Actaeon

Jakob Klemm

Inhaltsverzeichnis

1 Einführung

Dieses Dokument soll ohne grosse Umschreibungen und ohne Dramatisierungen einen Einblick in die Konzepte und Entscheidungen und Systeme bieten, welche schlussendlich zur `actaeon` Bibliothek geführt haben. Es ist ebenfalls wichtig zu verstehen, dass dieses Dokument sich lediglich auf die, im Rahmen der Maturarbeit entstandenen Projekte beschränkt, es handelt sich hier nicht um eine komplette Einführung in dezentrale Datensysteme, allerdings existieren genügend Quellen für grundlegendere Informationen[1]. Für das schlussendliche Produkt wurde die `Rust`-Programmiersprache verwendet, welche ebenfalls nicht hier erklärt werden soll.

2 Projekte

Nachdem die zentralen Problemen definiert wurden, ist es an der Zeit, andere Projekte zu analysieren um herauszufinden, ob die Probleme bereits vollständig oder zumindest teilweise gelöst wurden.

Zwar wurden viele Probleme und Gefahren angesprochen, `Engine: Orin` soll sich allerdings nur auf die folgenden beiden Aspekte fokussieren:

- ein Nachrichtensystem zum Senden und Lenken der einzelnen Nachrichten.
- ein System zur Verarbeitung der einzelnen Nachrichten.

Dementsprechend sollen auch andere Projekte, die sich mit diesen beiden Aspekten analysiert werden. Um aber die Wahl der Projekte richtig zu verstehen, muss man die Vision hinter `Engine: Orion` im Blick behalten. Denn es wird schnell klar, dass es für 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. Was allerdings kein bekanntes Projekt umzusetzen versucht, ist nicht nur die grundlegende Neuentwicklung zentraler Systeme zur Behebung bekannter Probleme, sondern dazu noch die nahtlose Integration der einzelnen Komponenten für ein durchgehend integriertes, zusammenspielendes System.

[1] Orion Wiki - Distributed Systems, Jakob Klemm. `https://orion.jeykey.net/distributed_systems.pdf`

Da ein solches Projekt nicht öffentlich existiert, müssen stattdessen die einzelnen Probleme und deren Lösungsversuche angeschaut werden. Dafür wird grundlegend in zwei zentrale Komponenten unterschieden. Die Integration der beiden Teile soll hier nicht besprochen werden, da diese sich hauptsächlich in der Umsetzung zeigt.

2.1 Kademlia

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

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

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

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

> ein Peer-to-peer Nachrichten System basierend auf XOR-Metrik.[2]

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

Da die einzelnen Server nicht in der Lage sind, Informationen über das komplette Netzwerk zu speichern oder zu verarbeiten, funktionieren `Kademlia`-Systeme grundlegend anders. Anstelle der hierarchischen Anordnung der Router ist jedes Mitglied eines Systems gleichgestellt. dabei kümmert sich jedes Mitglied auch nicht um das komplette System, sondern nur um sein direktes Umfeld. Während dies für kleinere Systeme gut funktioniert und

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

vergleichbare Geschwindigkeiten liefert, skaliert es nicht so einfach für grosse Systeme. Genau dafür gibt es die XOR-Metrik.

2.1.1 Distanz

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

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

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

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

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

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

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

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

Um dies zu verstehen, hilft es, sich das System als umgekehrten Baum vorzustellen. Untergeordnet zum zentralen Punkt zu oberst stehen alle Mitglieder im System. Mit jeder weiteren Abzweigung zweier Teilbäume halbiert sich

die Anzahl. Man wählt am einfachsten zwei Abzweigungen pro Knoten, da sich damit die Werte direkt als Binärzahlen darstellen lassen, wobei jeder Knotenpunkt einfach eine Stelle in der langen Kette aus 0 oder 1 darstellt. Der ganze Baum sieht dann wie folgt aus[3]:

