# Micropayments:
## Das schnelle und günstige Internet der Werte

Thorsten Knoll

thorsten.b.knoll@student.hs-rm.de

28. März 2018

# Inhalt

# 1. Einleitung

## 1.1. Bitcoin-Eigenschaften

Im Jahr 2008 wurde das Whitepaper „Bitcoin: A Peer-to-Peer Electronic Cash System“ [Nak08] unter dem Pseudonym Satoshi Nakamoto auf einer Kryptographie-Mailingsliste vorgestellt. Kurze Zeit später in 2009 hat Satoshi Nakamoto dazu die erste Referenzimplementierung, bekannt als Bitcoin-Client 0.1.0, veröffentlicht [Nak09]. Seitdem erfahren kryptographische Währungen (Kryptowährungen) ein ähnlich exponentiell wachsendes Interesse, wie das Internet oder die Email-Kommunikation in Ihrer Entstehungsphase. Zum Zeitpunkt der Erstellung dieses Buchs sind schon über 1.500 verschiedene Kryptowährungen bekannt und die Anzahl wächst weiter. Täglich werden neue Konzepte und Anwendungen der zugrunde liegenden Blockchain-Technologie veröffentlicht. Es hat eine breite Adaption der Technologie stattgefunden und man kann durchaus behaupten, Kryptowährungen sind im „Mainstream“ angekommen. Bitcoin, als größte und bekannteste Kryptowährung, gilt heutzutage als sicher, in den Eigenschaften mit denen es erstellt wurde. Diese Eigenschaften [Ant15] sind in Abbildung 1.1 dargestellt und im Folgenden (sehr) kurz definiert.

**Definition 1.1.1: Dezentral**

Keine Instanz des Netzwerks hat eine gesonderte, übergeordnete Rolle. Es gibt keine zentrale Instanz. Alle Teilnehmer sind gleichberechtigt und geographisch unabhängig. Es ergibt sich eine verteilte, hierarchielose Netzwerksturktur, die Dezentralität. Die Koppelung der Netzwerkknoten ist lose.

**Definition 1.1.2: Zugangsberechtigungslos (permissionsless)**

Es bestehen keine Zugangsberechtigungen zum Netzwerk. Anders als für Bankkonten ist keine Identifikation notwendig. Ein selbsterzeugtes, kryptographisches Schlüsselpärchen und optional die frei erhältliche Open-Source-Software sind für den Zugang und die Benutzung ausreichend.

**Definition 1.1.3: Vertrauenslos (trustless)**

Die Regeln des Netzwerks sind mit der Open-Source-Software bei Erstellung und Veröffentlichung bekannt und nur durch Mehrheitsentscheid der Benutzer änderbar (freiwilliger Umstieg auf neue Versionen). Es muss keiner zentralen Stelle oder den anderen Benutzern vertraut werden. Regelkonformes Verhalten wird vom Netzwerkprotokoll belohnt und schadhaft agierende Teilnehmer tragen automatisch selbst den Schaden.

**Definition 1.1.4: Unveränderbar (immutable)**

In Kryptowährungen werden Buchhaltungsbücher (Ledger) in Form von verteilten, öffentlichen Datenstrukturen geführt. In Bitcoin ist diese Datenstruktur eine durch Hashpointer verkettete Liste von Datenblöcken (genannt Blockchain). Die Hashpointer ergeben im Zusammenspiel mit dem Proof-of-Work Konsens-Mechanismus eine, mit derzeitigen Computerkapazitäten unveränderliche Aufzeichnung der Transaktionshistorie.

**Definition 1.1.5: Zensurresistent (censorship resistant)**

Keine einzelne oder gruppierte Instanz des Netzwerks kann Zensur ausüben. Die festgelegten Regeln können nicht von einer solchen Instanz geändert werden. Bitcoin ist somit nicht regulierbar und es kann keine Kontrolle von außen aufgefügt werden.

Manche der erwähnten, über 1500 weiteren Kryptowährungen folgen auch diesen Eigenschaften und sind daher als ähnlich sicher zu betrachten (z.Bsp: Ethereum, Litecoin, Monero). Einige Währungen haben leicht geänderte Eigenschaften und Grundkonzepte und müssen daher im Laufe der kommenden Jahre ihre Stabilität und Sicherheit erst noch unter Beweis stellen (z.Bsp: IOTA, Stellar). Und die wahrscheinliche Mehrzahl der Kryptowährungen wurde mit nur dem Ziel der persönlichen Bereicherung erstellt. Die Veröffentlichung von Bitcoin als OpenSource-Software ist eine Notwendigkeit, um die oben beschriebenen Eigenschaften zu erreichen. Aber auf der anderen Seite kann diese OpenSource-Software auch in simplen Copy-Paste Verfahren mit geringsten Veränderungen (z.Bsp ein finanizeller Bias zu eigenen Gunsten) für neue Kryptowährungen verwendet werden und wird auch manigfaltig dafür verwendet. Eine Hauptaufgabe in der Strukturierung von Informationen über Kryptowährungen ist es, die „Spreu vom Weizen“ zu trennen und dabei nicht deren, teilweise bodenlosen und nicht haltbaren Werbeversprechen zu verfallen. Ein ganz eigenes, aber hier nicht betrachtetes Kapitel der Kryptowährungen sind Initial-Coin-Offerings (ICOs), welche Börsengängen in der bisherigen Unternehmes- und Finanzwelt ähnlich sind.

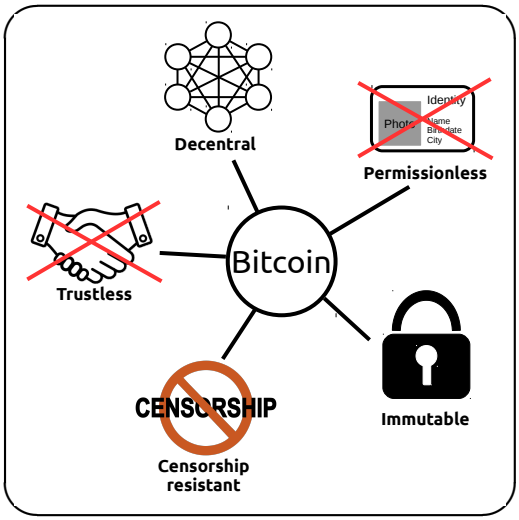


Abb. 1.1.: Bitcoin: Eigenschaften

## 1.2. Alltagswährung Bitcoin?

Es stellt sich die Frage, ob aus den besprochenen (Bitcoin-) Eigenschaften auch automatisch die mögliche Verwendung von Kryptowährungen als Zahlungsmittel für den alltäglichen Gebrauch folgt? Ist es möglich, beim Bäcker, im Supermarkt oder im Online-Buchhandel mit Kryptowährungen zu bezahlen? Und welche weiteren Anwendungen und Anforderungen warten in der Zukunft der Kryptowährungen? Antworten auf diese Fragen sind nicht immer definitiv oder eindeutig zu finden, aber Bewertungs-Ansätze und aktuelle Entwicklungen dazu beschreibt dieses Buch.

Im Folgenden betrachten wir die beiden finanziell größten Kryptowährungen Bitcoin und Ethereum, und besprechen anhand dieser die Begriffe Blockzeit, Transaktionszeit und -gebühr mit aktuellen Zahlen und Graphen. Mit diesen Grundlagen gewappnet, beschäftigen wir uns dann in Kapitel 2 mit Micropayments. Anwendungsbeispiele verdeutlichen hierbei die Funktionsweise und die Notwendigkeit von Micropayments in Kryptowährungen. Eine Übersicht der aktuell erfolgversprechenden Projekte und Entwicklungen im Bereich Micropayments stellt Kapitel 3 dar. Das Szenario einer Nachbarschaftshilfe wurde als Beispiel für Micropayments ausgearbeitet und mit Raspberry Pi Computern als Demonstrator aufgebaut. Die Entwicklung dieses Demonstrators und dessen Software-Implementierung sind in Kapitel 4 beschrieben. Das Buch endet in Kapitel 5 mit einer Zusammenfassung in Form eines Fazits.

**Aktualität dieses Buchs:**
Zum Zeitpunkt der Erstellung im Frühjahr 2018 sind die dargestellten Konzepte, Entwicklungen und Zahlen aktuell. Die rasante Entwicklung in diesen Bereichen lässt aber vermuten, das diese Aktualität nur kurze Zeit halten wird. Daher versteht sich dieses Buch als eine Art „Snapshot“ und die enthaltenen Informationen könnten in 1-2 Jahren schon längst wieder überholt sein. Viele Quellenreferenzen zeigen auf Internetseiten, auch für diese Informationen ist die Haltbarkeit begrenzt. Sollte eine Quelle nicht mehr erreichbar sein, hilft aber mit Sicherheit eine einfache Stichwortsuche nach dem Titel des Dokuments weiter. Somit aber genug der Warnungen; ab in das Abenteuer Micropayments.

## 1.3. Aktueller Stand, 2017-2018

### 1.3.1. Definitionen

Für die Analyse der aktuellen Situation in Bitcoin und Ethereum sind einige, wenige Begriffsdefinitionen notwendig. Die Definitionen sind möglichst allgemein gehalten, um Unterschiede in einzelnen Kryptowährungen auszugleichen. Trotzdem kann mit diesen Definitionen nicht das gesamte Universum aller Kryptowährungen abgedeckt werden. So macht die Definition einer Blockzeit wenig Sinn für Kryptowährungen, in denen es gar keine Blockchain und damit auch keine Blocks gibt (z.Bsp. IOTA).

**Definition 1.3.1: Transaktionsgebühr**

Transaktionen (Tx) sind elementare Einheiten in Blockchains und vergleichbar mit Banküberweisungen. Für Transaktionen wird eine variable Gebühr an das Blockchain-Netzwerk (z.Bsp. Bitcoin) entrichtet, **die Transaktionsgebühr**. Diese ist vergleichbar einer Banküberweisungsgebühr und wird in der Einheit der Kryptowährung pro Transaktion angegeben (z.Bsp. BTC/Tx für Bitcoin), aber zum Vergleich untereinander auch gerne in Dollar umgerechnet (USD/Tx).

**Definition 1.3.2: Blockzeit**

Mehrere Transaktionen werden zu einem Block zusammengefasst. Blöcke werden nach einem Konsens-Protokoll (z.Bsp. Proof-of-Work in Bitcoin) an die Blockchain angehängt und sind nach kurzer Zeit unveränderlich. Die Zeit zwischen dem Anhängen zweier Blöcke ist **die Blockzeit**. Die Einheit für Blockzeit ist Min/Block.

**Definition 1.3.3: Mittlere Transaktionszeit**

Das Produkt aus Blockzeit und Anzahl der im Block enthaltenen Transaktionen (Tx), gemittelt über einen festen Zeitraum ist **die Transaktionszeit**. Im Folgenden ist dieser Zeitraum zur Ermittlung des Mittelwerts auf einen Tag festgelegt. Die Einheit ist Tx/Sek.

### 1.3.2. Bitcoin

Mit der Fragestellung, ob Bitcoin oder Ethereum als alltägliches Zahlungsmittel geeignet sind, betrachten wir im Folgenden die Entwicklung der vorher definierten Kenngrößen über einen Zeitraum von März 2017 bis Februar 2018. Abbildung 1.2 zeigt die Bitcoin-Transaktionsgebühren in USD. Während der erste Jahreshälfte in 2017 sind Ausschläge bis zu ca. 8 USD/Tx zu erkennen. In der zweiten Jahreshälfte steigern sich diese bis zum Maximum von 55 USD/Tx im Dezember und fallen bis Ende Februar wieder auf ähnliche Werte wie zu Beginn des Darstellungszeitraums.

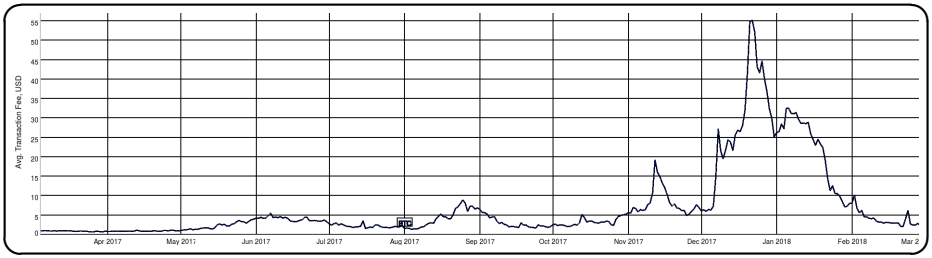


Abb. 1.2.: Bitcoin: Transaktionsgebühren in 2017-2018

Bildquelle: `https://bitinfocharts.com`

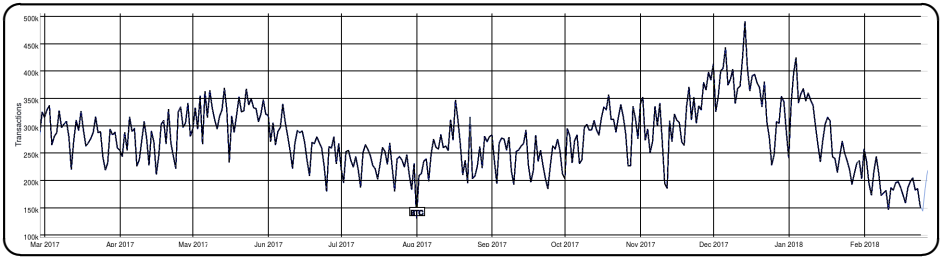


Abb. 1.3.: Bitcoin: Tägliche Transaktionen in 2017-2018

Bildquelle: `https://bitinfocharts.com`

Abbildung 1.3 zeigt die Zugehörige Anzahl von Transaktionen pro Tag über den selben Zeitraum. Die Amplitude reicht hier von ca. 200.000 - 500.000 Transaktionen pro Tag. Umgerechnet auf Tx/Sek ist das Minium dann ca.

