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 einige 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 liefen und das Datensystem eine Schnittstelle zwischen verschiedenen Servern ermöglichte. Der Chat musste 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 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 dem dezentralen Kommunikationssystem.

Nutzer sind mit dieser Applikation in der Lage, mit beliebigen anderen Nutzern zu kommunizieren, solange sie die Adresse von einem Chat Server kennen, der das passende Interface anbietet.

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 zentralen Ideen war die Modularität, welche es Nutzern erlaubt, 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 selbst für die einfachsten Demos mindestens vier der Komponenten benötigt werden, kann nahezu alles schiefgehen. Dazu kommt, dass viele Fehler nicht richtig isoliert und verarbeitet werden, weswegen sich die Probleme durch das System weiter verbreiten werden. 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 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. Aber gleichzeitig kommen 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.
- Zwar werden die Werkzeuge meistens immer besser und schneller, allerdings kommt es oftmals auch zu einer Spezialisierung. Dies führt schnell zu immer spezifischeren, exotischeren Lösungen, unzähligen Unterbereichen und immer kleineren Fachgebieten. So ist beispielsweise der Begriff *dezentrale Datensysteme*, der zwar ein einzelnes Gebiet genau beschreibt, für Aussenstehende eher bedeutungslos und sorgt für mehr Verwirrung als Aufklärung.
- Die immer neuen Fachgebiete können auch schnell zu einer Art 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 technischen Grundlagen würde Stunden benötigen.

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 in der zweiten Entwicklungsphase 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 vielseitigen 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, 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 am besten den dazugehörigen Bericht durch, oder schaut sich direkt die dabei entstandenen Programme an. [7]

Dieses Kapitel ist in drei Abschnitte unterteilt:

- Actaeon beschreibt das dezentrale Datensystem und die Rust Bibliothek mit der es sich verwenden lässt. Dazu werden einige der technischen Details angesprochen.
- Orion beschreibt das Videospiel, welches als wichtigste Demo dem gesamten Projekts den Namen gegeben hat.
- Anwendungen beschreibt die Integration in die eigenen Programmiersprache sowie die Chat Anwendung.

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 Jagd 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]

Als zentrales Verbindungsstück zwischen allen Applikationen hängen alle Programme von Actaeon ab. Um diese Integration einfach zu machen, sind nahezu alle Applikationen in Project Orion in der Programmiersprache Rust geschrieben. Der Code ist online auf Github verfügbar. 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 kann, ist es nicht von Nöten, die interne Funktionsweise vollständig zu verstehen. An dieser Stelle soll ein Überblick über die Verwendung und die Funktionen gegeben werden. In den folgenden Abschnitten werden dann zusätzlich noch einige technische Details zur Bibliothek angesprochen, diese sind aber nicht notwendig, um die Bibliothek zu verwenden. Für die technischen Feinheiten lohnt es sich, die online Dokumentation und Package Informationen zu lesen, welche auf crates.io verfügbar sind.

Es lohnt sich aufzulisten, welche Funktionen die Bibliothek übernimmt, und welche darüber hinaus vom Nutzer verlangt werden. Actaeon übernimmt das Folgende:

- Verbindungen (stateful & stateless): Das System entscheidet intern selbständig, wie Nachrichten versendet werden. 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 geringe Wahrscheinlichkeit einer Kollision über die 32 Bytes reicht allerdings aus.
 - Routing: Mit den Adressen lässt sich ebenfalls rechnen, wobei es hauptsächlich um die Kademlia XOR-Operation geht. Damit werden Distanzen, Wege und 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 generieren. Damit lassen sich im System Nachrichten automatisch End zu End verschlüsseln, ohne dass ein dedizierter Mechanismus zum Austausch von öffentlichen Schlüsseln benötigt wird.
- 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.

- Updates: Durch regelmässige Abfragen und Überprüfungen können Mitglieder automatisch neue Mitglieder finden und sie in ihren lokalen Routing-Table einbauen.

Der Nutzer muss sich aber um einige Aspekte selber kümmern:

- Die Verbindungsdaten für ein beliebiges Mitglied eines Clusters müssen bekannt sein, damit das System eine erste Verbindung aufbauen kann. Auch wenn gewisse Automatisierung möglich ist, wird dieser Schritt niemals vollständig aus den Händen der Nutzer genommen werden können, ohne 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 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 zu verwalten. Das zweite wichtige Element ist dabei das `Topic`, denn die meisten Interaktionen des Nutzers finden über ein solches Thema statt. Unter einem Thema werden Nachrichten gesammelt, welche laut dem Nutzer zusammengehö-

ren. 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.