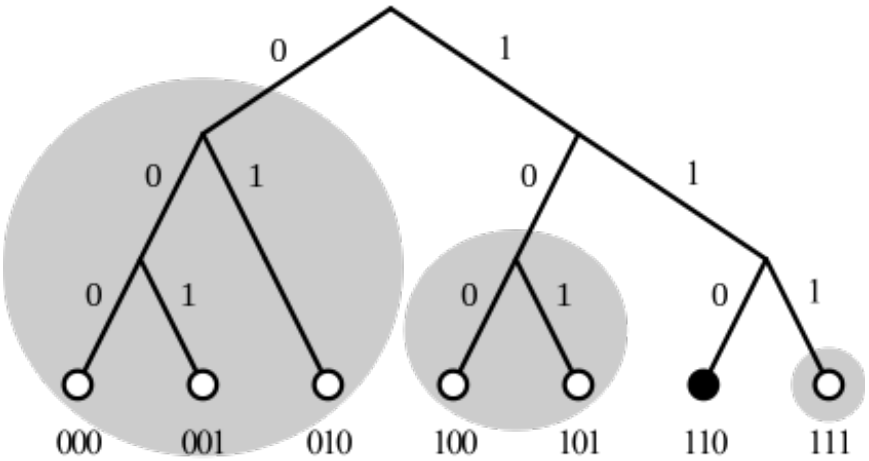

Abbildung 1: Beispielhafte Darstellung eines einfachen Kademlia-Systems

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

2.1.2 Routing-Table

In einem `Kademlia`-System hat jedes Mitglied eine gewisse Anzahl anderer Mitglieder, mit denen es sich verbunden hat. Da `Kademlia` ein sehr umfangreiches, kompliziertes Protokoll und System beschreibt, sollen hier nur einige zentrale Funktionen besprochen werden, die für diesen ersten Prototypen von `Engine: Orion` relevant sind. Besonders beim `Routing Table` lassen sich einige Abschnitte weglassen, welche zwar für die Optimierung und Skalierung eines Systems wichtig, allerdings für das Analysieren eines einfachen, kleinen Systems wie `Engine: Orion` irrelevant sind.

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

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

1. Sehr nahe an sich selbst (in der obigen Darstellung also: wenige Sprünge im Baum) werden alle Mitglieder gespeichert.

2. Je weiter weg sich die Mitglieder befinden, desto weniger werden gespeichert.

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

- In der obersten Ebene trennt sich der Baum in zwei vollständig getrennte Teile, was sich mit jeder weiteren Ebene wiederholt.

- Die einzige Möglichkeit vom einen *Ende* des Baums zum anderen zu kommen, ist über den obersten Knoten. Um also in die andere Hälfte zu kommen, braucht man mindestens eine Verbindungsstelle in der anderen Hälfte.

- Deshalb braucht ein `Routing-Table` nicht nur kurze Verbindungen zu nahen Punkten, sondern auch einige wenige Verbindungen zu Mitgliedern in der anderen Hälfte.

- Mit nur einer weit entfernten Adresse hat man eine Verbindung in *eigene* und die *andere* Hälfte. Hat man zwei solche Verbindungen auf die andere Seite hat man schon Verbindungen in jeden Viertel des Baumes.

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

Zwar hat ein vollständiges `Kademlia`-System noch komplexere Elemente, wie `k-Buckets`, welche den `Routing-Table` optimieren, allerdings sind

diese für die grundlegende Funktionsweise des Systems irrelevant.

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

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

2.2 BitTorrent

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

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

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

Auch wenn es noch nicht *vollständig* dezentralisiert ist, erlaubte es `Napster` Nutzern, Musik über ein automatisiertes System mit anderen Nutzern zu teilen und neue Titel direkt von den Geräten anderer Nutzer herunterzuladen. Dabei gab es allerdings immer noch einen zentralen Server, der die Titel sortierte und indizierte.
`Napster` musste am Ende abgeschaltet werden, nachdem die Klagen der Musikindustrie zu belastend wurden. Auch wenn das Produkt abgeschaltet

wurde, liess sich nichts mehr gegen die Idee unternehmen.