2,3 Tx/Sek und das Maximum ca. 5,8 Tx/Sek. Eine leicht zeitlich versetzte, positive Korrelation zu den Transaktionsgebühren ist erkennbar.

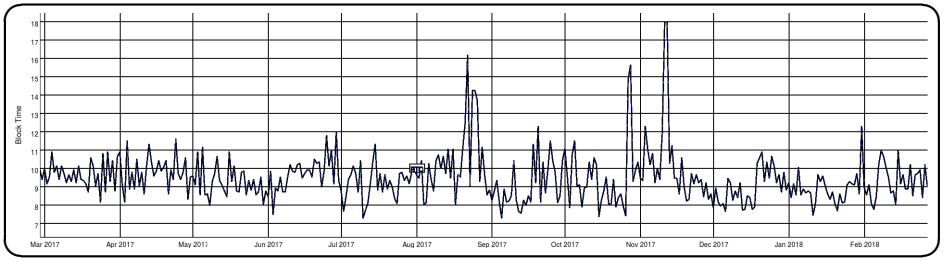


Abb. 1.4.: Bitcoin: Blockzeiten in 2017-2018

Bildquelle: `https://bitinfocharts.com`

Die Blockzeiten in Abbildung 1.4 sind bis auf 3 Maximal-Ausschläge im August, November und Dezember durchgängig um die 10 Minuten-Linie verteilt. Bitcoin hat den, im Protokoll verankerten Mechanismus der Schwierigkeits-Adjustierung, um die Blockzeit möglichst konstant auf 10 Min/Block zu halten. Da dieser Mechanismus nur alle 2 Wochen eine Adjustierung vornimmt, sind einzelne, aber kurze Ausschläge möglich und regulieren sich in einem Zeitfenster von maximal 2 Wochen zurück auf 10 Min/Block.

**Alltagstauglich?:** Die Blockzeit von ca. 10 Min/Block ist eine beständige Regulierung im Bitcoin-Netzwerk. Sie wird machmal auch als der „Bitcoin-Herzschlag“ bezeichnet. Da die Größe eines einzelnen Blocks beschränkt ist, kann auch nur eine maximale Anzahl an Transaktionen pro Block aufgenommen werden. In der transitiven Folgerung bedeutet dies, dass auch die maximale Anzahl von Transaktionen im Bitcoin-Netzwerk eine Obergrenze hat. Es scheint, dass diese Obergrenze in der Praxis bei dem tages-gemittelten Maximum von ca. 5,8 Tx/Sek liegt. Hierüber gibt es verschiedene Berechnungsvarianten, wovon sich aber keine um wesentliche Größenordenungen von den hier dargestellten Werten unterscheidet. Anders sieht die Betrachtung der Transaktionsgebühren aus. Diese Gebühren sind variabel vom Anwender für jede einzelne Transaktion anzugeben. Steigt der Bedarf nach Transaktionen (viele Anwender möchten „überweisen“), startet damit eine Art Wettbewerb um die Ausführung der Transaktionen. Wenig Gebühr führt zu eventuell verzögerter Ausführung der Transaktion und hohe Gebühr zu möglichst schnellerer Aus-

führung. Das Maximum von bis zu 55 USD/Tx in Abbildung 1.2 dürfte auf dieses Verhalten zurückzuführen sein. Eine Nutzung in alltäglichen Bezahlvorgängen ist mit Spitzenwerten in den gennanten Größen nicht möglich. Einen Kaffee To-Go inklusive Gebühren im mehrfachen Wert des Kaffees in Bitcoin zu bezahlen, ist nicht akzeptabel. Auch die Transaktionszeit von maximal 5,8 Tx/Sek ist für den Anspruch eines weltweit funktionierenden Systems nicht für alltägliche Bezahlvorgänge ausreichend. Vermutlich kaufen alleine in der Frankfurter Innenstadt an einem Werktag morgens mehr als 6 Leute pro Sekunde einen Kaffee To-Go. Die dargestellten Zahlen deuten also darauf hin, dass eine Lösung für dieses Problematik entwickelt werden muss. Hierzu betrachten wir dann ab Kapitel 2 das Konzept der Micropayments. Aber vorher werfen wir noch einen Blick auf die Situation in Ethereum.

### 1.3.3. Ethereum

Die Betrachtung der Transaktionsgebühren in Ethereum über den gleichen Zeitraum (März 2017 bis Februar 2018) sieht der Entwicklung in Bitcoin sehr ähnlich. Die Gründe dafür sind allerdings ganz Andere. Bis in den Dezember hält sich Ethereum, abgesehen von einzelnen, kleinen Ausschlägen, unter der 0,5 USD/Tx Marke. Erst im Dezember und Januar sind schnell auftretende, heftige Anstiege der Transaktionsgebühren zu bemerken, welche dann allerdings auch mehrere Wochen anhalten. Das Dezember-Maximum liegt bei ca. 1,5 USD/Tx und das Januar-Maximum bei ca. 4 USD/Tx. Die Gründe für diese hohen Gebühren sind unter den Stichpunkten „Cryptokitties“ (Dezember) und „Geth Berechnungsfehler“ (Januar) weiter unten beschrieben.

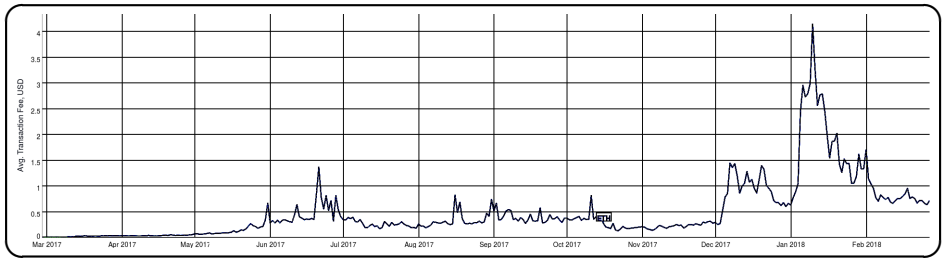


Abb. 1.5.: Ethereum: Transaktionsgebühren in 2017-2018

Bildquelle: `https://bitinfocharts.com`

Die Transaktions- und Blockzeiten in Ethereum sind in dem dargestellten Zeitraum über mehr Faktoren als in Bitcoin gekoppelt. Das Ethereum-Netzwerk hat am 16.10.2017 das sogenannte „Ice Age“ mittels eines HardForks gestartet. Ausführlichere Informationen über diesen langfristigen Update-Prozess sind in [Mad18] und [Tea17] nachzulesen. In Abbildung 1.6 ist ein treppenförmiger Anstieg der Blockzeiten bis zum Maximum von 0,5 Min/Block am 16.10.2017 zu erkennen. Danach fallen die Blockzeiten aufgrund des Updates wieder auf Werte um die 0,25 Min/Block zurück. Da das Ethereum-Netzwerk bis zum 16.10.2017 nicht an seiner Kapzitätsgrenze für maximale mögliche Transaktions-Anzahl ausgelastet war, hat sich der Treppeneffekt auch erstmal nicht in den Transaktionszeiten (Abbildung 1.7) bemerkbar gemacht.

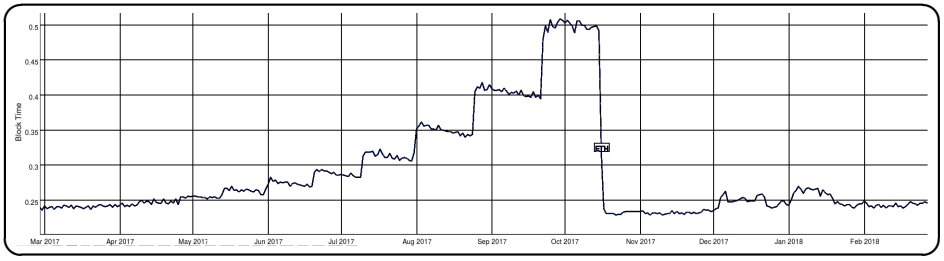


Abb. 1.6.: Ethereum: Blockzeiten in 2017-2018

Bildquelle: `https://bitinfocharts.com`

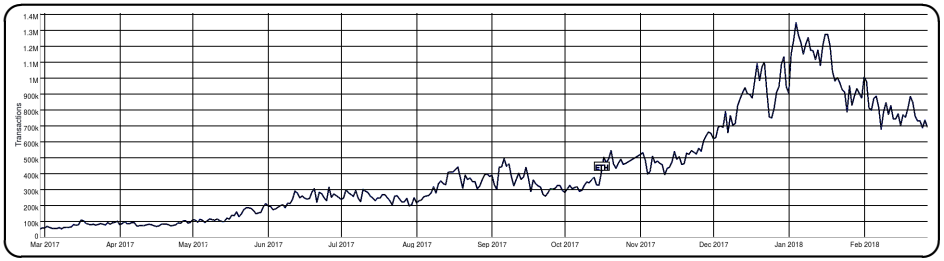


Abb. 1.7.: Ethereum: Tägliche Transaktionen in 2017-2018

Bildquelle: `https://bitinfocharts.com`

Die Transaktionszeiten sind bis in den Januar 2018 kontinuerlich gestiegen. Im Januar wurden dann Höchstwerte von ca 1,35 Millionen Tx/Tag erreicht. Das

entspricht einem Wert von ca. 15,6 Tx/Sek. Dieser Wert liegt allerdings sehr wohl an der derzeit maximal möglichen Grenze der Ethereum-Transaktionzeit und spielt daher auch in der folgenden Betrachtung des Januar-Ereignisses „Geth Berechnungsfehler“ eine entscheidend Rolle.

**Cryptokitties:** Der spontane Anstieg der Transaktionsgebühren in Dezember 2017 ist auf das Spiel „Cryptokitties“ zurückzuführen [Küh17]. In diesem Spiel sind Katzen als virtuelle Sammel-Gegenstände in der Ethereum-Blockchain festgehalten. Aus zwei Cryptokitties kann ein Drittes als Kind generiert werden. Ausserdem können die Kitties natürlich unter den Spielern gehandelt werden. Während solche Spielprinzipien schon lange bekannt sind (Collectable Cardgames), war die Abbildung auf einer Blockchain ein Novum und hat weltweites Interesse angezogen. Die Preise der seltensten Cryptokitties sind in unrationale Größenordnungen geschossen. Das Ergebnis für Ethereum war eine, über Wochen hinweg völligst „verstopfte“ Blockchain. Jeder Spiel- und Handelsvorgang ist nämlich eine Transaktion in der Ethereum-Blockchain, und davon gab es im Dezember wirklich viele.

**Geth Berechnungsfehler:** Im Januar 2018 hat sich dann aber ein weiteres, noch sehr viel höheres Maximum für die Transaktionsgebühren in Ethereum ergeben. Und diesmal waren wohl keine Katzenbilder beteiligt, sondern ein Algorithmus im Ethereum-Client Geth. Den Benutzern des Geth-Clients wird für neu erstellte Transaktionen eine Gebühr vorgeschlagen. Die Berechnung dieses Vorschlags basierte bis dahin auf dem Median der vergangenen Transaktionsgebühren. Und dabei hat sich, durch eine Reihe von Transaktionen mit sehr hoch eingestellten Gebühren, der Median ständig weiter nach oben verschoben. Zusätzlich ist dieser Fehler wohl erst wirksam aufgetreten, als das Ethereum-Netzwerk an seine Transaktions-Kapazitätsgrenze kam. Die Vorschlags-Berechnung wurde daraufhin umgestellt, und die Gebühren haben sich ordnungsgemäß nach unten angepasst. In einem Interview mit Nick Johnson (Ethereum Foundation) ist dieser Vorgang ausführlicher beschrieben [Dan18].

