KANTI BADEN

Project Orion

Dominik Keller, Jakob Klemm, G4a

Maturaarbeit, Kanti-Baden
Simon Hallström

Baden, Schweiz
9. November 2021

Inhaltsverzeichnis

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 von der Weltherrschaft 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: Nebst einem dezentralen Kommunikationssystem sollen auch verschiedene, praktisch einsetzbare Anwendungen die Prinzipien 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 Datensysteme 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 sie nach technischem Detailgrad sortiert sind. Wer nur ein oberflächliches Verständnis über die Arbeiten und eine Analyse des Umfelds will, ohne dabei zu technisch zu werden, muss nicht über den Umfang dieses Dokuments hinaus. Aber wer vollständigen Einblick in die Errungenschaften und Konzepte erhalten will, muss gewisses Vorwissen und genügend Zeit mitbringen.

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

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

- Code: Neben der schriftlichen Dokumentation des Projekts existiert eine weitere Form der Dokumentation. Nahezu jede Funktion und jedes Modul über die verschiedenen *Crates* sind dokumentiert. Diese Dokumentationen lassen sich nicht in einem klassisch strukturierten Dokument finden, stattdessen ist die Code-Dokumentation online über automatisch generierte Rust-Dokumentation zu finden. Wer den Code von Project Orion verwenden will wird sich dort gut zurecht finden.

1 Vision

1. Oktober 1964 - UCLA an Stanford: 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 unser Gesellschaft gar nicht erst existieren. Das Internet hatte einen selbstverstärkenden Effekt auf sein eigenes Wachstum. Der um ein Vielfaches schnellere Datenaustausch und die enorme Interkonnektivität führten dazu, dass jede neue Innovation und jede neue Plattform noch schneller noch mehr User erreichte und auf immer unvorstellbarere Grössen anwuchs.

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

Egal wie sicher und zuverlässig unsere *öffentliche* Infrastruktur auch scheinen mag, es lassen sich doch schnell Risse im System erkennen. Nicht nur

an der Oberfläche, sondern auch im Herzen unseres digitalen Leben, gibt es Probleme. Oftmals handelt es sich dabei nicht um *Kleinigkeiten*, *Meinungsverschiedenheiten* oder *Kontroversen*, sondern um physikalische Grenzen, grundlegende Designfehler und das vielseitige Versagen der involvierten Parteien.

In den nächsten Kapiteln sollen einige dieser zentralen Probleme besprochen werden. Dabei soll versucht werden, nicht nur die fehlerhaften Implementierungen zu erklären, sondern auch die dadurch entstandenen Probleme in Verbindung mit unseren täglichen Interaktionen und Verwendungen des Internets zu bringen. In einem nächsten Schritt soll dann eine Lösung besprochen werden: ein System, mit welchem sich möglichst viele der grössten Probleme lösen lassen, und welches tatsächlich praktischen Nutzen bietet.

1.1 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-Addresse` 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.

1.1.1 IP-V4

In der Geschichte der Menschheit haben wir aus vielen verschiedenen Gründen Krieg geführt. Für Wasser, Nahrung, Öl, Frieden oder Freiheit in den Krieg zu ziehen, scheint zu einer fernen Welt zu gehören. Aber auch wenn diese grundlegenden Verlangen gedeckt sind, werden schon bald neue Nöte aufkommen. Während *Daten* oft als Gold des 21. Jahrhunderts bezeichnet werden, gibt es noch eine andere Ressource, deren Vorräte wir immer schneller erschöpfen.

$4'294'967'296$. So viele `IP-V4`-Adressen wird es jemals geben. `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 nichts Anderes. Mit `IP-V6` wird zurzeit eine Alternative angeboten, die solche Limitierungen nicht hat. Aber der Wechsel ist eine freiwillige Entscheidung, für die nicht nur alle Betroffenen bereit sein müssen, sondern für die auch jede einzelne involvierte Komponente diese neue Technologie unterstützen muss.

Für jeden Einzelnen kann dies verschiedene Konsequenzen haben:

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

1.1.2 Routing

Freiheit und Unabhängigkeit sind menschlich. Es darf niemals bestraft werden, nach diesen fundamentalen Rechten zu streben. Und doch führt das egoistische Streben nach Freiheit zu Problemen, oftmals allerdings nicht für die nach Freiheit Strebenden.