Über viele Iterationen und Generationen hinweg wurden die verteilten Systeme immer weiter verbessert, jegliche zentrale Server entfernt und in die Hände immer mehr Nutzer gebracht. Heute läuft ein Grossteil des Austauschs über `BitTorrent`.
`BitTorrent` baut auf der gleichen grundlegenden Idee wie `Napster` auf: Nutzer stellen ihren eigenen Katalog an Medien zur Verfügung und können Inhalte von allen anderen Mitgliedern im System herunterladen. Anders als `Napster` gibt es bei `BitTorrent` keine zentrale Komponente, stattdessen findet selbst das Indizieren und Finden von Inhalten dezentralisiert statt[4]. Dafür wird über das `Kademlia`-System aktiv bekannt gegeben, wer welche Inhalte zur Verfügung stellt, wobei einzelne Mitglieder speichern können, wer die gleichen Inhalte anbietet. Neben Dezentralisierung und Sicherheit lassen sich über `BitTorrent` tatsächlich gute Geschwindigkeiten erreichen, da sich Inhalte von mehreren Anbietern gleichzeitig herunterladen lassen. Da es sich bei `BitTorrent` eigentlich um ein grosses Dateisystem handelt, lassen sich direkt die `SHA1`-Hashwerte der Inhalte als `Kademlia`-Adressen verwenden.

2.3 Tox

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

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

ist Chat weiterhin eine zentrale Funktion, es wird aber auch Video- und Audiokommunikation sowie Filesharing gearbeitet.

Basierend auf der bekannten `NaCl`-Bibliothek[5] wird die gesamte Kommunikation über das `Tox Protocol` [6] zwingend End- zu Endverschlüsselt. Intern wird ein dezentrales Routing System basierend auf Kademila verwendet, mit welchem Kontakt zwischen Nutzern (Freunden) aufgebaut wird. Während im Kademila Whitepaper Addressen mit einer Länge von 20 Bytes definiert werden, nutzt `Tox` 32 Bytes. Dies vereinfacht die Verschlüsselung stark, da `NaCl` Schlüssel verwendet, welche ebenfalls 32 Bytes lang sind. Nebst der eingesparten Verhandlung von Schlüsseln und der zusätzlichen Kommunikation bindet diese Idee die Verschlüsselung direkt stärker in das Routing System ein, denn es werden keine zusätzlichen Informationen zum Verschlüsseln einer Nachricht gebraucht und sie kann direkt mit der Addresse des Ziels verschlüsselt werden.

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

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

- Auch muss man bedenken, dass nicht jedes Gerät im Internet in der Lage ist direkte Verbindungen mit jedem anderen Gerät aufzubauen. Besonders Firewalls können schnell zu Problemen führen. Um den Aufwand für die Nutzer zu minimieren wird `UDP hole punching` [7] verwendet, allerdings existieren auch dafür gewisse Kriterien und Probeleme.

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

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

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

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

3 Architektur

In diesem Kapitel sollen einige der zentralen Konzepte und Entscheidungen erläutert werden, welche schlussendlich zur Actaeon-Applikation geführt haben.

3.1 Verschlüsselung

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

Eigentlich beschreibt Kademlia Adressen mit einer Länge von *160 bits* oder *20 bytes*, allerdings ist es ohne grosse Probleme möglich, die Adressen Länge auf *32 bytes* zu verlängern. Dies erlaubt es uns, die Verschlüsselung und das Adresssystem direkt miteinander zu verbinden. Das mag auf den ersten Blick etwas umständlich und unlogisch wirken, es erlaubt allerdings, ohne weitere Operationen verschlüsselte Nachrichten an einen Nutzer zu schicken, wobei

lediglich dessen Adresse bekannt sein muss. Insgesamt fällt damit viel Komplexität weg und macht das erstellen, verifizieren und finden von Adressen viel einfacher.

3.2 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, bedeutet sie folgen einem gewissen Schlüssel und erhalten Nachrichten von diesem.
- Jeder Nutzer kann dann in den Themen denen 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 Emailsystem, ein komplexer Datenverarbeitungsmechanisums oder ein Datennetzwerk, alle lassen sich relativ einfach mit diesen beiden Funktionen modellieren.