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 werden nun einige der technischen Konzepte angesprochen. Dabei geht es allgemein um die Funktionen eines Pub/Subs, mögliche Verschlüsselungsmethoden sowie um einige der Besonderheiten der technischen Umsetzung für die Actaeon Bibliothek.

5.1.1 Verschlüsselung

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 Verschlüsselungsbibliothek NaCl, beziehungsweise deren modernen Variante `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, dessen 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, die Verschlüsselung und das Adresssystem direkt miteinander zu verbinden. Das mag auf den ersten Blick etwas umständlich und unlogisch wirken, es erlaubt allerdings, verschlüsselte Nachrichten ohne weitere Operationen direkt an Nutzer zu schicken, wobei lediglich dessen Adresse bekannt sein muss. Insgesamt fällt damit viel Komplexität weg und das Erstellen, Verifizieren und Finden von Adressen wird 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 zu 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 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 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 durch die Checksumme des Themenschlüssels ersetzt, lässt sich Kademia ohne weitere Veränderungen für einen dezentralen PubSub einsetzen.
- Danach bleibt noch das Problem der Nachrichtenverbreitung. Dafür gibt es verschiedene Möglichkeiten:
 - Die Nachrichten werden direkt an ein zuständige Mitglied gesendet, von dort werden sie weitergeleitet. Vorteilhaft an diesem Konzept ist, dass die Verwender des Systems 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 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 genauere Kontrolle über die Auswahl der Abonnenten.

2. Probleme: Zwar haben PubSubs als Systemkonzept viele Vorteile, 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 über-

tragen werden und daher Verschlüsselung nahezu zwingend benötigt wird, muss man sie auch hier berücksichtigen. Wie bereits im Abschnitt zur Verschlüsselung angesprochen, werden Nachrichten mit dem öffentlichen Schlüssel des Empfängers, welcher auch gleichzeitig die Adresse darstellt, verschlüsselt. 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 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, weswegen der gesamte Pub/Sub Nachrichtenverkehr nicht verschlüsselt werden könnte.

- 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. Die Daten 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 komplex, wenn man also plötzlich nebst einem Nachrichtensystem ohne verteilte Zustände auch noch eine verteilte Datenbank verwalten muss, übersteigt dies meist die 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 Mitglieder im System
- Funktionen zum Berechnen der nächsten Ziele von Nachrichten

Um die Funktionen des Routers zu verstehen, lohnt es sich, den Abschnitt zu Kademia, oder besser noch das tatsächliche Kademia Whitepaper, zu lesen, denn der Router ist der Punkt im System, der die meisten der Kademia spezifischen Funktionen übernimmt. Als erstes ist es wichtig zu verstehen, welche Daten im Router überhaupt gespeichert werden. Grundsätzlich wird im Router jedes Mitglied gespeichert, von dem Daten empfangen werden, oder das durch eine Anfrage gefunden wurde. Allerdings definiert Kademia gewisse Regeln, nach welchen neu gefundene Mitglieder zum Routing Table hinzugefügt werden:

- Grundsätzlich werden alte, bereits seit langem bekannte Mitglieder gegenüber neueren bevorzugt, selbst wenn diese ansonsten bessere Eigenschaften (kleine Distanz) hätten. Damit wird verhindert, dass das Netzwerk durch eine Vielzahl böswillig generierter Mitglieder angegriffen werden kann.
- Für jedes Mitglied wird ein Status gespeichert, welcher angibt, ob das Mitglied aktuell erreichbar ist. Da sich dies während der Laufzeit ändern kann, muss der Status regelmässig überprüft werden.
- Die Anzahl der gespeicherten Mitglieder ist umgekehrt proportional zum Abstand zwischen dem Mitglied 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 Bucket dar, in welchem die tatsächlichen Mitglieder 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 die Regel, dass die Häufigkeit der Mitglieder mit steigendem Abstand abnimmt umgesetzt. Sollte es nicht möglich sein, einen Bucket zu teilen, werden neue Mitglieder entweder vergessen, oder bereits vorhandene Mitglieder 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 Mitglieder aus dem gleichen Bucket zu verwenden. Nur wenn diese nicht ausreichen werden Mitglieder aus anderen Buckets verwendet.