Genau diese Situation findet man im aktuellen Konflikt um die Grösse von *Address-Abschnitten* vor. Um dieses Problem richtig zu verstehen, muss als erstes die Funktion der *Zentralrouter* und der globalen Netzwerkinfrastruktur erklärt werden:

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

- Ganz unten finden sich die Blätter, die Abschlusspunkte der Struktur. Sie stellen die *Endnutzergeräte* 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 allerdings nicht um Geräte, die sich in einem persönlichen Setup oder einem normalen Haushalt finden. Mit Switches sind die Knotenpunkte (`POP-Switches`) der Internet-Anbieter gemeint. Diese teilen eingehende Datenströme auf und leiten die richtigen Daten über die richtigen Leitungen.

- Ganz oben findet sich der Stamm. Während ein normaler Baum natürlich nur einen Stamm hat, finden sich in der Infrastruktur des Internets aus Zuverlässigkeitsgründen mehrere. Von diesen `Zentralroutern` gibt es weltweit nur eine Handvoll und sie sind der Grund für das Problem.

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

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

1.2 Zentralisierung

Die Macht in den Händen einiger weniger Kapitalisten und internationaler Unternehmen ist unvorstellbar gross. Einige wenige CEO's, welche nie gewählt, überprüft oder zur Rede gestellt wurden, sind in voller Kontrolle unserer Leben. Egal welcher politischen, wirtschaftlichen oder gesellschaftlichen Ideologie jemand auch folgt, eine solche Abhängigkeit wirft gewisse Fragen und Probleme auf.

Aber neben den ideologischen Fragen und Sicherheitsbedenken gibt es auch noch sehr praktische Probleme in der Art, wie moderne Internet-Dienste implementiert sind.

1.2.1 Datenschutz

Wenn man nicht für etwas zahlt, ist man das Produkt.

Nach dieser Idee ist man für ziemlich viele Firmen ein Produkt. Doch leider muss man realisieren, dass man selbst bei kostenpflichtigen Diensten als Produkt gesehen wird. Denn das Internet hat einen neuen Rohstoff zur Welt gebracht. Wer viele Daten über Menschen besitzt, bekommt binnen kürzester Zeit Macht.

In ihrer einfachsten Funktion werden Daten für personalisierte Werbung eingesetzt. 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 damit lassen sich 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 uns im Internet anhäufen, zeigen mehr als unser Kaufverhalten. Von Echtzeit-Positionsupdates, Anrufen und Suchanfragen bis hin zu privaten Chats und unseren tiefsten Geheimnissen, sind wir meist überraschend unvorsichtig im Umgang mit digitalen Werkzeugen.

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

- Selbst für die einfachsten Anfragen im Internet sind wir von einer Vielzahl von Firmen und Systemen abhängig. Dieses Problem wird noch etwas genauer im Abschnitt Abhängigkeit besprochen.

- Wir haben weder ein Verständnis von den involvierten Parteien noch die Bereitschaft, Bequemlichkeit dafür aufzugeben.

1.2.2 Abhängigkeit

In einem fiktionalen Szenario[1] erklärt *Tom Scott* auf seinem YouTube-Kanal, was passieren könnte, wenn eine einzelne Sicherheitsfunktion beim Internetgiganten `Google` fehlschlagen würde. In einem solchen Fall ist es natürlich logisch, dass es zu Problemen bei den verschiedensten `Google`-Diensten kommen würde. Aber schnell realisiert man, auf wie vielen Seiten Nutzer die *Sign-In with Google* Funktion benutzen. Und dann braucht es nur eine böswillige Person um den Administrator-Account anderer Dienste und Seiten zu öffnen, wodurch die Menge an Sicherheitsproblemen exponentiell steigt.

Aber es muss nicht immer etwas schief gehen, um die Probleme zu erkennen. 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 natürlich 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[2], 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 Dieste überrascht werden können. Der wirtschaftli-

[1] Tom Scott: Single Point of Failure `https://youtu.be/y4GB_NDU43Q`, heruntergeladen am 24.05.2020.