**Alltagstauglich?** Auch für die Ethereum-Blockchain ergibt sich das Bild der Nicht-Verwendbarkeit für alltägliche Bezahlvorgänge. Zwar ist die Transaktionszeit um den Faktor 2-3 größer als in Bitcoin, aber bei weitem immernoch

nicht ausreichend. Die Transaktionsgebühren sind wesentlich niedriger als in Bitcoin (meist >0,5 USD/Tx), aber auch das ist noch weit zu unterbieten. Die spontanen Maxima sind ein weiterer Grund für die Nicht-Alltagtauglichkeit. Wenn der Bezahlvorgang eines Kaffee To-Go in diesem Maße mal schneller, langsamer, teurer oder billiger wird, möchte wohl kaum jemand dieses System dafür nutzen. Sehen wir uns also im Folgenden an, was Micropayments zur Situation beitragen können und welche Entwicklungen es in diesem Bereich gibt.

# 2. Micropayments

## 2.1. Was sind Micropayments?

Eine lockere Übersetzung von Micropayments wäre „kleine Zahlungsvorgänge“. In Betrachtung der vorausgehenden Analyse der aktuellen Zahlen für Transaktionsgeschwindigkeiten und -kosten kann dies um die Eigenschaften „günstig“ und „schnell“ erweitert werden. Wir sprechen also von Micropayments als schnelle, günstige und kleine Zahlungstransaktionen. Gibt es für die einzelnen Eigenschaften objektive, vielleicht sogar absolute Werte?

**Geschindigkeit und Gebühren:**
Internationaler Zahlungsverkehr kann, in einem Weg über nationale Banken, bis zu mehreren Werktagen dauern. Mit spezialisierten Dienstleistern (z.Bsp. MoneyGram, WesternUnion) geht das zwar schneller, die Gebühren steigen dafür aber auch gerne in zweistellige Eurobeträge. Verglichen mit diesen Geldtransfermethoden, sind Ethereum und Bitcoin rasant und günstig. Im Bargeldverkehr ist eine Transaktion zum Kauf eines Kaffees erstens kostenlos (Münzen übergeben) und sehr schnell (<30 Sekunden). Verglichen mit diesem Bezahlvorgang, sind Ethereum und Bitcoin langsam und teuer. Abbildung 2.1 stellt die Situation von Anwendung und Betrachter in einen graphischen Zusammenhang.

**Beträge:**
Die herkömmliche Sichtweise auf kleine Zahlungsbeträge hat meistens ihre Untergrenze in der Granularität der verwendeten Währung. Es gibt wenige Zahlungen im Euroraum, die unterhalb eines Eurocents liegen. Wir sind es im Allgemeinen nicht gewohnt, über kleinste Beträge unterhalb der Darstellungsgrenze unserer staatlichen Währung nachzudenken. Solche Beträge begegnen uns (bisher) im Alltag nicht. In Kryptowährungen sind Transaktionen

solch kleiner, oder sogar winzigster Beträge aber prinzipell möglich. In den nachfolgenden Beispielen betrachten wir Anwendungen, für die solch winzige Betragsgrößen sogar unverzichtbar sind.

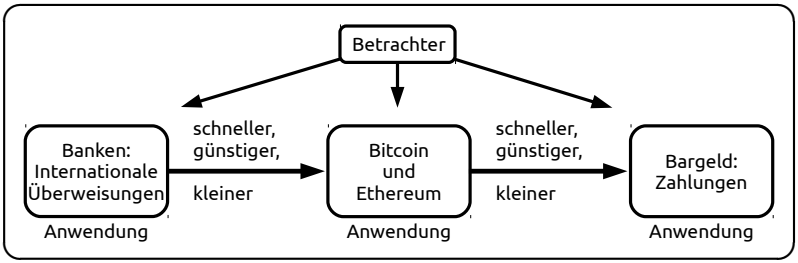


Abb. 2.1.: Micropayment: Betrachter und Anwendungen

**Objektiv und absolut?**
Wie wir feststellen, ist es schwer oder vielleicht sogar gar nicht möglich, absolute Werte für klein, schnell und günstig zu finden. Auch die gewünschte Objektivität scheint nicht erreichbar. Die Definition dieser Eigenschaften hängt also stark vom Verwendungszweck, der Anwendung und dem Beobachter ab. Daher betrachten wir im Folgenden einige Anwendungsbeispiele für Micropayments und deren Anforderungen an Geschwindigkeit und Kosten.

## 2.2. Energie

Elektromobilität und Energieverteilungsnetze stehen in starkem Zusammenhang. Die flächendeckende Adaption von Elektromobilität setzt ein gewisses Maß an dezentraler Verteilung der Energie voraus. Elektrische Fahrzeuge werden wahrscheinlich nicht mit dem bisherigen Konzept von Benzin- und Dieseltankstellen kompatibel sein. Wesentlich mehr Ladestationen an vielen unterschiedlichen Stellen werden benötigt. Der Tagesablauf eines solchen Fahrzeugs könnte 10-20 Ladevorgänge beinhalten. Alleine die Aufladungen an festen Aufladestationen (Zuhause, Arbeit, Supermarkt) würde die erwähnten 10-20 Ladevorgänge am Tag verursachen. In einem nächsten Schritt könnte die Aufladetechnik aber auch über die gesamte Strassen-Infrastruktur verteilt werden. Mit Induktionsschleifen an Ampeln und in Parkplätzen würde diese Anzahl

schnell hunderte Ladevorgänge am Tag verursachen. Und eventuell wäre in diesem Konstrukt keine zentrale Abrechnungsstelle enthalten. Die Abrechnung über ein sekundengenaues, schnelles Bezahlsystem mit Transaktionsgebühren im Millicent-Bereich wäre von Nöten. Eventuell müssten diese Abrechnungen mit unterschiedlichen Elektrizitäts-Anbietern dezentral als Peer-to-Peer-System erfolgen.

## 2.3. Videostreams

Am Beispiel von Videostreams lässt sich das Konzept der direkten, zeitgesteuerten Bezahlung verdeutlichen. Videostreams müssten nicht mehr komplett gekauft oder gemietet werden, sondern könnten in kleinen Zeiteinheiten genau abgerechnet werden. In jeder Sekunde, in der ein Stream läuft, wird eine kleine Währungseinheit an den Ersteller oder Vertreiber des Inhalts gezahlt. In diesem Modell müssten für jeden Videostream im Sekundentakt winzige Beträge bezahlt werden. Bei einem angenommenen Preis von 5 Euro für einen 90 minütigen Spielfilm wären das ca 0,001 Euro pro Sekunde. Natürlich müsste die Transaktionsgebühr um weitere Magnituden geringer als dieser Preis sein. Im besten Fall werden gar keine Transaktionsgebühren berechnet.

## 2.4. Computerspiele

Die Monetarisierungskonzepte von Computerspielen haben sich in den letzten Jahren stark verändert. Früher galt der Erwerb eines Datenträgers (CD, DVD) zusammen mit einem Schlüssel (gedruckt, in der Verpackung) als Beweis des Eigentums über eine Kopie des Spiels. Mittlerweile werden die meisten Spiele als digitale, virtuelle Güter vertrieben und die Eigentumsverhältnisse sind damit wesentlich komplizierter geworden. In vielen Spieleplattformen erwirbt der Käufer nur die Nutzungsrechte am Spiel, nicht mehr eine persönliche Kopie. Zusätzlich haben sich durch In-App-Käufe, In-Game-Gegenstände und Pay-to-Win-Systeme virtuelle, ökonomische Systeme in Computerspielen entwickelt. Eine eigene Sparte der Blockchain-Entwicklungen möchte diese Ökonomien wieder in die Kontrolle und das Eigentum der Spieler zurück bringen. Eine zentralisierte, geschlossene Ökomonie (z.Bsp. das Steam-Netzwerk von Valve) kann solche Micropayments natürlich sehr gut abbilden. Eine Abbildung auf öffentliche, dezentrale und globale Blockchain-Infrastruktur hätte extreme An-

forderungen an Geschindigkeit und Kosten. Das oben genannte Spiel „Cryptokitties“ ist nur ein kleiner Vorgeschmack auf die kommenden Entwicklungen. Alleine die Abbildung jedes Spielzugs, jeder Spielaktion und jedes Spielgegenstands in dem sehr bekannten „League of Legends“ würde die Dimensionen von „Cryptokitties“ um mehrere Magnituden verschieben. Eine aktuelle Entwicklung in diesem Bereich stellt zum Beispiel das „GameCredits“-Projekt dar [1].

## 2.5. Internet der Dinge

Das Internet der Dinge bietet viele Möglichkeiten und Ansätze für Micropaymentbeispiele. Ein mögliches Szenario sind Sensornetzwerke rund um den Globus. Diese weltweiten Sensoren liefern Echtzeitdaten aus physikalischen Messungen. Das könnten zum Beispiel Wetterdaten, geologische Daten oder Luftmessungen sein. Weiterhin müsste dieses Netzwerk nicht zwingend einem einzigen Anbieter gehören. Unter der Annahme eines dezentralen, Micropaymentbasierten Abrechnungssystems könnten Privatpersonen rund um die Welt ihre eigenen Sensoren ins Netzwerk anbinden und die Daten an interessierte Firmen, Institutionen oder Regierungen verkaufen. Auch hierfür würden winzige Beträge in einem konstanten, dezentralen Abrechnungsfluss verschoben.

## 2.6. Alltagsgebrauch

Wie schon in Kapitel 1 besprochen, sind selbst die Bezahlvorgänge im Alltag momentan nicht mehr mit Kryptowährungen wie Bitcoin oder Ethereum möglich. Wenn Micropaymentsysteme entwickelt werden, könnte diese Funktionalität als Mindestmaß der Tauglichkeit herangezogen werden. Ein Kaffee oder ein Sandwich sollte zu geringsten, oder gar keinen Gebühren und mit dem gleichen Zeitaufwand einer Bargeldbezahlung in einem Micropaymentsystem zu kaufen sein.

[1] `https://gamecredits.com/`

## 2.7. Und jetzt alles zusammen

Aus den bisherigen Beispielen wird deutlich, dass es eine breite Palette der möglichen Anwendungen für Micropaymentsysteme gibt und viele weitere hinzukommen werden. Im bisherigen Internet ist das Thema Bandbreite und wachsende Contentnachfrage ein „Hase und Igel Rennen“. Wer hätte sich vor der Entwicklung von Smartphones gedacht, dass Livestreaming auf Telefone über mobiles Internet möglich sein wird? Die Bandbreite wächst beständig, die Anforderungen an die Bandbreite wachsen genauso beständig, und das in einer bisher nicht aufhörenden Schleife. Nimmt man die denkbaren Anwendungen für Micropayments zusammen und bildet diese auf ein einzelnes Kryptowährungssystem wie Bitcoin oder Ethereum ab, ergibt sich eine wage Vorstellung von der geforderten Leistungsfähigkeit zukünftiger Kryptowährungs- und Micropaymentsysteme.

Während die oben genannten Beispiele in geschlossenen, zentralisierten Systemen (z.Bsp. Youtube, Facebook, Twitter) durchaus machbar wären, ist die Abbildung dieser Beispiele in Kryptowährungen und Blockchain-Technologien ein schwieriges Vorhaben. Die Vorteile für die einzelnen Anwender sind aber vielfältig und könnten unsere Vorstellungen von Eigentum und Bezahlungen auf ein global neues Verständnis bringen. Einige dieser Vorteile sind:

- Denzentraler Peer-to-Peer Austausch von Werten
- Vollständige Benutzerkontrolle über das Eigentum
- Wegfall von Zwischenhändlern
- Wenig oder keine Teilnahmebedingungen
- Demokratisierung des weltweiten Zahlungsverkehrs

Recherchen zu aktuellen Definitionen von Micropayments beinhalten die hier genannten Beispiele und Anforderungen meistens noch nicht. Die Entwicklungen in diesem Bereich sind erst wenige Jahre (oder sogar erst Monate) alt. Die Definition von Micropayments in der deutschen und englischen Wikipedia nennt gänzlich andere Beispiel und Anforderungen [2]. Es ist zu erwarten, dass

[2] `https://en.wikipedia.org/wiki/Micropayment`

diese Definitionen sich in den nächsten Jahren stark verändern und anpassen werden. Mit dieser Idee der Anforderungen und Eigenschaften zukünftiger Micropaymentsysteme werden im nächsten Kapitel einige aktuelle, technische Entwicklungen vorgestellt.

# 3. Aktuelle Entwicklungen

## 3.1. Paymentchannels

### 3.1.1. Konzept

Ausgehend von dem weiter oben beschriebenen Problem der Skalierung von Blockchains für Micropayments, sind Paymentchannels ein Ansatz zur Lösung. Zur Verdeutlichung des Konzepts nehmen wir die Akteure Alice und Bob an. Alice und Bob möchten mehrfach aufeinanderfolgende Zahlungen in einem gewissen Zeitraum austauschen. Anstatt jeden einzelnen Zahlungsvorgang in der Blockchain als Transaktion zu übertragen, eröffnen die Beiden einen Paymentchannel.