Der Router bietet alle nötigen Funktionen direkt in einer öffentlichen API an, in der tatsächlichen Applikation wird er allerdings auf eine andere Weise 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, also gleiche Daten bei mehreren Mitgliedern, 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 sich 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 werden

allerdings 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 bereits 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, beispielsweise der Hash eines Namens, 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, die auch aktuell Actaeon verwendet, ist 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

Während der ersten Entwicklungsphase wurde erkannt, dass trotz aller technischer Innovationen dezentrale Datensysteme für Aussenstehende nicht leicht zugänglich sind. Genau aus dieser Problematik entstand das Orion Videospiel. Im ersten Moment mag es nicht besonders logisch wirken, mit einem Videospiel gegen die grossen Probleme der übermässigen Zentralisierung und technischen Fehler unserer aktuellen Infrastruktur vorzugehen. Allerdings muss erkannt werden, dass niemand so einfach bereit ist, seine aktuelle Chat-, Datenspeicher- oder Automatisierungssysteme durch neuartige, dezentrale Alternativen zu ersetzen. Aber die Chance, dass jemand bereit ist, ein neues Videospiel auszuprobieren und dabei etwas über dezentrale Datensysteme und deren Vorteile lernt ist einiges höher.

Als Spiel wurde ein Flugsimulator mit einer `low-poly` Grafik gewählt. Obwohl verschiedene, sehr komplexe Flugsimulatoren bereits existieren und ein Flugsimulator nicht wie die einfachste Demo für ein zeitlich so begrenztes Projekt wirken mag, gibt es doch einige Vorteile, die einen Flugsimulator perfekt für eine solche Demo machen:

- Es werden keine hochauflösenden Charaktermodelle und Figuren benötigt. Tatsächlich haben die meisten bekannten Flugsimulatoren keinen besonders detaillierten Grafikstil.
- Es ist eher einfach, den Spielern Aufgaben zu stellen und diese spannend zu gestalten. Für dieses Spiel bedeutet dies, Spieler müssen durch Ringe fliegen, um Punkte zu machen. In einer zukünftigen Version wird zusätzlich noch die Möglichkeit eingebaut, andere Flugzeuge anzugreifen.

- Umgebungen und Welten können gut mit prozeduraler Generation von Terrain umsetzen. Damit kann mit wenig Aufwand für den Entwickler eine abwechslungsreiche Spielerfahrung gebaut werden. Solches Terrain eignet sich ebenfalls optimal für den gewählten Grafikstil.
- Viele Flugsimulatoren beschränken sich auf einen einzelnen Spieler oder sind limitiert, wenn es um die Anzahl Spieler pro Runde geht. Mit einem dezentralen Datensystem kann genau hier eine neuartige Alternative angeboten werden.

Als zentrale Demo des gesamten Projekts erhielt das Videospiel den Namen Orion. Auch hier ist der gesamte Code online auf Github verfügbar. Um Actaeon möglichst einfach in das Spiel zu integrieren, ist auch dieses in Rust geschrieben. Intern wird wgpu als Grafik API und Bevy als Framework verwendet, welches sich um die Verwaltung aller relevanter Daten kümmert.

Zwar kann die Grafik dieses Spiels kaum mit den modernsten Titeln mithalten, trotzdem bietet es einen gut ersichtlichen, schönen Stil, der mit Einstellungen den Möglichkeiten des jeweiligen Systems angepasst werden kann.



Screenshot der Landschaft im Orion Videospiel.

Der Screenshot zeigt das Terrain, welches vollständig zufällig generiert wird. Damit wird eine abwechslungsreiche Umgebung garantiert. Dazu ist auf dem Bild ebenfalls ein roter Ring zu sehen. Dieser stellt das aktuelle Spielziel dar. Spieler müssen durch zufällig generierte Ringe fliegen, wodurch sie Punkte erhalten. Dabei sieht die Perspektive des Nutzers allerdings etwas anders aus. Um den Effekt eines Flugsimulators deutlicher zu machen, wird zusätzlich das Innere eines Cockpits angezeigt. Aktuell ist dieses noch ein einfaches Bild, in Zukunft sollen sich die Werte und Anzeigen aber dem tatsächlichen Spiel anpassen.



Screenshot der Cockpit Ansicht im Orion Videospiel.

Gleichzeitig ist auf diesem Bild das rudimentäre UI zu erkennen, mit welchem Spieler über ihren aktuellen Punktestand informiert werden.