[2] Fastcompany: Google outage `https://www.fastcompany.com/90358396/that-major-google-outage-meant-some-nest-users-couldnt-unlock-doors-or-use-the-ac`, heruntergeladen am 24.10.2021.

che Schaden solcher Ausfälle ist unvorstellbar, aber noch wichtiger muss die zerstörende Wirkung dieser unvorhergesehenen, scheinbar entfernten Problemen auf Millionen von Menschen bedacht werden.

1.3 Komplexität

In diesem Abschnitt soll noch kurz die unglaubliche Komplexität angesprochen werden, welche die heutige Web-Entwicklung mit sich bringt. Natürlich existieren automatisierte Dienste und Anbieter, die den Prozess vereinfachen. Wer aber Wert auf seine Privatsphäre und auf die Verwendung von open-source Software legt, muss sich um vieles selbst kümmern. Nicht nur die Auswahl an verschiedenen Programmen kann erschlagend wirken, sondern der Fakt, dass diese untereinander kompatibel sein müssen. Zwar reden wir oft von einem Webserver, allerdings sind es tatsächlich viele verschiedene Programme, die alle fehlerfrei miteinander interagieren müssen, um Resultate zu liefern. Dies kann den Einstieg schwer machen, in gefährlicheren Fällen kann es dazu führen, dass Sicherheit und Datenschutz aus Zeit- oder Komplexitätsgründen weggelassen oder vernachlässig werden.

Dabei geht es oben nur um *klassische* Webseiten oder Webserver. Die Welt der dezentralen Technologien ist im Vergleich dazu wie der wilde Westen, ohne Standards, ohne Kompatiblität oder Regelungen. Dies führt dazu, dass es zwar für gewisse Anwendungen speziell entwickelte Netzwerke gibt, diese allerdings kaum allgeimen einsetzbar sind.

1.4 Präsentation

Ein weiteres Problem, das es zu berücksichtigen gibt, ist die Frage, wie man die hier behandelten Probleme technisch nicht versierten Personen erklären kann. Tatsächlich sind sowohl die besprochenen Probleme, als auch deren Lösungsansätze nicht nur abstrakt, sondern dazu noch Teil einer kleinen Nische in der Welt der Informatik. Manche der angesprochenen Probleme wurden bereits von anderen Applikationen zumindest teilweise behandelt, diese haben aber oftmals das Problem, dass sie viel Fachwissen und Aufwand benötigen, um sie effizient und sicher einzusetzen.

2 Prozess

Oftmals ist es nicht besonders spannend, über den Prozess einer Programmierarbeit zu hören, denn für Aussenstehende scheint sich von Tag zu Tag

nichts zu ändern. Sinnvolles kann erst berichtet werden, wenn der Zeitrahmen erhöht wird, sodass grössere Entscheidungen und ihre Konsequenzen sichtbar gemacht werden können. Die Entwicklung und die Produkte dieser Arbeit sollen in zwei Abschnitte getrennt werden, wobei die meisten Produkte aus der zweiten Phase hervorgegangen sind.

2.1 Modularität

Da während dieser ersten Entwicklungsphase viele wichtige Erkenntnisse entstanden, ist es wichtig, die Ideen und die Umsetzung genau zu analysieren. Zwar unterscheiden sich die Ziele und Methoden der beiden Ansätze stark, gewisse Konzepte und einige Programme aber lassen sich für die aktuelle Zielsetzung vollständig übernehmen.

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

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

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

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

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

2.1.1 Shadow

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

Sobald eine Nachricht im System am Ziel angekommen war, wurde sie über einen `Unix-Socket` an den nächsten Komponenten im System weitergegeben. Dies geschah nur, wenn das einheitlich verwaltete Registrierungssystem für Personen und Dienste, eine Teilfunktion von Hunter, ein Resultat lieferte. Ansonsten wurde der interne Routing-Table verwendet. Dieser bestand aus einer Reihe von Prozessen, welche selbst auch direkt die TCP-Verbindungen verwalteten.