**Ein einzelner Paymentchannel:**
Für diesen Paymentchannel (weiterhin nur als Channel bezeichnet) vereinbaren Alice und Bob eine Gesamtsumme und eine Gesamtzeit (Abbildung 3.1). Nun werden nur diese vereinbarten Daten in der Blockchain festgehalten (Alice, Bob, Betrag, Zeit). Damit reduziert sich die Anzahl der Transaktionen in der Blockchain auf, im minimalsten Fall eine Start- und eine End-Transaktion. Der Channel verwaltet ab jetzt die vereinbarte Gesamtsumme treuhändisch für Alice und Bob. Dies ist die Kapazität des Channels.

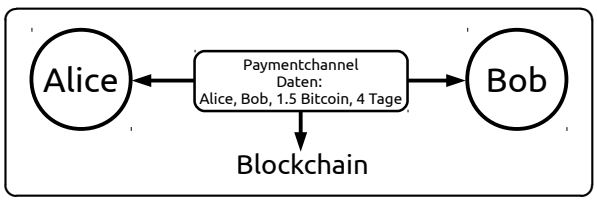


Abb. 3.1.: Paymentchannel zwischen Alice und Bob

Alle Zahlungsvorgänge zwischen Alice und Bob werden dann ausserhalb der Blockchain (genannt „Offchain“) transferiert. Wird der Transfer nur in eine Richtung festgelegt, wird von einem unidirektionalen Channel geprochen. Sind beide Richtungen festgelegt, ist der Channel bidirektional. Die Transfersumme kann nie die Kapazität überschreiten und der Channel kann jederzeit von beiden Seiten geschlossen werden. Wird der Channel geschlossen oder das zeitliche Ende ist erreicht, wird der aktuelle Stand des transferierten Guthabens als Endtransaktion in die Blockchain geschrieben. Der Channel ist damit beendet.

**Viele Paymentchannels:**

Anhand eines weiteren Akteurs (Chris) betrachten wir, wie Paymentchannels im gesamten Netzwerk skalieren können. Alice möchte nun nicht nur mit Bob Zahlungen austauschen, sondern auch mit Chris. Die erste, einfachste Möglichkeit für Alice wäre die Eröffnung eines weiteren Channels mit Chris (Abbildung 3.2). Dieses Vorgehen wird für Alice allerdings schnell recht teuer. Für jeden weiteren Channel muss Alice wieder Guthaben zur Eröffnung an die treuhändische Funktion des Channels „einfrieren“.

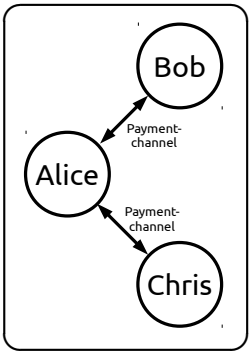


Abb. 3.2.: Alice's Channels

Was aber, wenn Bob einen Channel mit Daniel offen hat und Daniel seinenseits einen Channel mit Chris? Alice müsste dann gar keinen eigenen Channel zu Chris eröffnen, sondern könnte die bestehenden Channels von Bob und Daniel nutzen, um mit Chris zu interagieren. Dieses Channel-routing ist in Abbildung 3.3 dargestellt. Natürlich können Alice und Chris nur soviel Guthaben transferieren, wie die minimalste Channel-Kapazität auf dem gesamten Weg zulässt.

**Globales Paymentchannel Routing:**
Aber es könnten ja auch mehrere Wege von Alice zu Chris existieren. Für die Auswahl des passenden Wegs ist ein Routing-Algorithmus im Netzwerk notwendig. Und spätestens jetzt fällt die Ähnlichkeit des Paymentchannels-Konzepts zum bestehenden Internet-Routing mit TCP/IP auf. Als globales

Netzwerk aus Channels erweitert sich die Funktionalität von Kryptowährungen auf routbare, kleine Pakete. Eine dezentrale Infrastrukur, vergleichbar dem TCP/IP-Routing im Internet könnte mit weltweiten Paymentchannels funktionieren. Dann würde nicht mehr die Frage, ob es einen Channel zwischen zwei Netzwerk-Knoten gibt, sondern die Frage nach dem schnellsten und günstigsten Routing im Vordergrund stehen. Im Fazit in Kapitel 5 wird diese Möglichkeit als das „Internet der Werte“ aufgegriffen.

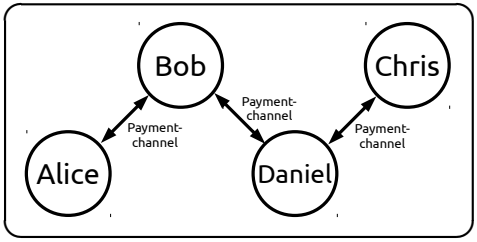


Abb. 3.3.: Paymentchannel-Routing

### 3.1.2. Bitcoin Lightning Network

Die Implementierungen von Paymentchannels in Bitcoin tragen den Namen „Lightning Network“ [PD16]. Für das Lightning Network wurde die Entwicklung des Standards BOLT (Basis of Lightning Technology) angefangen und ist mittlerweile weit fortgeschritten [Com]. Die Spezifikationen in BOLT sind als OpenSource veröffentlicht und enthalten jetzt schon 75 Tests auf Kompabilität für die Anwendungsentwicklung. Mehrere Firmen arbeiten derzeit an Implementierungen des Lightning Network Protokolls. Hier sind die bekanntesten drei aufgelistet:

- Lightning Network Deamon (Lightning Network Labs) in Go [1]
- C-Lightning (Blockstream) in C [2]
- Eclair (ACINQ) in Scala [3]

[1] `https://github.com/lightningnetwork/lnd`
[2] `https://github.com/ElementsProject/lightning`
[3] `https://github.com/ACINQ/eclair`

Anfang 2018 haben alle drei Implementierungen erstmalig die 75 BOLT Kompabilitätstests bestanden und die ersten Lightning Network Transaktionen wurden auf dem Bitcoin-Mainnet getätigt. Der BOLT-Standard garantiert die Interoperabilität zwischen diesen Implementierungen und es ist zu erwarten, dass in 2018 noch sehr viel von dieser Technologie zu hören sein wird. Die Zielvorgaben für die langfristige Entwicklung sind in ähnlichen Größenordnungen wie TCP/IP Routing angesetzt, also mehrere Milliarden Transaktionen pro Tag.

### 3.1.3. Ethereum: Raiden Network

Auch in Ethereum wird das Konzept von Paymentchannels von verschiedenen Unternehmen implementiert. Die Basis dieser Implementierungen ist der ERC20-Token Standard in Ethereum. Damit sind die bisherigen Implementierungen als Smart Contracts gestaltet. Dies ist ein wesentlicher Unterschied zum Lightning Network. ERC-20 Smart Contracts bieten ein festgelegtes Set an Funktionen und Eigenschaften, um mit anderen Smart Contracts auf diesem Standard kommunizieren zu können. Die wahrscheinlich bekannteste Entwicklung ist das Raiden Network [4] von der Brainbot AG aus Mainz. Während das Raiden Network die grundlegende Funktionalität für Paymentchannels herstellt, ist darin das Unterprodukt Micro-Raiden eingebettet. Mit Micro-Raiden wurde auf die häufig verwendete Funktionalität des „many to one“-Transfers mit einem festgelegten, bekannten Empfänger eingegangen und ein System ohne Transaktionsgebühren für diesen Spezialfall geschaffen.

Eine weitere, aktuelle Implentierung für Paymentchannels ist unter dem Namen Orinoco bekannt [5]. Auch Orinoco ist als ERC-20 kompatibler Smart Contract gestaltet, funktioniert schon auf dem Ethereum-Ropsten Testnetzwerk. Die Community um Orinoco scheint weniger groß als die des Raiden Networks zu sein und die Entwicklung daher etwas langsamer voran zu schreiten.

Weitere Micropayment-Entwicklungen für Ethereum sind in 2018 zu erwarten.

[4] `https://raiden.network/`
[5] `http://www.orinocopay.com/`

## 3.2. Schnelle Kryptowährungen

### 3.2.1. Konzept

Einige Kryptowährungen gehen ganz andere Wege, als die bisher beschriebenen Paymentchannels, um Micropayments zu ermöglichen. Wird eine Kryptowährung vollständig von Grund auf neu entwickelt, kann die Erfahrung der letzten 10 Jahre über Blockchain-Technologien (und damit z.Bsp. Bitcoin) für ganz neue Konzepte genutzt werden. Meistens wird hierbei die grundlegende Eigenschaftsmenge (Kapitel 1) verändert. So kann durch Einschränkung der Eigenschaften „Vertrauenslos“ oder „Dezentral“ ein großer Teil der in Bitcoin vorhandenen, aufwendigen Mechanismen zur Einhaltung dieser Eigenschaften reduziert werden. Das führt in aller Regel zu schnelleren, günstigen kleinen Transaktionen. Ob diese Konzepte dann allerdings auch genauso robust und stabil wie Bitcoin sind, steht in den meisten Fällen noch aus und wird erst mit der Zeit durch intensive Analyse (oder auch Angriffe) geklärt werden. Die im Folgenden vorgestellten Kryptowährungen sind eine kleine Auswahl der aktuellen Entwicklungen und decken bei weitem nicht das gesamte Spektrum ab. Es sind aber ausschließlich solche hier aufgelistet, die schon eine gewisse Marktbeachtung gefunden haben und eine aktive Entwicklergemeinschaft aufweisen können.

### 3.2.2. IOTA

Das auffälligste Merkmal der, von einem Berliner StartUp entwickelten, Kryptowährung IOTA ist die Verwendung eines gerichteten, azyklischen Graphen (Directed acyclic graph, DAG) als innere Datenstruktur anstelle einer Blockchain. Das Anwendungsszenario für IOTA soll zukünftig das Internet der Dinge sein. Damit hat IOTA den Anspruch, auf kleinsten Computern in Gegenständen des Alltags lauffähig zu sein. Vorweggenommen sei hier, dass IOTA davon noch ziemlich weit entfernt ist, aber in letzter Zeit einige weiterführende Kooperationen mit Automobilherstellern und -zulieferern eingegangen ist. Diese könnten die Entwicklung maßgeblich beschleunigen. Der verwendete DAG ist symbolhaft in Abbildung 3.4 dargestellt. Jeder Knoten im DAG ist die Entsprechung der Blöcke in einer Blockchain. Die Nummern der Knoten stellen die Einfügereihenfolge dar und jeder Knoten verfiziert genau zwei vorherige Knoten (gerichtet). Es darf kein Kreis aus gerichteten Kanten entstehen, somit

bleibt die azyklische Eigenschaft erhalten. Genau wie in Blockchains, werden die Knoten mittels Hashing ineinander eingehängt und ergeben somit eine unveränderliche Datenstruktur. Nur muss in diesem DAG nicht jeder Knoten das vollständige Abbild der vergangenen Transaktionen kennen. Somit reduziert sich die Datenhaltung auf jedem einzelnen Knoten auf ein Minimum, gegenüber einer Blockchain. Damit soll die Lauffähigkeit auf winzigen IoT-Geräten ermöglicht werden. Weiterhin sind in IOTA keine Transaktionsgebühren vor-

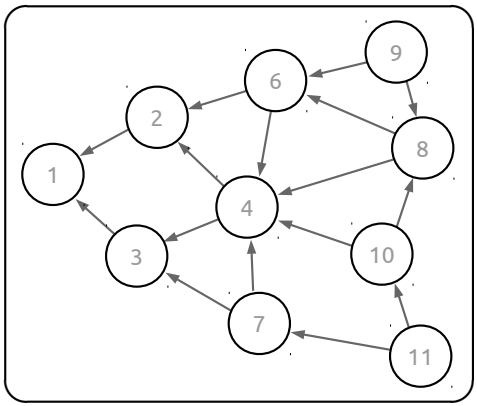


Abb. 3.4.: IOTA: Gerichteter azyklischer Graph (DAG)

gesehen. Es ist auch kein Proof-of-Work oder Mining in IOTA vorhanden. Die „Belohnung“ für das erfolgreiche Verifizieren von zwei vorausgehenden Transaktionen ist die Möglichkeit zur Erstellung einer eigenen Transaktion.

Eine weitere Besonderheit in IOTA ist die interne Daten-Verarbeitung im ternären System. Damit sind 3 Zustände als kleinste Einheiten erlaubt. In heutigen Computern hat sich das binäre System durchgesetzt („0“ und “1“ als Zustände), aber schon Donald Knuth hat in seinem Standardwerk „The art of computer programming, volume 2“ [Knu97] auf die Vorzüge von ternären Systemen hingewiesen. In der Zukunft sind eigene Hardware-Entwicklungen für IOTA und das Internet der Dinge mit ternären CPUs denkbar.