Wichtig für diese Arbeit ist das Modul *network.rs*. Das Modul kümmert sich um die Integration mit Actaeon. Insgesamt hat das Modul eine Länge von 140 Zeilen und definiert die Zugangsmöglichkeiten zur Actaeon Bibliothek und das Nachrichtenformat, mit welchem Nachrichten versendet werden. Normalerweise macht der Multiplayer einen grossen Teil der Komplexität eines Videospiels aus, doch dank der Actaeon Bibliothek ist der Aufwand äusserst überschaubar.

5.2.1 Umsetzung

In diesem Abschnitt werden einige der technischen Feinheiten der Umsetzung des Spiels angeschaut. Dabei geht es aber nicht um das Spiel als ganzes, sondern lediglich um die Integration mit Actaeon. Das wichtigste Objekt ist dabei das *Network*

```
#[derive(Component)]
pub struct Network {
    center: Center,
    interface: Mutex<Interface>,
    topic: Option<Mutex<Topic>>,
}
```

welches direkt beim Start der Applikation erstellt wird, und alle wichtigen Interaktionspunkte mit der Actaeon Bibliothek verwaltet. Dieses Objekt wird dann in einer unendlichen Schleife nach neuen Nachrichten befragt und kümmert sich darum, die Eingaben des Nutzers an alle verbundenen Spieler weiter zu geben. Im Abschnitt zur Actaeon Bibliothek wurde die Funktionsweise eines PubSubs erklärt. Aktuell verwendet Orion ein einziges Thema, über welches alle Spieler kommunizieren. In Zukunft könnte dies aber auf verschiedene Themen ausgeweitet werden. So könnte sich jedes Thema beispielsweise

um einen bestimmten Abschnitt der Spielwelt kümmern, womit sich die Anzahl möglicher Spieler stark erhöhen lässt.

Ebenfalls wird ein Nachrichtenformat definiert. Diese Nachrichten werden dann zwischen den einzelnen Nutzern versendet und sind komplett unabhängig von den Actaeon Strukturen.

5.2.2 Geschwindigkeit

Aktuell leidet die Demo noch unter starken Leistungsproblemen. Auf schwachen Geräten ist das Spiel kaum auszuführen und selbst auf neueren Geräten kommt es häufig zu kleineren Leistungseinbrüchen. Das Optimieren von Videospielen ist ein äusserst komplexes Gebiet, mit welchem selbst grosse Firmen und Entwickler immer wieder Probleme haben. Als zentrales Problem wurde bei Orion bereits die Generierung der Spielwelt identifiziert. Durch ein intelligentes System, welches automatisch die verschiedenen Abschnitte der generierten Welt verbindet, wieder trennt und sich darum kümmert, diese zum richtigen Zeitpunkt zu laden oder zu entladen, könnte die Leistung um ein Vielfaches gesteigert werden. Die beste Lösung zum aktuellen Zeitpunkt ist es, die Auflösung der Oberflächen zu reduzieren, allerdings entsteht dadurch ein sichtbarer Unterschied im Detailgrad und im Aussehen der Welt.

5.3 Anwendungen

In den folgenden Abschnitten werden die beiden anderen Demos angesprochen. Während Actaeon als zentrales Produkt und Orion als zentrale Demo für dieses Produkt den Hauptteil dieser Arbeit ausmachen, war es trotzdem ein Ziel, auch andere Einsatzmöglichkeiten beispielhaft zu zeigen.

5.3.1 Chat

Wie bereits in der ersten Entwicklungsphase auch 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. 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 Konsolenapplikation 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 wird, 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 abgedruckt und wird dort auch kurz erklärt.

5.3.2 Arrow

Schon in der ersten Entwicklungsphase entstand eine, speziell auf das Verarbeiten von Nachrichten ausgelegte, Programmiersprache, damals unter dem Namen NET-Script. Ziel dieser Sprache war es nie, eine fertige Alternative zu bekannten Sprachen wie Python oder Rust zu bieten. Stattdessen ging es eher um die einfache Integration zwischen von Nutzern entwickelten Programmen und dem dezentralen Datensystem. Besonders dafür gab es in der Programmiersprache verschiedene Funktionen besonders für das Verarbeiten, Manipulieren und Senden von Nachrichten über das dezentrale Datensystem.