2.1.2 Hunter

Während Shadow die Rolle des verteilten Routers übernimmt, ist `Hunter` der lokale Router. Es geht bei Hunter also nicht darum, Nachrichten an andere Mitglieder des Netzwerks zu senden, sondern sie an verschiedene Applikationen auf der gleichen Maschine zu senden. Jedes beliebige Programm, unabhängig von Programmiersprache und internen Strukturen, müsste dann also nur das verhältnismässig Protokoll implementieren und wäre damit in der Lage, mit allen anderen Komponenten zu interagieren. Anders als Shadow wurde Hunter komplett in Rust entwickelt und liess sich in zwei zentrale Funktionen aufteilen:

- Zum einen diente das Programm als Schnittstelle zu einer einfachen *Datenbank*, in diesem Fall eine `JSON-Datei`. Dort wurden alle lokal aktiven Adressen und die dazugehörigen Applikationen gespeichert. Ein Nutzer, der sich beispielsweise über einen Chat mit dem System verbindet, wird dort mit seiner Adresse oder seinem Nutzernamen und dem Namen des Chats eingetragen. Wenn dann von einem beliebigen

anderen Punkt im System eine Nachricht an diesen Nutzer kommt, wird der passende Dienst aus der Datenbank gelesen. All dies läuft durch ein *Command Line Interface*, welches dann ins Dateisystem schreibt.

- Das eigentliche Senden und Weiterleiten der Nachrichten war nicht über ein kurzlebiges Programm möglich, da dafür längere Verbindungen existieren müssen. Deshalb muss Hunter als erstes gestartet werden, wobei das Programm intern für jede Verbindung einen dedizierten Thread startet.

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

2.1.3 NET-Script

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

2.1.4 Probleme

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

Trotzdem wurden die beschriebenen Komponenten vollständig entwickelt, getestet und vorgeführt. Zwar war es umständlich und nur bedingt praktisch

einsetzbar, trotzdem war es aber eine technisch neuartige, funktionsfähige Lösung für komplexe und relevante Probleme. Nachdem die erste Entwicklungsphase erfolgreich abgeschlossen wurde, kam allerdings noch ein weiteres Problem auf, welches die folgenden Entscheidungen stark beeinflusst hat. Es ist ein Problem, welches sich auf die grundlegende Natur der Informatik zurückführen lässt:
Anders als in nahezu allen Studienrichtungen, Wissenschaften und Industrien, werden in der Informatik die gleichen Werkzeuge verwendet und entwickelt. Wer die Werkzeuge der Informatik verwenden kann, ist gleichzeitig in der Lage (zumindest bis zu einem gewissen Grad) neue Werkzeuge zu entwickeln. Diese Eigenschaft erlaubt schnelle Iterationen und viele fortschrittliche Werkzeuge, kommen gleichzeitig neue Probleme auf:

- Neue Methoden und Werkzeuge werden mit unglaublicher Geschwindigkeit entwickelt und verbreitet. Wer also nicht mit den neusten Trends mithält, kann schnell abgehängt werden. Dies macht auch das Unterrichten besonders schwer.

- Natürlich werden die Werkzeuge meistens immer besser und schneller, allerdings kommt es oftmals auch zu einer Spezialisierung. Dies führt schnell zu immer spezifischeren, exotischeren Lösungen und unzähligen Unterbereichen und immer kleineren Gebieten. So ist beispielsweise der Begriff *dezentrale Datensysteme*, der zwar ein einzelnes Gebiet genau beschreibt, für Aussenstehende mehrheitlich bedeutungslos und sorgt für mehr Verwirrung als Aufklärung.

- Die immer neuen Gebiete und Gruppen können auch schnell zu Elitismus führen, wodurch es für Anfänger schwer sein kann, Zugang zu finden.

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

Da am Ende dieser Arbeit zwingend eine zeitlich begrenzte Präsentation vor einem technisch nicht versierten Publikum steht, mussten nach dieser

ersten Entwicklungsphase gewisse Aspekte grundlegend überarbeitet werden, diesmal mit einem besonderen Fokus auf die *Präsentierbarkeit*.