Durch die Verwendung von DAGs, dem Wegfallen von Proof-of-Work, der Optimierung durch das ternäre System und die nicht vollständige Konsens-

Kentniss jedes Teilnehmers kann IOTA (in der Theorie) für Micropayments im Internet der Dinge geeignet sein. Im Moment wird IOTA noch ziemlich kontrovers diskutiert, da zum Betrieb des Netzwerk sogenannte Validator-Server eingesetzt werden und diese nicht unter OpenSource veröffentlich wurden. Diese Validatoren sollen bei Erreichen einer Grenze der Größe des Netzwerks unnötig werden und abgeschaltet werden. Derzeitig stellen diese Validatoren aber noch einen zentralen Punkt der Kontrolle über IOTA in der Hand der Entwickler dar.

### 3.2.3. Stellar

Stellar hat das komplett neue Konsens-Modell „Stellar consensus protocol“ (SCP) entworfen und in ihrer Kryptowährung implementiert. Im Vergleich zu den in Kapitel 1 genannten Bitcoin-Eigenschaften weichen die SCP Eigenschaften davon ab und sind im Folgenden genannt:

- Dezentral
- Niedrige Latenz
- Flexibles Vertrauen (flexbile trust)
- Asymptotische Sicherheit (asymptotic security)

Die letzten zwei Punkte unterscheiden sich stark von den meisten anderen Kryptowährungen (z.Bsp. Bitcoin und Ethereum). Anstelle eines komplett vertrauenslosen Systems (trustless), kann in SCP jeder Teilnehmer selbst entscheiden, wem er vertraut (flexible trust). Diese Vertrauensliste ist in jedem Stellar-Knoten dynamisch organisiert. Stellar löst damit das Problem der Byzantinischen Fehlertoleranz in verteilten Systemen mit dem neuen Ansatz „Federated Byzantine Agreement“ (Eingebundene Byzantinische Übereinstimmung). Für die Anpassung der lokalen Vertrauenslisten sind einfache Regeln im SCP festgelegt. Solange die Mehrzahl der Knoten diese Regeln befolgt und sich mit einer maximalen Anzahl von, aus ihrer Sicht vertrauenswürdigen anderen Knoten verbindet, konvergiert das Netzwerk zu einem gemeinsamen Konsens. SCP toleriert also willkürliches und eventuell absichtlich falsches Verhalten und konvergiert trotzdem. Diesen Vorgang nennt Stellar asymptotische Sicherheit (asymptotic security). Knoten, die sich gegenseitig als Gruppe vertrauen, werden in Stellar als Quorum bezeichnet. Eine Untergruppe eines

Quorum ist ein Quorum-Slice. Damit SCP konvergiert, muss zusätzlich sichergestellt sein, dass es keine isolierten Quorums gibt. Die Quorums müssen also überlappen (Quorum-Intersection). Eine einstiegsfreundliche Erklärung über SCP und Quorum wurde vom Stellar-Team als Comic veröffentlicht [Ste15].

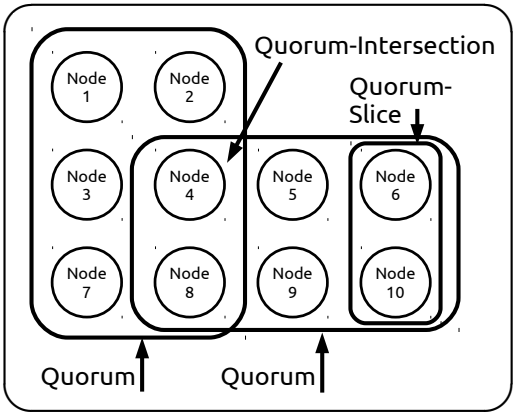


Abb. 3.5.: Stellar: Quorum, Quorum-Slice und Quorum-Intersection

In Stellar gibt es keinen Proof-of-Work Mechanismus. Damit verbunden, ist auch kein Mining möglich und es gibt keinen intrisischen Anreiz (finanzielle Belohnung) für gutes Benehmen. Fehlerhaftes Verhalten wird einfach nur durch das Protkoll reguliert. Genau durch diese Eigenschaften erreicht Stellar allerdings seine sehr hohen Transaktionsgeschwindigkeiten von mehreren tausend Transaktionen pro Sekunde. Die Konstruktion von Stellar und SCP ist vormerklich an internationalem Zahlungsverkehr orientiert und ist daher eine interessante Plattform für Micropayment-Systeme. Einen tieferen Einblick in SCP (vorsicht Mathematik!) gibt David Mazières in diesem Google-Talk [Maz15].

## 3.3. Weitere Kurzvorstellungen

### 3.3.1. DappChains in Ethereum

Die Entwicklergruppe Loom Network hat das Konzept von Sidechains aufgegriffen, modifiziert und unter dem Namen DappChains für die Ethereuem-Blockchain weiterentwickelt. Sidechains sind Forks einer bestehenden Blockchain (Mainchain). Diese Sidechains existieren parallel zur Mainchain und können mittels Checkpoints mit dieser interagieren (Abbildung 3.6). Das Ziel ist die Auslagerung von Transaktionen und Computer-Workload aus der Ethereum-Mainchain heraus. DappChains erweitern dieses Konzept auf die Kopplung an Smart Contracts (Dapps). Jede Dapp forkt eine eigene Sidechain, genannt DappsChain. Diese DappsChains sind für die Skalierung von Datenhaltung und -durchsatz optimiert, nicht wie andere Konzepte für die Optimierung von Transaktionszeiten.

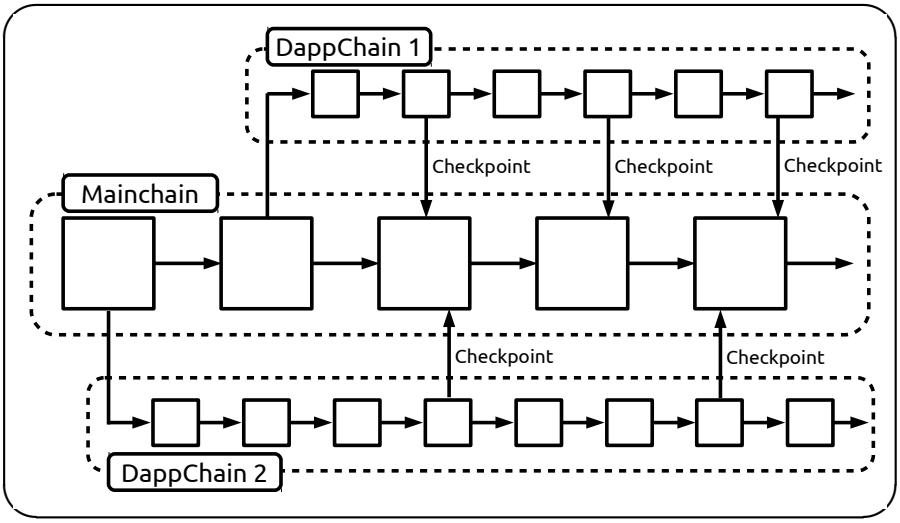


Abb. 3.6.: Loom Network: DappChains

Das Loom Network will die DappChains zusammen mit einem eigenen Software-Development-Kit (SDK) im März 2018 veröffentlichen. Zur Vorbereitung und zum Erlernen der Ethereum Programmiersprache Solidity ist jetzt schon ein

Online-Workshop [6] von Loom erhältlich. In diesem Workshop wird eine Dapp programmiert, welche am Ende des Workshops ein Zombie-Spiel ergibt und zukünftig auf einer DappChain laufen wird. Dieser Ansatz ist nicht nur erfolgversprechend für die Skalierung von Smart Contracts in Ethereum, sondern macht auch ziemlich viel Spaß zum Erlernen der Basics in Solidity.

### 3.3.2. Nano (Ehemals RaiBlocks)

RaiBlocks hat sich Anfang 2018 in Nano umbenannt. Nano verwendet Blockgitter anstelle von Blockchains. Jeder Teilnehmer hält seine eigene Blockchain (Kontokette) mit den Daten seines „Blockchain-Kontos“ vor. Während in herkömmlichen Blockchains die Transaktionshistorie aller Teilnehmer gemeinsam in der Blockchain aufgezeichnet wird, ist in Nano die Guthabenverwaltung jedem Teilnehmer zugeordnet. Nur der Teilnehmer selbst kann seine eigene Kontokette fortführen und um zwischen den Kontoketten zu transferieren sind jeweils zwei Transaktionen notwendig: Eine Sende- und eine Empfangstransaktion. Proof-of-Work wird in Nano nur als „Anti-Spam“ Mechanismus eingesetzt und benötigt wesentlich weniger Computation als z.Bsp. in Bitcoin. Das Sichern des Netzwerks funktioniert in Nano über den Proof-of-Stake Mechanismus. Nano bietet nach eigenen Angaben Skalierung für mehrere tausend Transaktionen pro Sekunde. Für mehr Informationen zu Nano ist das Whitepaper [LeM] zu empfehlen.

[6] `https://cryptozombies.io/`

# 4. Demonstrator

## 4.1. Konzeptidee: Nachbarschaftshilfe

Das Ziel des Demontrators ist die anschauliche, einfach erlebbare Umsetzung des komplexen Themengebiets der Micropayments. Die erste Ideensammlung beinhaltete:

- Konsolenanwendung, rein technisch
- Spielkonzepte (Monopoly, Spiel des Lebens)
- Dezentrale Energiewirtschaft
- Gesellschaft und menschliche Interaktion

Die letztgenannte Idee wurde zur Umsetzung ausgewählt, um die Einstiegshürde einer verbalen Erklärung des Demonstrators möglich niedrig zu halten. Der Demonstrator sollte auch ohne tiefere Kenntnis von Bockchains und Kryptowährungen zu bedienen und zu erfahren sein.

Abb. 4.1.: Die Nachbarn: Alice, Bob, Chris, Daniela

Vier fiktive Personen sind die Protagonisten dieses Demonstrators und agieren in leicht verständlichen Aktionen miteinander. Diese Personen erhalten die

Namen Alice, Bob, Chris und Daniela. Sie werden durch Plastik-Spielfiguren (Abbildung 4.1) abgebildet und sind farblich unterscheidbar (Blau, Grün, Rot, Gelb). Diese Vier sind Nachbarn und besitzen jeweils ein eigenes, kleines Zuhause in ihrer Farbe. Zusammen haben sie eine Nachbarschaftshilfe gegründet. Sie helfen sich untereinander bei den folgenden Tätigkeiten:

- Rasen mähen
- Einkaufen gehen
- Kleine Reparaturen
- Fenster putzen
- Babysitten

Eigentlich würden sie diese Tätigkeiten untereinander direkt mit Bargeld abrechnen. Aber Bob ist Informatiker und Daniela schürft seit geraumer Zeit Bitcoins, also haben sie eine Abbildung der Bezahlvorgänge auf ein Micropayment-System in Kryptowährungen beschlossen. Sie möchten nämlich folgende Eigenschaften für Ihre Nachbarschaftshilfe haben:

- Neue Mitglieder einfach integrierbar
- Skalierbar für ganze Städte (oder größer!)
- Keine zentrale Kontrolle oder Abrechnungsstelle
- Keine oder nur wenig Gebühren für die gegenseitige Bezahlung
- Fair und transparent für Alle
- Direkte Bezahlung über kleine Zeiteinheiten

## 4.2. Plattform: Stellar

Für den Bau und die Implemtierung des Demonstrators wurde ein Drittel des gesamten Projektzeitraums eingeplant. Das ergibt etwas mehr als einen Monat für den gesamten Demonstrator inklusive dieser Dokumentation. Wie weiter oben beschrieben, sind Lightning Network und Raiden Network noch in frühen Entwicklungsphasen und daher nicht allzu umfangreich dokumentiert.

IOTA ist noch weit davon entfernt, auf kleiner Hardware zu funktionieren. Aber Stellar hat gute Dokumentation, APIs in mehreren Sprachen und die Netzwerk-Infrastruktur ist gut skalierbar. Es ergab sich der Eindruck, dass mit Stellar die Zeitvorgabe für den Demonstrator am ehesten einzuhalten ist. Die anderen Systeme sind mit Sicherheit interessant und eventuell sogar besser für die Projektidee geeignet. Aber die tollste Technologie nützt wenig, wenn das Projekt damit nicht fertig wird. Also wurde Stellar als Basis für die Implementierung ausgewählt.

## 4.3. Hardware

Aus der Definition der Nachbarshaftshilfe ergibt sich der Bedarf für die vier Spielfiguren und deren Häuser. Die Spielfiguren sollen mit jedem Haus interagieren können. Dafür ist eine Erkennung der jeweiligen Spielfigur als Akteur an jedem Haus notwendig. Hierfür musste Sensorik entwickelt werden. In Abbildung 4.2 ist Bobs Haus (in Grün, wie Bob auch) zu sehen. Vor dem Haus sind drei Aktionsfelder für interagierende Nachbarn installiert. Im Bild besetzen Bob selbst, Daniela und Alice diese Felder. Sie verrichten damit jeder eine unterschiedliche Tätigkeit für Bob. Bob übt eine dieser Tätigkeiten selbst aus.