Die ursprüngliche Umsetzung der Programmiersprache war noch auf die ursprünglichen Prinzipien der Modularität ausgelegt. Daher kommunizierte die Sprache selbstständig mit den anderen Komponenten. Auch wenn sich die Prinzipien und Ziele für die zweite Entwicklungsphase änderten, liess sich die existierende Implementierung von NET-Script ohne grossen Aufwand auf die neuen Bedingungen anpassen. Anstelle einer unabhängigen Kommunikation mit den restlichen Komponenten integrierte die neue Applikation Actaeon direkt als Rust Abhängigkeit. Dieser Wechsel wurde zusätzlich mit einer Namensänderung symbolisiert. Unter dem neuen Namen Arrow lässt sich die Programmiersprache jetzt als alleinstehende Applikation für das Verarbeiten von Nachrichten in beliebigen Actaeon Systemen einsetzen. Der gesamte Code ist wie alle anderen Produkte auch online auf Github und unter einer freien Lizenz verfügbar.

An der technischen Umsetzung hat sich gegenüber der ersten Entwicklungsphase tatsächlich nicht viel verändert. Eine genaue Besprechung der Umsetzung oder des Codes würde den Umfang dieser Arbeit übersteigen und ist auch nicht nötig, um die Applikation zu verwenden. Um die Programmiersprache zu verwenden, werden lediglich die folgenden Befehle benötigt:

```
$ git clone https://github.com/EngineOrion/arrow
$ cd arrow
$ cargo run --release
```

Von dort aus lassen sich nun beliebige Scripts schreiben, welche mit geringem Aufwand auf Ereignisse des Systems reagieren können. Die Syntax orientiert sich dabei an der bekannten Programmiersprache Lisp, was den Einstieg erleichtern solle. Nachdem das Programm gestartet wurde, lassen sich in einer interaktiven Umgebung Subroutinen und Variablen definieren und der Nutzer hat Zugang zu den bereits bekannten Funktionen des Actaeon Systems. Im Github Repository werden einige beispielhafte Befehle gezeigt.

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 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 haufenweise Vorteilen mit sich bringen. Seien es zensursichere Speicher- und Nachrichtensysteme oder zum Beispiel die schiere Grösse von 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 umfassende 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 erleichtert 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.

Im Anhang dieses Dokuments ist der definierte Vertrag zu finden, mit welchem die Ziele des Projekts definiert wurden. Die beschriebenen Produkte erfüllen alle diese Ziele. Es existieren drei vollständige Demos, welche alle das Actaeon System für den dezentralen Nachrichtenaustausch verwenden. Die Bibliothek ist online verfügbar und gut dokumentiert. Eine Unstimmigkeit ist die *Orion Game-Engine*. Der Vertrag definiert, dass ein Videospiel komplett ohne ein spezialisiertes Programm für die Entwicklung von Spielen umgesetzt werden sollte. Aktuell ist dies nicht der Fall.

Tatsächlich existiert eine frühere Version des Orion Games, welches komplett ohne *Game-Engine* auskommt. Es wurde aber schnell klar, dass ein solches Unterfangen den Umfang

dieser Arbeit weit übersteigen würde. Da ein grosser Fokus auf einer grafischen Demo liegt, wurde die Entscheidung getroffen, eine Game-Engine zur Hilfe zu nehmen. *Bevy*, die gewählte Game-Engine, ist dabei kaum vergleichbar mit bekannteren Beispielen wie Unity oder Unreal. Stattdessen kümmert sich Bevy lediglich um die Datenverwaltung und bietet keine vereinfachte, grafische Entwicklungsumgebung.

Anhang

7.1 Ausblick

Auch wenn 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 nahezu jedes 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 können:
 - Anstelle des aktuellen Threading-Models muss der Wechsel auf die neuen Rust `async-await` Funktionen gemacht werden. Zwar entstehen damit mehr Abhängigkeiten und grössere Komplexität, allerdings sollte sich dieser Schritt lohnen, da eine solche Umsetzung signifikant schneller läuft.
 - Als weiteres grosses Feature soll *Record-Republishing* eingebaut werden. Damit funktioniert das Netzwerk auch bei abrupten, grossflächigen Änderungen der Netzwerkstruktur, was aktuell noch zu Problemen führt.
- Orion:
Nebst der offensichtlichen Frage der Performance, gibt es auch ein grundlegendes 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 der 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 die Hoffnung, dass andere Personen in der Zukunft Programme für das Ökosystem bauen.