2.2 Präsentation

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

- Ein möglichst vielseitig einsetzbares Nachrichtensystem basierend auf den bereits bekannten Prinzipien wird als Bibliothek für die Anwendungen öffentlich angeboten. Entwickelt in Rust wird Geschwindigkeit und Sicherheit garantiert und es lassen sich möglichst viele Möglichkeiten finden, Integrationen in andere Projekte und Applikationen zu ermöglichen..
- Aufbauend auf diesem Datensystem sollen mit verschiedenen Anwendungen die Vorteile und vielseitige Einsatzmöglichkeiten gezeigt werden. Auch wenn damit die weltverändernde Revolution noch nicht direkt gestartet wird, so wird ein Aspekt angesprochen, welcher in technischen Kreisen oftmals vergessen geht, nähmlich die Frage, wie man komplexe Themen und Programme einfachen Nutzern näher bringt.

3 Produkt

Nun ist es an der Zeit, die Errungenschaften und Produkte des Projektes anzuschauen. Eigentlich müssten auch hier die Programme der ersten Entwicklungsphase besprochen werden, dies geschah allerdings bereits grösstenteils während der Analyse des Arbeitsprozesses. Wer noch mehr über die Programme erfahren will sollte sich am besten den dazugehörigen Bericht

durchlesen[3] oder direkt die dabei entstandenen Programme anschauen. Alle lassen sich auf Github finden und sind unter offenen Lizensen einfach weiterzuverwenden[4]. Auch muss angemerkt werden, dass viele der dabei entstandenen Ideen viel Potential haben und in gewissem Ausmass ihren Weg bereits in die Produkte der zweiten Phase gefunden haben.
Dieses Kapitel ist in zwei Abschnitte geteilt:

- `Actaeon` beschreibt das dezentrale Datensystem, die Rust Bibliothek sowie einige der einfachen Anwendungen, wie zum Beispiel ein Chat-Client die ohne grossen Aufwand darauf aufgebaut werden können.

- Da das Videospiel als Vorzeigebeispiel für das gesamte Projekt entwickelt wurde, erhielt es den gleichen Namen wie die gesamte Arbeit.

Hier muss nochmals der Umfang dieser Arbeit angesprochen werden. Dieses Dokument soll mit möglichst wenig Vorwissen verständlich sein und nur oberflächlich die technischen Feinheiten ansprechen. Wer genauere Informationen zur Umsetzung sucht oder tatsächlich mit den Bibliotheken arbeiten will, sollte sich die technische Dokumentation oder Code-Dokumentation durchlesen.

3.1 Actaeon

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

Als erstes muss aber wahrscheinlich kurz der Name angesprochen werden, da sich auch hier einiges an Bedeutung dahinter versteckt. In der griechischen Mythologie ist Actaeon (auch Aktaion) ein Jäger, der mit seinen Hunden durch den Wald streift und die Göttin der Jagt beim Bad in einer Quelle überrascht. Die Göttin verwandelt Actaeon in einen Hirsch, der daraufhin von seinen eigenen Hunden in Stücke gerissen und gefressen wird. Der Philosoph Peter Sloterdijk, der sich dabei auf Giordano Bruno bezieht, deutet die Geschichte von Actaeon als Gleichnis für die menschliche Suche nach Wahrheit. Ein Blick auf die ganze, göttliche Wahrheit ist für einen Menschen nicht möglich. Wer das versucht, wird vom Jäger zum Gejagten, wird

[3] Github: Engine: Orion Bericht, `https://github.com/EngineOrion/kommentar/blob/54489464214ed7833f182df09aab29eae4a591e4/EngineOrion.pdf`.
[4] Github: `https://github.com/EngineOrion`

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.[5]

Nun müssen einige Grundlagen erklärt werden, welche hoffentlich die folgenden Abschnitte verständlich machen:
Nahezu alle Applikationen in Project Orion sind in der Programmiersprache Rust geschrieben. Als zentrales Verbindungstück zwischen allen Applikationen hängen alle Programme von Actaeon ab. Um die Verwaltung der internen Abhängigkeiten einfacher zu machen, wurde Actaeon mit dem offiziellen Rust *Package-Manager* `crates.io`[6] veröffentlicht. Dies erlaubt es jedem Rust Entwickler Actaeon einfach in eigene Programme zu integrieren, dafür ist lediglich eine Zeile Code nötig:

```
actaeon = "0.2.0"
```

Da es sich hier um eine Bibliothek handelt, welche dann in andere Programme eingebunden werden soll, ist es nicht von Nöten, die interne Funktionsweise vollständig zu verstehen. In der technischen Dokumentation werden die Ideen und Entscheidungen, die zu Actaeon führten genauer angeschaut. Hier soll es allerdings hauptsächlich um einen groben Überblick gehen, sowie um die API, mit welcher mit der Bibliothek interagiert werden kann.