Abb. 4.2.: Bobs Zuhause mit interagierenden Nachbarn

Die Technik in einem solchen Haus besteht aus jeweils einem Raspberry Pi 3 mit WLAN, einem LCD-Display und den Sensoren für die Erkennung der Nachbarn. Abbildung 4.3 zeigt eine Komponentenübersicht dieser Bestandteile.

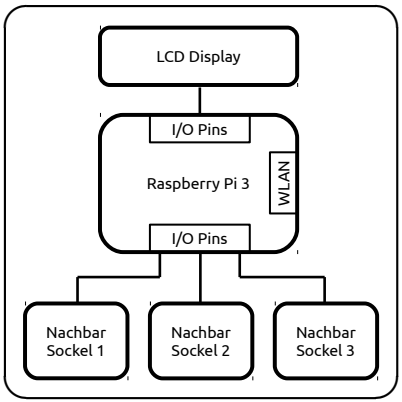


Abb. 4.3.: Komponenten eines Hauses

Um die Nachbarn in den Sockeln unterscheiden zu können, sind die Nachbarn unär kodiert. Abbildung 4.4 zeigt diese Kodierung. In den Sockeln sind jeweils vier Microtaster verbaut und an die I/O-Pins des Raspberry Pi verbunden. Jeder Nachbar löst bei Einschub in einen Sockel genau einen dieser vier Taster aus und der Raspberry Pi kann softwareseitig erkennen, welcher Nachbar in welchem Sockel steht.

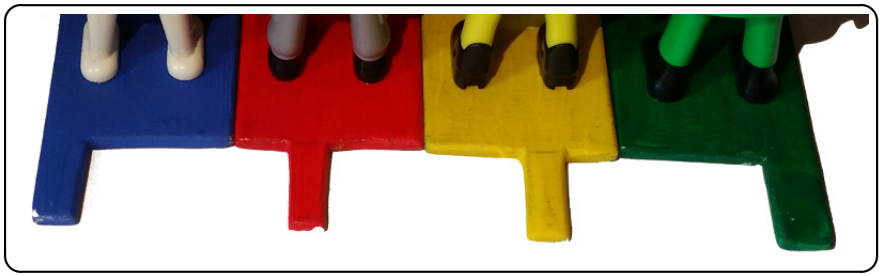

Abb. 4.4.: Unäre Kodierung der Nachbarn

Zur korrekten Erkennung der Nachbarn mussten die Microtaster entprellt werden. Hierzu sind in den Python-Libraries softwareseitige Parameter einstellbar. Diese brachten aber nicht vollständig den gewünschten Effekt. Alleine nur mit Software-Entprellung wurden die Nachbarn oft nicht oder falsch erkannt. Erst die Entprellung mit kleinen 100 nF Kondensatoren an jedem Taster brachte das gewünschte Ergebnis, in Kombination mit der softwareseitigen Lösung.

Die Erkennung der Nachbarn kennt die Ereignisse „Nachbar X fängt Tätigkeit Y an“ und „Nachbar X stoppt Tätigkeit Y“. Diese Erkennung wird auf dem LCD-Display (Abbildung 4.5) in den entsprechenden Farben der Nachbarn darstellt und triggert im Raspberry Pi die Bezahlvorgänge im Stellar-Netzwerk.

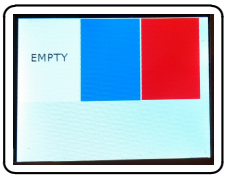


Abb. 4.5.: Display

Der gesamte Demonstrator besteht dann aus einem Android-Tablet, den Nachbarn, ihren Häusern und einem Laptop. Die Nachbarn interagieren mit den Häusern und triggern dabei Transaktionen in Stellar. Diese Transaktionen führen zu Guthaben-Verschiebung der Nachbarn. Deren aktuelles Guthaben und dessen Veränderung wird auf dem Laptop dargestellt. Das Android-Tablet zeigt einen Stellar-Blockchain-Explorer an. Mit diesem Explorer sind die laufenden Transaktionen live in der Stellar-Blockchain zu sehen. So bildet sich für den Benutzer die Kette aus Aktion, Transaktion und Live-Ansicht. Die Geschindigkeit des Stellar-Netzwerks ist hierdurch direkt durch eigenes Handeln in mehreren Stufen zu erfahren.

## 4.4. Software

### 4.4.1. Stellar Produkte

Das Stellar Universum macht einen recht aufgeräumten Eindruck und lädt dazu ein, einfach mal auszuprobieren. Die Dokumentation ist in verschiedenen Schwierigkeitsstufen organisiert. Einführende Erläuterungen und Videos helfen zum Einstieg, aber die tiefergehenden Informationen sind jeweils auch nur wenige Klicks entfernt. Das Hauptprodukt ist der Stellar-Core-Server. Dieser ist vergleichbar dem Bitcoin-Core-Client und kann genauso selbst aus dem OpenSource-Quellcode kompiliert und vielfältig konfiguriert werden. Für den

Start ist dies aber gar nicht notwendig. Stellar bietet erstens öffentliche Core-Server zur Benutzung und zweitens fertig konfigurierte Docker-Images zum Download an. Die Core-Server können Transaktionen und Blöcke validieren, die Blockchain durch SCP fortführen und das Netzwerk aktuell halten. Darauf aufbauend sind die Stellar-Horizon-Server die Anbindung an eigene Anwendungen. Die Horizon-Server bieten Application-Programming-Interfaces (APIs) an, welche dann in verschiedenen Programmiersprachen als Bibliotheken implementiert sind. In dem beschriebenen Core-Server-Docker-Image ist ein Horizon-Server mit enthalten. Implementierungen der API gibt es unter anderem in JavaScript, Go, Java, Python, Ruby und C#. Abbildung 4.6 stellt eine Übersicht der Stellar-Produkte, deren Zusammenspiel und der Einordnung der Demonstrator-Anwendungen dar.

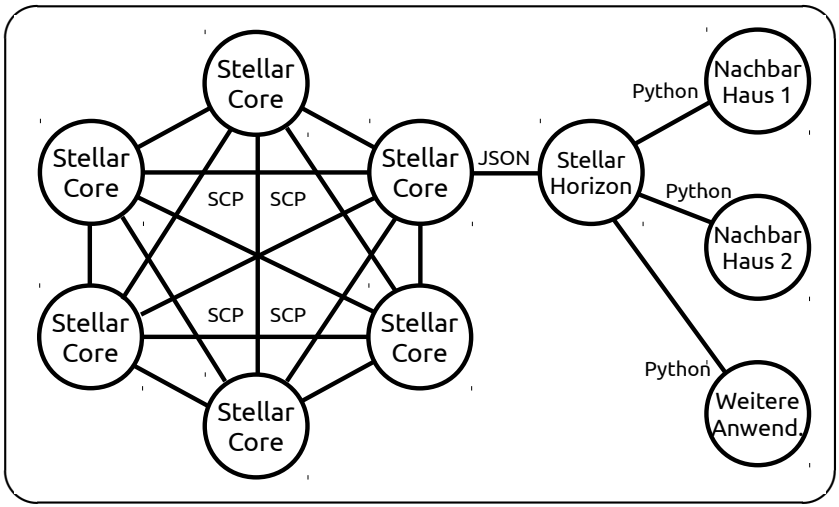


Abb. 4.6.: Stellar: Core- und Horizon-Server mit Anwendungen

### 4.4.2. Stellars Python API

Da in den Nachbarhäusern Raspberry Pis venwendet werden, bietet sich die Benutzung der Python API an. Die Installation der Bibliotheken ist einfach

über Python-PIP möglich. Das PIP-Paket heisst stellar-base. Von da ab ist der Weg ausführlich in der Dokumentation der Python-Implementierung [1] festgehalten. Dies sind hauptsächlich die folgenden Punkte:

- Erstellung eines Stellar-Schlüsselpaars
- Erstellung eines Stellar-Accounts mit den Schlüsseln
- Anfrage für Guthaben im Stellar-Testnetzwerk
- Transaktionen und Guthaben-Abfragen

Der Python-Quellcode zu diesen Vorgängen wird in den folgenden zwei Unterkapiteln zusammen mit den Implementierungen besprochen.

### 4.4.3. Die Haus-Software

Die Häuser der Nachbarn mit den eingebauten Raspberry Pis halten die Schlüsselpärchen ihrer Besitzer. Jedes Haus kann also den Stellaraccount seines Besitzers selbstständig verwalten. Das ist notwendig, da die Nachbarn für ihre Tätigkeiten direkt ausbezahlt werden sollen. Die Häuser weisen also bei Erkennung eines Nachbarn im Sockel vor dem Haus kleine Überweisungen an den jeweiligen Nachbarn an. Diese Überweisungen (Transaktionen) werden im festgelegten Intervall von 5 Sekunden immer wieder ausgelöst, solange der Nachbar tätig ist (im Sockel vor dem Haus steht). Die verschiedenen Sockel stellen verschiedene Tätigkeiten dar und werden unterschiedlich entlohnt:

- Sockel links: 1 Stellar pro 5 Sekunden
- Sockel mitte: 2 Stellar pro 5 Sekunden
- Sockel rechts: 3 Stellar pro 5 Sekunden

Der Quellcode der Haus-Software ist im Anhang A.1 aufgelistet. Die Haus-Software muss die im Folgenden erläuterten Funktionen erfüllen.

**Verwaltung der Schlüssel:**
Die Stellar Accounts der Nachbarn wurden im Vorfeld mit ein paar Zeilen Pythoncode aus der Stellar Python-Dokumentation erstellt (siehe Fußnote).

[1] `https://github.com/StellarCN/py-stellar-base`

Die Accounts liegen im Stellar-Testnetzwerk und wurden mit jeweils 10000 Lumen (die Coin-Einheit in Stellar) aufgeladen. Diese Coins sind nur im Testnetzwerk zu benutzen und haben daher keinen Gegenwert in Euro oder Dollar. Für die Erstellung und Aufladung der Accounts existiert kein eigenes Programm, sondern die erstellten Schlüssel wurden in Python Notebook live erstellt. Die öffentlichen Schlüssel aller Nachbarn und der private Schlüssel des jeweiligen Hausinhabers sind als globale Variablen im Quellcode definiert (Zeilen 26-30).

**Fullscreen - Die Klasse für das Display:**
Die Klasse `Fullscreen` (Zeilen 32-56) erstellt im Konstruktor das Tkinter-Rootwidget `self.tk` für die Darstellung auf dem LCD-Display. Dieses Rootwidget wird mit drei Labels aus der Tkinter Bibliothek befüllt. Diese Labels können Bilder für die Statusse der Sockel (empty.gif, red.gif, green.gif, blue.gif, yellow.gif) aufnehmen. Die Anordnung der Labels auf dem Bildschirm wird mit dem Tkinter-Gridmanager gelöst. Die drei Labels erscheinen dann nebeneinander auf dem Bildschirm (Abbildung 4.5). Drei Updatefunktionen für die drei Labels können diese Bilder jeweils ändern.
Die Instanz von `Fullscreen` wird im Hauptteil des Programms als `root` erstellt (Zeilen 249-256) und die Labels initial mit dem Bild empty.gif befüllt. In Zeile 265 wird die Tkinter-Funktion `tk.mainloop()` auf `root` ausgeführt und läuft ab da in einer Endlossschleife.

**Überwachung der I/O-Pins auf Events in den Sockeln:**
Zur Überwachung der I/O-Pins mit den jeweils 4 Tastern in jedem Sockel wird die Python Bibliothek RPi.GPIO verwendet. Für jeden Sockel ist eine eigene Callback-Funktion enthalten (Zeilen 147-208). Nach der Initialisierung der I/O-Pins (Zeilen 209-225) werden die Callback-Funktionen den Pins zugeordnet. Somit löst jeder Taster (3 Sockel * 4 Taster = 12 Taster pro Haus) eine Callbackfunktion aus und in der Funktion wird die entsprechende globale Sockel-Variable (slot1, slot2, slot3) auf den aktuellen Status geändert. Somit enthalten diese Variablen immer den aktuellen Status jedes Sockels. Die Erkennung, welcher Nachbar in welchem Sockel steht ist damit auf Zahlen in den Sockelvariablen abgebildet und wird durch die Callback-Funktionen aktualisiert. Zusätzlich ändern die Callback-Funktionen das Label-Bild auf dem Display für den entsprechenden Sockel. Die Aktualisierung erfolgt in Tkinter sofort und der sichtbare Status des Sockels ändert sich (Anzeigen der Farbe des Nachbarn im Sockel).