3.2.1 Dezentraler PubSub

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

- Themen lassen sich vereinfacht als Einträge in einer Datenbank beschreiben. Die Identifikation der Themen ist dabei der Schlüssel, wobei die Abonnenten als dazugehörige Felder ausgedrückt werden können. Oben wurde allerdings bereits ein System beschrieben, welches zuverlässig dezentral Daten speichern kann. Wenn man in der beschriebenen `Kademlia` Implementierung die Checksumme des Inhalts mit der Checksumme des Themesschlüssels ersetzt, lässt sich `Kademlia` ohne weitere Veränderungen für einen dezentralen `PubSub` einsetzen.
- Danach bleibt natürlich noch das Problem der Nachrichtenverbreitung. Dafür gibt es allerdings verschiedene Möglichkeiten:

  - Die Nachrichten werden direkt an das Zuständige Mitglied gesendet, von dort werden sie weitergeleitet. Vorteilhaft an diesem Konzept ist natürlich, dass die Verwender des Systems unglaublich einfach gehalten werden können. Sie müssen lediglich Nachrichten an eine Adresse schicken, das System kümmert sich dann von alleine um die Weiterverbreitung. Damit geben die Nutzer allerdings auch 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.
  - Gegen gesetzt dazu könnten auch die Aktionen des Abonnierens und Deabonnierens als Nachrichten im System verbreitet werden. Jeder abonnierte Nutzer wird somit also über neue Abonnenten informiert und speichert deren Details lokal. Zwar erhöht dies die Komplexität enorm, erlaubt aber schnellere Übertragung und genauer Kontrolle über die Auswahl der Abonnenten.

3.2.2 Probleme

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

- Wie bereits eben angesprochen kann es zu gewissen Unklarheiten und Problemen im Zusammenhang mit der Verschlüsselung der Nachrichten in dezentralen Systemen kommen. Da Nachrichten meist über das offene Internet übertragen werden und daher Verschlüsselung nahezu zwingend benötigt wird, muss man sie auch hier berücksichtigen. Wie bereits im Abschnitt zur Verschlüsselung angesprochen, sollen Nachrichten mit dem öffentlichen Schlüssel, welcher auch gleichzeitig die Adresse darstellt, des Empfängers verschlüsselt werden. Beim durchgehen der oben beschriebenen Architekturen wird ein Problem offensichtlich: Wenn ein Thema ein normaler Empfänger im System ist, muss seine öffentliche Adresse verwendet werden. Allerdings wurde definiert, dass sich 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 es natürlich nicht möglich, ein geheimer Schlüssel lässt sich nicht aus nur dem öffentlichen errechnen. Hier wird also eigentlich ein System verlangt, bei welchem der private Schlüssel durch eine Checksumme errechnet wird, wobei der daraus entstehende öffentliche Schlüs-

sel als Adresse verwendet wird. Gleichzeitig darf der private Schlüssel nicht bekannt sein, sonst wäre die gesamte Verschlüsselung sinnlos. Es wird schnell offensichtlich, dass solche Bedingungen nie erfüllt werden können.

- Da es sich bei der Liste der Abonnenten um Daten handelt, welche während der Laufzeit gespeichert und verwaltet werden müssen, bringt man plötzlich eine Vielzahl neuer Probleme ins Spiel. So müssen die Daten repliziert und gesichert werden, da ein einzelnes Mitglied jeder Zeit unerreichbar sein könnte, sie müssen verifiziert werden, da man kein Vertrauen in die Mitglieder des Systems haben darf und sie müssen trotz häufiger Änderungen konstant gehalten werden. Das Gebiet der dezentralen oder verteilten Datenbanken alleine ist sehr gross und komplex, wenn man also plötzlich nebst einem Nachrichtensystem ohne verteilte Zustände auch noch eine verteilte Datenbank verwalten muss, übersteigt dies meist die erhoffte Komplexität vieler Projekte.

4 Actaeon

TODO: Actaeon