Wahrscheinlich lohnt es sich aufzulisten, welche Funktionen die Bibliothek übernimmt, und welche weiterhin vom Nutzer verlangt werden. Actaeon übernimmt:

- Verbindungen (stateful & stateless): Das System entscheidet intern selbständig, wie Nachrichten versendet werden sollen. Entweder es werden einzelne Nachrichten an andere Mitglieder geschickt, oder es kann eine langlebige Verbindung aufgebaut werden, welche für mehrere Nachrichten verwendet werden kann.

- Adressen: Anstelle von IP-Adressen oder UUIDs verwendet das System ein eigenes, vielseitiges Adresssytem. Dieses wird auf verschiedene Arten eingesetzt:

  - Identifikation: Jede Adresse identifiziert ein Objekt (Nutzer oder Thema) eindeutig. Dabei gibt es zwar keinen Mechanismus um

[5] Sloterdijk, Peter: Den Himmel zum Sprechen bringen, 2020, S321.
[6] Crates.io: `https://crates.io/`

dies zu garantieren, die kleine Wahrscheinlichkeit einer Kollision über die 32 Bytes reicht allerdings aus.

  - Routing: Mit den Adressen lässt sich ebenfalls rechnen, wobei es hauptsächlich um die Kademlia `XOR`-Operation geht. Damit werden Distanzen und die Wege sowie Orte für Nachrichten bestimmt.
  - Verschlüsselung: Dank der Familie der `NaCl`-Verschlüsselungsbibliotheken ist es möglich, öffentliche und private Schlüssel mit einer Länge von nur 32 bytes zu haben. Damit lassen sich im System Nachrichten automatisch End-zu-End verschlüsseln, ohne einen dedizierten Mechanismus zum Austausch von öffentlichen Schlüsseln zu benötigen.

- Signaling: In Netzwerken ist es immer eine wichtige Frage, wie neue Nutzer beitreten können. Actaeon verlangt dabei lediglich die Kontaktdaten von einem bekannten Mitglied des Systems und kümmert sich dann um den Rest.