**Auslösen der Stellar-Transaktionen:**
Die Klasse `Sendthread` (Zeilen 61-145) fragt den Zustand der Sockel ab und erstellt danach die Stellar-Transaktionen. Hier sind die Preise für die Tätigkeiten (Sockel 1-3) festgelegt. Die Bezahlung findet jeweils vom Besitzer des Hauses zu dem, im Sockel stehenden, Nachbarn statt. Hierzu werden die global definierten Schlüssel verwendet. Ist eine Transkation (Builder1 bis Builder3) fertiggestellt, wird sie an das Stellar-Netzwerk übertragen und dort ausgeführt.
`Sendthread` wird, wie der Name schon vermuten lässt, als Python-Thread instanziiert (Zeilen 262-263) und läuft ab da dauerhaft parallel zum Tkinterwidget `root`. In `Senthread` ist ein delay von 5 Sekunden eingebaut (Zeile 141). Somit werden die aktuellen Sockel-Belegungen alle 5 Sekunden abgefragt und bei Bedarf die notwendigen Transaktionen ausgeführt.

### 4.4.4. Der Guthaben-Watchdog

Das Python Programm `watchdog.py` ist viel einfacher aufgebaut. Es fragt von den global definierten öffentlichen Schlüsseln der Nachbarn das Guthaben in einer Endlossschleife alle 2 Sekunden ab und stellt dieses in der Konsole dar. Hier für sind die Schüssel wieder in Variablen gespeichert (Zeilen 12-16). Die Instanzen der Klasse `Address` aus der Stellar Bibliothek erhalten diese Schlüssel und die Anwendungen der Funktion `Address.get()` holt die Informationen über das aktuelle Guthaben dieses Schlüssels aus der Stellar-Blockchain.

## 4.5. Ergebnis

Der Demonstrator ist eine kleine, übersichtliche Einheit geworden, die das Prinzip von dezentralen Bezahlungsvorgängen als Micropayments zum Anfassen und Spielen ermöglicht. Die Implementierung der Interaktionen mit der Stellar Blockchain sind durch die Python API schlank und verständlich abgebildet. Von einer Schlüsselverwaltung in globalen Variablen ist für produktive Systeme im höchstem Maße abzuraten, aber für diesen Demonstrator völlig ausreichend. Die sichere Schlüsselverwaltung in IoT-Geräten ist ein ganz eigenes Forschungsfeld und kann hier nicht ausführlicher betrachetet werden.
Die besprochene Implementierung enthält an einigen Stellen Code-Redundanz

und könnte wesentlich kürzer programmiert sein. Da der Quellcode aber insgesamt nicht allzu lang ist, wurde auf intensives Refactoring verzichtet. Die Verständlichkeit ohne weitere Kapselung erschien dem Autor besser und lesbarer. Das ist aber eventuell nur eine „Geschmacksfrage“ des Codingstils.

Der Zeitaufwand für den Bau dieses Demonstrators lag höher als vorher gedacht, daher ist nur ein Haus entstanden (Das Haus von Bob, grün). Mit diesem Haus sind aber alle Funktionen aufzeigbar. Die Implementierung wurde ausgiebigen Livetests unterzogen und hat selbst ohne WLAN, sondern nur mit schlechter Mobilfunkverbindung genau innerhalb der gewünschten Parameter funktioniert (Transaktionen alle 5 Sekunden). Stellar erscheint als robustes, einfach zu implementierendes Bezahlungsnetzwerk mit der nötigen Geschwindigkeit für viele Micropayment-Anwendungen.

# 5. Fazit

## 5.1. Komplexität

Aus der Analyse über die derzeitigen Transaktionszeiten und -kosten in Bitcoin und Ethereum hat sich dargestellt, dass die aktuelle Situation weit von Alltagstauglichkeit entfernt ist. Die Betrachtung der Beispiele für Micropayments haben einen Einblick in die zu erreichenden Größenordnungen gegeben. Die Beispiele zeigen aber auch, dass noch längst nicht alle Ideen für Micropayments gedacht, gesagt und geschrieben wurden. Die Entwicklung dieser Anwendungsideen ist noch in der Startphase und intensive Diskussionen über Gesellschaft, Soziales, Technik und viele weitere Gebiete sind notwendig. Informatik alleine kann diese Aufgabe nicht bewältigen, es scheint eher eine gesamtgesellschaftliche Aufgabe zu sein. Die Komplexität von Kryptowährungen und Micropayments in 5 oder 10 Jahren ist im Moment schlecht abschätzbar. Die Entwicklung der technischen Möglichkeiten in Form von neuen Kryptowährungen oder Erweiterungen für Bestehende nimmt auch gerade erst Fahrt auf und ist genauso wenig für das kommende Jahrzent vorauszusehen. Auch die Aufgabe der Bewertung dieser Anwendungsideen und Technologien ist nur als Gemeinschaftsarbeit zu lösen. Es gibt einzelne Akteure (z.Bsp. Andreas Antonopoulos, Jimmy Song, Charlie Lee), die sich in jahrelanger Arbeit eine sehr respektable Übersicht verschafft haben. In den folgenden, dieses Buch abschließenden Kapiteln werden zwei weiterführende und teilweise als utopisch erscheinende Bilder gezeichnet, die in ihrer Art die Dringlichkeit der Diskussion durch alle Gesellschaftsbereiche darstellen.

## 5.2. Internet der Werte

Das bestehende Internet basiert als stark vereinfachte Darstellung auf einem dezentralen Routing der TCP/IP Pakete. Die Idee von Paymentchannels im

Lightning Network könnte zu einer ähnlichen, dezentralen Routing-Struktur führen. Jeder einzelne Paymentchannel ist nur zum Austausch von Werten zwischen zwei Teilnehmern fähig. Viele davon bilden ein dezentrales, eventuell weltumspannendes Netzwerk. Im BOLT-Standard (Basis of Lightning Technology) sind solche Routing-Mechanismen schon integriert (Onion-Routing). In Zukunft könnte die Frage nach der Erreichbarkeit eines Zahlungspartners also nicht die Frage nach einem einzelnen Paymentchannel sein, sonder die Frage nach dem bestmöglichen Routing zu ihm. Heimnetzwerk-Geräte (auch oft als WLAN-Router bezeichnet) könnten, genau wie für TCP/IP-Pakete, in der Lage sein, Micropayments durch das weltweite Netzwerk zu routen. Ein Internet der Werte wäre entstanden. Die Anwendungen und die Funkionalität eines solchen Internets der Werte wird nur durch die Kreativität der Benutzer begrenzt. Jeder Benutzer hat mittels seiner kryptographischen Schlüssel volle Kontrolle über sein Guthaben, kann aber mit jedem beliebigen anderen Benutzer dieses Guthaben als Werte austauschen. Von außen betrachtet, könnte die Benutzung ähnlich wie mit einem Internetbrowser erfolgen. In Browsern werden Internetadressen in einer lesbaren Form eingegeben und der Benutzer braucht kein Wissen um das Routing der Pakete bis zu (oder von dieser) Adresse.

## 5.3. Ein ungewöhnliches Taxi

Derzeit wird weltweit an autonomen Fahrzeugen geforscht. Erste Testfahrzeuge befahren die Strassen dieser Welt (leider noch nicht so sehr in Deutschland). Nehmen wir einfach mal ein paar Zutaten aus der Welt der Kryptowährungen hinzu und bauen ein leicht utopisches Szenario daraus zusammen. Die benötigten Zutaten sind:

- Selbstfahrendes Auto
- Bitcoin-Konto für Gegenstände (IoT)
- Smart Contracts
- Micropayments

Das selbstfahrende Auto kann als Taxi Fahrgäste befördern und hat damit eine Einnahmequelle (Abbildung 5.1). Die Fahrgäste bezahlen das Taxi in

Kryptowährungen. Da das Taxi ein eigenes Schlüsselpärchen besitzt, hat es auch ein eigenes Kryptowährungskonto in der Blockchain. Braucht das Taxi neue Reifen oder sonstige Wartung, fährt es in die Werkstatt und lässt sich warten. Im Fall eines Liegenbleibens unterwegs alarmiert das Taxi einen Reparaturservice. Diese Ausgaben bezahlt das Taxi selbstständig von seinem Kryptowährungsguthaben in der Blockchain.

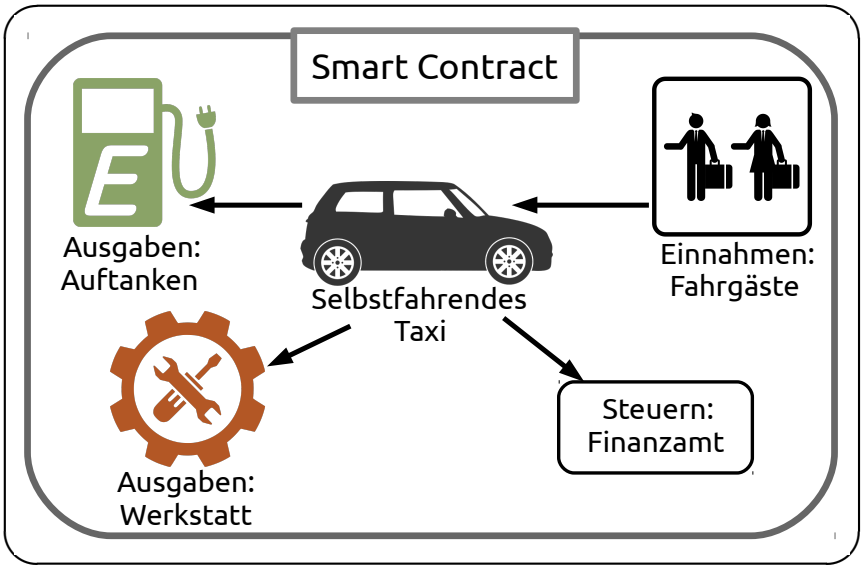


Abb. 5.1.: Selbstfahrendes Taxi als autonomes Unternehmen

Bildquelle für Cliparts: `https://openclipart.org`

Elektrische Aufladung oder Tanken funktioniert auf die gleiche Weise und Bezahlung. Nicht zu vergessen ist die Besteuerung der Ein- und Ausgaben durch das Finanzamt. Auch die Steuerbescheide erledigt das Taxi selbstständig mit der staatlichen Finanzbehörde. Und all das zusammen wird über einen Smart Contract abgebildet. Das Taxi ist damit sein eigenes, autonomes Unternehmen ohne Mensch. Hat das Taxi genug Gewinn erwirtschaftet, kauft es ein weiteres Taxi für sein Unternehmen. Und so weiter...

## 5.4. Lessons learned

Die inhaltliche Abgrenzung für dieses Buch war eine schwierige Aufgabe. In Kryptowährungen und Blockchain-Technologien scheint die Entwicklungskurve exponentiell zu verlaufen. Im Fokus auf den roten Faden durch die Erzählung mussten einige Ansprüche auf Vollständigkeit und Darstellungstiefe abgeschwächt werden. Es ist absehbar, dass Micropayments in Zukunft eine wachsende Rolle im allgemeinen Internet spielen werden. Die Umsetzung eines Internets der Werte scheint mit dem Lightning Network greifbar nahe. Die Fachbereiche der Informatik an den Hochschulen sollten schnellstens Angebote und Kurse zu diesen Themen erstellen. Ein direktes Folgeprojekt der, mit diesem Buch vorliegenden, studentischen Projektarbeit ist geplant. Auch die Anzahl der verfügbaren Informatik-Lehrbücher über Kryptowährungen und Blockchain-Technologien ist bisher bedrückend gering und bietet genügend Raum für weitere Projekte.