7.2 Recherche

In diesem Abschnitt werden einige andere Projekte mit ähnlichen Zielen genauer angeschaut. 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.

7.2.1 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. Zwar 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, dass jemand wie Parker, der nur wenige Jahre zuvor mit Napster die komplette Musikindustrie in Unruhe gebracht hatte, später eine wichtige Rolle bei Facebook, einem der zentralisiertesten Megaunternehmen der Welt, einnehmen konnte.

Auch wenn es noch nicht *vollständig* dezentralisiert war, 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 Kademliasystem 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 Anbie-

tern gleichzeitig herunterladen lassen. Da es sich bei BitTorrent eigentlich um ein grosses Dateisystem handelt, lassen sich direkt die SHA1-Hashwerte der Inhalte als Kademia-Adressen verwenden.

7.2.2 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 End Verschlüsselt. Intern wird ein dezentrales Routing System, basierend auf Kademia, verwendet, mit welchem Kontakt zwischen Nutzern (Freunden) aufgebaut wird. Während im Kademia 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 Kommunikation, 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 Kademia 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. Zwar ist der Inhalt weiterhin verschlüsselt, aber ein solches System setzt in erster Linie auf 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.

7.2.3 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. [11] 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.

7.3 Code

Der Code aller Produkte beider Entwicklungsphasen ist online auf Github verfügbar. Die verschiedenen Repositories sind unter einer Organisation gruppiert: github.com/EngineOrion. Dort sind die verschiedenen Applikationen, das Arbeitsjournal sowie dieser Abschlussbericht in je einzelnen Repositories zu finden. Dazu ist Actaeon für Rust Entwickler veröffentlicht. Details über die Einsatzmöglichkeiten und die Code Dokumentation sind auf [crates.io](https://crates.io/crates/actaeon) verfügbar: crates.io/crates/actaeon

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 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 zwischen Personen oder öffentliches Verbreiten. Kann sowohl über dezentrale Programme wie Torrents, im direkten Austausch per E-Mail oder über einen 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 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:** Menschenlesbare 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 gewissem Anteil ihrer Adresse übereinstimmen.
- `Napster`: Eine der ersten Methoden um 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 Schlüssel.
- `Topic Record`: Topic Objekt an der rechnerisch korrekten Stelle in einem Actaeon Cluster. Kümmt sich um das Weiterverbreiten neuer Abonnenten.
- `Low-poly`: Welten, Figuren und Objekte in Videospielen werden aus vielen Dreiecken zusammengesetzt.

Abbildungsverzeichnis

3.1	Beispielhafte Darstellung eines einfachen Kademliasystems, Wikipedia: https://en.wikipedia.org/wiki/Kademlia ,	14
4.1	Engine: Orion Chat Website, intern verbunden mit dem dezentralen Kommunikationssystem.	19
5.1	Screenshot der Landschaft im Orion Videospiel.	30
5.2	Screenshot der Cockpit Ansicht im Orion Videospiel.	31

Literaturverzeichnis

- [1] J. Postel, "Internet protocol," STD 5, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc791.txt>.
- [2] L. E. Hughes, "Ipv4 and ipv6 address spaces." <https://thirdinternet.com/ipv4-and-ipv6-address-spaces/>, 2019. 21.11.2021.
- [3] Google, "Google privacy terms." <https://policies.google.com/privacy>. 21.11.2021.
- [4] T. Scott, "Single point of failure: The (fictional) day google forgot to check passwords." https://www.youtube.com/watch?v=y4GB_NDU43Q, 2014. 24.05.2021.
- [5] M. 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> 19.10.2021.
- [6] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," tech. rep., 2002.
- [7] D. Keller and J. Klemm, "Engine: Orion." <https://github.com/EngineOrion/kommentar/blob/54489464214ed7833f182df09aab29eae4a591e4/EngineOrion.pdf>, 2021.
- [8] P. Sloterdijk, *Den Himmel zum Sprechen bringen*. 2020.
- [9] L. Wang and J. Kangasharju, "Measuring large-scale distributed systems: Case of bittorrent mainline dht," tech. rep., 2013.
- [10] "Tox specs." <https://toktok.ltd/spec.html>, 2016. 23.11.2021.
- [11] C. J. DeLisle, "Cjdns whitepaper." <https://github.com/cjdelisle/cjdns/blob/f909b960709a4e06730ddd4d221e5df38164dbb6/doc/Whitepaper.md>, 2019.

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	