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

Natürlich übernimmt die Bibliothek noch viele weitere Funktionen, ein genauerer Funktionsumfang findet sich in der technischen Dokumentation. Nun muss man sich fragen, welche Funktionen potentielle Nutzer überhaupt übernehmen müssen. Dafür lohnt es sich die einfachste Beispielanwendung anzuschauen (Dieser Code ist ebenfalls im Readme und der Dokumentation zu finden):

```
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 dann intern verschiedene Threads gestartet und die Initialisierung des Routing-Tables (Bootstrapping) beginnt. Der Nutzer muss sich dabei um nichts kümmern, ausser potentiell aufkommende Fehler handhaben. Das zweite wichtige Element ist dabei das `Topic`, denn die meisten Interaktionen des Nutzers finden über ein solches Thema statt. Für genauere Informationen empfiehlt sich die technische Dokumentation. Als einfache Zusammenfassung lässt sich sagen: Unter einem Thema werden Nachrichten gesammelt, welche laut dem Nutzer zusammengehören. Dabei ist wichtig festzustellen, dass die Wahl der Themen vollkommen in der Hand des Nutzers liegt, die Bibliothek hat keinen Einfluss darauf. Sobald aber ein solches Thema erstellt wurde, gibt es eigentlich nur zwei wichtige Aktionen:

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

Nur mit diesen beiden Operationen ist die Menge an potentiellen Applikationen nahezu unbegrenzt, denn nahezu alle Interaktionen, Programme und Netzwerke lassen sich irgendwie mit diesen beiden Befehlen umsetzen. Natürlich gibt es noch mehr interne Funktionen und auch mehr Möglichkeiten für den Nutzer, diese gehen aber über den Umfang dieses Dokuments hinaus.

3.2 Orion

TODO: Dominik Orion

4 Ausblick

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

- Actaeon:
  Nebst den technischen Feinheiten, wie eine Möglichkeit einzelne Threads unabhängig voneinander neuzustarten oder eine zusätzliche Ebene der Verschlüsselung, gibt es einige grössere, strukturelle Probleme, welche noch angesprochen werden müssen. Besondes die mehrfach angesprochenen Probleme im Zusammenhang mit IP-Adressen haben mit unseren Arbeiten keine abschliessene Lösung gefunden. Die Errungenschaft hier ist Unabhängigkeit, denn das Netzwerk ist nicht an gewisse Adress-Systeme gebunden oder auf nur eines limitiert. Auch wenn diese gewonnene Unabhängigkeit eine Errungenschaft für sich ist, so werden die allermeisten Nutzer weiterhin problematische Systeme, besonders IP-V4, verwenden. Zwar ist es fragwürdig, wie viel eine zeitlich stark limitierte Arbeit in einem so umfangreichen Feld tatsächlich erreichen kann, es würde sich allerdings lohnen, zumindest eine existierende Alternative, wie beispielsweise `CJDNS` anzubieten. Auch existiert aktuell noch keine Möglichkeit, schädliche Mitglieder zu blockieren oder zu erkennen. Das aktuelle Netzwerk ist also anfällig gegen potientielle Angriffe oder Sabotage. Trotz gewisser Mängel und Kritikpunkte ist das gesamte System im aktuellen Zustand aber funktionsfähig und nützlich.

- Orion:
  Natürlich sind bei einem Videospiel die Möglichkeiten nahezu unlimitiert, allerdings existieren auch hier einige interne, technische Probleme. Nebst der offensichtlichen Frage der Performance gibt es auch ein grundlegenderes Problem, welches die aktuelle Umsetzung davon abhalten würde, verbreiteter eingesetzt zu werden: Auch wenn ein *offizieller* Client existiert, so 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:
  Natürlich beschränkt sich der Umfang des Projekts nicht nur auf Videospiele oder Chat Programme. Eine Vielzahl anderer Anwendungen lässt sich mit der actaeon Bibliothek bauen. Dank einer offenen Lizenz gibt es ebenfalls die Hoffnung, dass andere Personen in der Zukunft Programme für das Ökosystem bauen.

5 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 Kommunikationsssytemen grosse Probleme für alltägliche Situationen und Menschen bringt, lassen sich die Wenigsten auf ein tiefgreifendes Gespräch über die Probleme der Zentralrouter ein. Doch wenn diese Debatte nur in den Büros der Megaunternehmen geführt wird, könnten sich die Fortschritte des Internets schnell in eine finstere Dystopie verwandeln. Nebst den offensichtlichen Problemen und Gefahren übermässiger Zentralisierung und der damit verbundenen Abhängigkeit darf nicht vergessen werden, dass kollaborative, dezentrale Systeme Unmengen von Vorteilen mit sich bringen. Seien es zensursichere Speicher- und Nachrichtensysteme oder die schiere Grösse von beispielsweise Videospielen, die ohne einen zentralen Knotenpunk möglich werden, existiert noch viel ungeschöpftes Potential.

Project Orion hat weder alle der genannten Probleme gelöst, noch das volle Potiential ausgenutzt und eine umfangreiche Alternative entwickelt. Stattdessen entstanden aus diesem Projekt 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 soll zusätzlich der Einstieg in die Welt der dezentralen Kommunikationssysteme einfacher gemacht werden. Da alle entstandenen Programme öffentlich zugänglich sind und unter freien Lizenzen weiterverbreiten werden können, existiert ebenfalls die Hoffnung, dass das Ökosystem in Zukunft durch weitere Applikationen erweitert wird. Mit insgestamt etwa 12000 Zeilen Code über 350 Commits sind mit diesem Projekt verschiedene, umfangreiche Programme entstanden, welche realen Nutzen bieten und vielseiting einsetzbar sind.

KANTI BADEN

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:	Abteilung/-en:
Dominik Keller, Jakob Klemm	G4a

Maturaarbeit:

Project Orion

Ort, Datum:

Baden, Schweiz, 7. 11. 2021

Unterschrift/-en:

Jakob Klemm

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

Kantonsschule Baden
5400 Baden
www.kanti-baden.ch

KANTON AARGAU