„IoT ist, wenn der Toaster Bitcoin schürft,
um seine Spielschulden beim Kühlschrank zu bezahlen.“
Quelle: Internetforum, Autor unbekannt

# A. Anhänge

## A.1. Die Haus-Software

```
1  #-------------------------
2  #------- Fullscreen ------
3  #-------------------------
4  # Author: Thorsten Knoll
5  # Date:    28.11.2017
6  #-------------------------

8  import RPi.GPIO as GPIO
9  import sys
10 from tkinter import *
11 from PIL import Image, ImageTk

13 from stellar_base.keypair import Keypair
14 from stellar_base.address import Address
15 from stellar_base.builder import Builder

17 import requests
18 import threading
19 import time

21 # set global slot variables ---------------------------------------
22 slot1 = 0
23 slot2 = 0
24 slot3 = 0

26 # set global publickey variables ----------------------------------
27 publickey_1 = '<here goes the public key 1>' #yellow
28 publickey_2 = '<here goes the public key 2>' #green
29 publickey_3 = '<here goes the public key 3>' #red
30 publickey_4 = '<here goes the public key 4>' #blue

32 # set seed (from neighbor 1 / Green) ------------------------------
33 seed = '<here goes the seed for the private key>'

35 class Fullscreen:

37     def __init__(self):
38         self.tk = Tk()
39         self.tk.attributes('-fullscreen', True)
40         self.img_e = ImageTk.PhotoImage(Image.open("empty.gif"))
41         self.label1 = Label(self.tk, image=self.img_e)
42         self.label2 = Label(self.tk, image=self.img_e)
43         self.label3 = Label(self.tk, image=self.img_e)
44         self.label1.grid(row=0, column=0, sticky="w")
45         self.label2.grid(row=0, column=1)
46         self.label3.grid(row=0, column=2, sticky="e")
47         self.tk.focus_set()

49     def updateLabel1(self, img):
50         self.label1.configure(image=img)
51         self.label1.image=img
```

```
53        def updateLabel2(self, img):
54            self.label2.configure(image=img)
55            self.label2.image=img

57        def updateLabel3(self, img):
58            self.label3.configure(image=img)
59            self.label3.image=img

61    class Sendthread (threading.Thread):

63      def __init__(self):
64        threading.Thread.__init__(self)

66      def run(self):
67        while True:
68          if (slot1 == 1):
69            print("Tyring to send 1 1")
70            builder1 = Builder(secret=seed)
71            builder1.append_payment_op(publickey_1, '1', 'XLM')
72            builder1.add_text_memo('GREEN to YELLOW: 1 XLM')
73            builder1.sign()
74            builder1.submit()
75          if (slot1 == 2):
76            builder1 = Builder(secret=seed)
77            builder1.append_payment_op(publickey_2, '1', 'XLM')
78            builder1.add_text_memo('GREEN to GREEN : 1 XLM')
79            builder1.sign()
80            builder1.submit()
81          if (slot1 == 3):
82            builder1 = Builder(secret=seed)
83            builder1.append_payment_op(publickey_3, '1', 'XLM')
84            builder1.add_text_memo('GREEN to RED   : 1 XLM')
85            builder1.sign()
86            builder1.submit()
87          if (slot1 == 4):
88            builder1 = Builder(secret=seed)
89            builder1.append_payment_op(publickey_4, '1', 'XLM')
90            builder1.add_text_memo('GREEN to BLUE  : 1 XLM')
91            builder1.sign()
92            builder1.submit()
93          if (slot2 == 1):
94            builder2 = Builder(secret=seed)
95            builder2.append_payment_op(publickey_1, '2', 'XLM')
96            builder2.add_text_memo('GREEN to YELLOW: 2 XLM')
97            builder2.sign()
98            builder2.submit()
99          if (slot2 == 2):
100           builder2 = Builder(secret=seed)
101           builder2.append_payment_op(publickey_2, '2', 'XLM')
102           builder2.add_text_memo('GREEN to GREEN : 2 XLM')
103           builder2.sign()
104           builder2.submit()
105         if (slot2 == 3):
106           builder2 = Builder(secret=seed)
107           builder2.append_payment_op(publickey_3, '2', 'XLM')
108           builder2.add_text_memo('GREEN to RED   : 2 XLM')
109           builder2.sign()
110           builder2.submit()
111         if (slot2 == 4):
112           builder2 = Builder(secret=seed)
113           builder2.append_payment_op(publickey_4, '2', 'XLM')
114           builder2.add_text_memo('GREEN to BLUE  : 2 XLM')
115           builder2.sign()
116           builder2.submit()
117         if (slot3 == 1):
118           builder3 = Builder(secret=seed)
119           builder3.append_payment_op(publickey_1, '3', 'XLM')
120           builder3.add_text_memo('GREEN to YELLOW: 3 XLM')
121           builder3.sign()
122           builder3.submit()
123         if (slot3 == 2):
124           builder3 = Builder(secret=seed)
```

```
125             builder3.append_payment_op(publickey_2, '3', 'XLM')
126             builder3.add_text_memo('GREEN to GREEN : 3 XLM')
127             builder3.sign()
128             builder3.submit()
129           if (slot3 == 3):
130             builder3 = Builder(secret=seed)
131             builder3.append_payment_op(publickey_3, '3', 'XLM')
132             builder3.add_text_memo('GREEN to RED   : 3 XLM')
133             builder3.sign()
134             builder3.submit()
135           if (slot3 == 4):
136             builder3 = Builder(secret=seed)
137             builder3.append_payment_op(publickey_4, '3', 'XLM')
138             builder3.add_text_memo('GREEN to BLUE  : 3 XLM')
139             builder3.sign()
140             builder3.submit()
141           time.sleep(5)
142           print("Slept 10 sec")
143           print("S1: " + str(slot1))
144           print("S2: " + str(slot2))
145           print("S3: " + str(slot3))

147     def onButton1(channel):
148       global slot1
149       if not(GPIO.input(channel)):
150         if (channel == 37):
151           img1 = img_y
152           slot1 = 1
153         if (channel == 38):
154           img1 = img_g
155           slot1 = 2
156         if (channel == 40):
157           img1 = img_r
158           slot1 = 3
159         if (channel == 35):
160           img1 = img_b
161           slot1 = 4

163         root.updateLabel1(img1)
164       else:
165         slot1 = 0
166         root.updateLabel1(img_e)

168     def onButton2(channel):
169       global slot2
170       if not(GPIO.input(channel)):
171         if (channel == 13):
172           img2 = img_y
173           slot2 = 1
174         if (channel == 7):
175           img2 = img_g
176           slot2 = 2
177         if (channel == 15):
178           img2 = img_r
179           slot2 = 3
180         if (channel == 11):
181           img2 = img_b
182           slot2 = 4
183         root.updateLabel2(img2)
184       else:
185         slot2 = 0
186         root.updateLabel2(img_e)

188     def onButton3(channel):
189       global slot3
190       if not(GPIO.input(channel)):
191         if (channel == 31):
192           img3 = img_y
193           slot3 = 1
194         if (channel == 29):
195           img3 = img_g
196           slot3 = 2
197         if (channel == 33):
```

```
198             img3 = img_r
199             slot3 = 3
200         if (channel == 32):
201             img3 = img_b
202             slot3 = 4
203         root.updateLabel3(img3)
204     else:
205         slot3 = 0
206         root.updateLabel3(img_e)


209 # setup GPOIs ----------------------------------------------------
210 GPIO.setmode(GPIO.BOARD)

212 GPIO.setup(40, GPIO.IN, pull_up_down=GPIO.PUD_UP)
213 GPIO.setup(35, GPIO.IN, pull_up_down=GPIO.PUD_UP)
214 GPIO.setup(37, GPIO.IN, pull_up_down=GPIO.PUD_UP)
215 GPIO.setup(38, GPIO.IN, pull_up_down=GPIO.PUD_UP)

217 GPIO.setup(29, GPIO.IN, pull_up_down=GPIO.PUD_UP)
218 GPIO.setup(31, GPIO.IN, pull_up_down=GPIO.PUD_UP)
219 GPIO.setup(33, GPIO.IN, pull_up_down=GPIO.PUD_UP)
220 GPIO.setup(32, GPIO.IN, pull_up_down=GPIO.PUD_UP)

222 GPIO.setup(7, GPIO.IN, pull_up_down=GPIO.PUD_UP)
223 GPIO.setup(11, GPIO.IN, pull_up_down=GPIO.PUD_UP)
224 GPIO.setup(13, GPIO.IN, pull_up_down=GPIO.PUD_UP)
225 GPIO.setup(15, GPIO.IN, pull_up_down=GPIO.PUD_UP)

227 # add event handlers ----------------------------------------------
228 GPIO.add_event_detect(40, GPIO.BOTH, callback=onButton1)
229 GPIO.add_event_detect(35, GPIO.BOTH, callback=onButton1)
230 GPIO.add_event_detect(37, GPIO.BOTH, callback=onButton1)
231 GPIO.add_event_detect(38, GPIO.BOTH, callback=onButton1)

233 GPIO.add_event_detect(29, GPIO.BOTH, callback=onButton3)
234 GPIO.add_event_detect(31, GPIO.BOTH, callback=onButton3)
235 GPIO.add_event_detect(33, GPIO.BOTH, callback=onButton3)
236 GPIO.add_event_detect(32, GPIO.BOTH, callback=onButton3)

238 GPIO.add_event_detect(7, GPIO.BOTH, callback=onButton2)
239 GPIO.add_event_detect(11, GPIO.BOTH, callback=onButton2)
240 GPIO.add_event_detect(13, GPIO.BOTH, callback=onButton2)
241 GPIO.add_event_detect(15, GPIO.BOTH, callback=onButton2)

243 # generate keypair from seed --------------------------------------
244 kp = Keypair.from_seed(seed)

246 # decode publickey from keypair -----------------------------------
247 publickey = kp.address().decode()

249 # create rootpanel ------------------------------------------------
250 root = Fullscreen()

252 img_e = ImageTk.PhotoImage(Image.open("empty.gif"))
253 img_r = ImageTk.PhotoImage(Image.open("red.gif"))
254 img_g = ImageTk.PhotoImage(Image.open("green.gif"))
255 img_b = ImageTk.PhotoImage(Image.open("blue.gif"))
256 img_y = ImageTk.PhotoImage(Image.open("yellow.gif"))

258 root.updateLabel1(img_e)
259 root.updateLabel2(img_e)
260 root.updateLabel3(img_e)

262 sendthread = Sendthread()
263 sendthread.start()

265 root.tk.mainloop()
```

Listing A.1: Quellcode: Die Haus-Software

# A.2. Der Guthaben-Watchdog

```
1  #————————————————————
2  #———— Watchdog ————
3  #————————————————————
4  # Author: Thorsten Knoll
5  # Date:    28.11.2017
6  #————————————————————

8  from stellar_base.address import Address
9  import time
10 import os

12 # Public Keys
13 n1_pubkey = 'GDMB...'
14 n2_pubkey = 'GC2A...'
15 n3_pubkey = 'GDW7...'
16 n4_pubkey = 'GDPV...'

18 n1_address = Address(address=n1_pubkey)
19 n2_address = Address(address=n2_pubkey)
20 n3_address = Address(address=n3_pubkey)
21 n4_address = Address(address=n4_pubkey)

23 while True:
24     n1_address.get()
25     n2_address.get()
26     n3_address.get()
27     n4_address.get()

29     os.system('clear')
30     print ('Neighbor GREEN : ' + str(n1_address.balances[0]["balance"]) + ' XLM')
31     print ('Neighbor YELLOW: ' + str(n2_address.balances[0]["balance"]) + ' XLM')
32     print ('Neighbor RED   : ' + str(n3_address.balances[0]["balance"]) + ' XLM')
33     print ('Neighbor BLUE  : ' + str(n4_address.balances[0]["balance"]) + ' XLM')

35     time.sleep(2)
```

Listing A.2: Quellcode: Der Guthaben-Watchdog

# Abbildungsverzeichnis

# Stichwortverzeichnis

## N



## O



## P



## Q



## R



## S



## T



## U



## V



## Z

# Literaturverzeichnis


[Ant15] Andreas M. Antonopoulos. *Mastering Bitcoin : [unlocking digital cryptocurrencies]*. 1. Aufl. Beijing [u.a.], 2015. ISBN: 9781449374044.

[Com] Lightning Network Community. *BOLT: Basis of Lightning Technology*. `https://github.com/lightningnetwork/lightning-rfc/blob/master/00-introduction.md`.

[Dan18] Jordan Daniell. *Ethereum Foundation's Nick Johnson On GWei Transaction Fees*. `https://www.ethnews.com/ethereum-foundations-nick-johnson-on-gwei-transaction-fees`. 2018.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN: 0-201-89684-2.

[Küh17] Eike Kühl. *Mein Leben als Kryptokatzenzüchter*. `http://www.zeit.de/digital/internet/2017-12/cryptokitties-blockchain-ethereum-bitcoin`. 2017.

[LeM] Colin LeMahieu. *RaiBlocks: Ein gebührenfreies verteiltes Kryptowährungsnetzwerk*. `https://raiblocks.net/media/RaiBlocks_Whitepaper__German.pdf`.

[Mad18] Antonio Madeira. *What is the Ethereum Ice Age?* `https://www.cryptocompare.com/coins/guides/what-is-the-ethereum-ice-age/`. 2018.

[Maz15] David Mazières. *The Stellar Consensus Protocol, Talks at Google*. `https://www.youtube.com/watch?v=vmwnhZmEZjc`. 2015.

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[Nak09] Satoshi Nakamoto. *Bitcoin Client 0.1.0*. http://www.metzdowd.com/pipermail/cryptography/2009-January/014994.html. 2009.

[PD16] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. https://lightning.network/lightning-network-paper.pdf. Jan. 2016.

[Ste15] Stellar.org. *Comic: Adventures in galactic consensus*. https://www.stellar.org/stories/adventures-in-galactic-consensus-chapter-1. 2015.

[Tea17] Ethereum Team. *Byzantium HF Announcement*. https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/. 2017.