Bachelorarbeit

im Studiengang
Wirtschaftsinformatik und digitale Medien

Ein sicheres Peer-to-Peer-Instant-Messaging-Protokoll unter Berücksichtigung von Blockchain-Technologie

vorgelegt von

Nicole Sauer

an der Hochschule der Medien Stuttgart
am 29.01.2024
zur Erlangung des akademischen Grades eines
Bachelor of Science

Erstprüfer	Prof. Dr. Peter Thies
Zweitprüfer	Prof. Dr. Stephan Wilczek

Ehrenwörtliche Erklärung

Hiermit versichere ich, Nicole Sauer, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Ein sicheres Peer-to-Peer-Instant-Messaging-Protokoll unter Berücksichtigung von Blockchain-Technologie“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Ebenso sind alle Stellen, die mit Hilfe eines KI-basierten Schreibwerkzeugs erstellt oder überarbeitet wurden, kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO), der HdM einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Leutenbach, den 29.01.2024 

Ort, Datum	Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Architektur des Protokolls

In diesem Kapitel wird die Architektur des Protokolls eingehend erläutert. Dabei werden zunächst die fundamentalen Grundlagen des Protokolls beleuchtet, gefolgt von einer Betrachtung zur Identifikation der beteiligten Teilnehmer. Anschließend wird das Verbindungsmanagement detailliert beschrieben, gefolgt von einer Darstellung des Nachrichtenformats. Abschließend erfolgt eine umfassende Erklärung zur Integration der Blockchain.

1.1 Grundlagen des Protokolls

Dieses Kapitel beschreibt die grundlegenden Konzepte und Technologien, die für das Protokoll verwendet werden. Zunächst wird die Peer-to-Peer Funktionalität erläutert, gefolgt von einer Beschreibung des ICE-Protokolls. Anschließend wird die Blockchain-Technologie erläutert, gefolgt von einer Beschreibung der verwendeten Kryptographie.

Peer-to-Peer ist ein dezentrales Kommunikationsmodell, bei dem alle Teilnehmer gleichberechtigt sind und sowohl als Client als auch als Server fungieren können. Im Gegensatz zu einem Client-Server-Modell, bei dem ein zentraler Server die Kommunikation zwischen den Teilnehmern verwaltet, kommunizieren die Teilnehmer in einem Peer-to-Peer-Netzwerk direkt miteinander. Dieses Modell bietet eine Reihe von Vorteilen, wie z.B. eine höhere Skalierbarkeit, da die Last auf mehrere Teilnehmer verteilt wird, und eine höhere Ausfallsicherheit, da das Netzwerk nicht von einem zentralen Server abhängig ist. Das Verwenden einer zentralen Instanz kann zu einem Single Point of Failure führen, der das gesamte Netzwerk beeinträchtigen kann, wenn er ausfällt. Doch es hat auch Vorteile, ein zentrales System zu verwenden, da es einfacher ist, die Teilnehmer zu identifizieren, da es eine zentrale Instanz gibt, die die Teilnehmer verwaltet. Dafür sind Peer-to-Peer-Netzwerke schwieriger zu zensieren, da es keine zentrale Instanz gibt, die

zensiert werden kann. Ein Nachteil des Peer-to-Peer-Modells ist jedoch, dass es in Abwesenheit eines Servers schwieriger ist, Teilnehmer zu finden oder sie zu verwalten. Dieses Problem kann durch die Verwendung eines strukturierten Peer-to-Peer-Netzwerks gelöst werden, das eine explizite Organisationsstruktur für die Teilnehmer bietet, um die Suche und Verwaltung zu erleichtern (Luntovskyy und Gütter 2020).

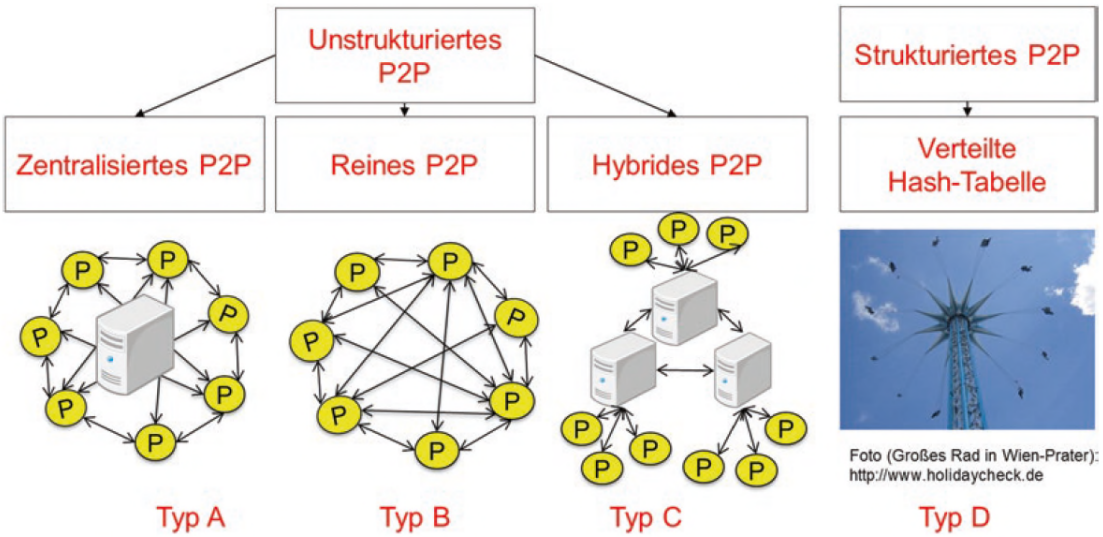

Abbildung 1.1: Typen von Peer-to-Peer-Netzwerken (Luntovskyy und Gütter 2020)

Unstrukturierte, strukturierte und hybride Peer-to-Peer-Netzwerke sind unterschiedliche Ansätze zur Organisation von Knoten und Ressourcen in dezentralen Netzwerken. Unstrukturierte Netzwerke sind charakterisiert durch ihre fehlende explizite Organisationsstruktur, was eine einfache Konnektivität ermöglicht. Diese Netzwerke sind dynamisch und erlauben es Knoten frei beizutreten oder das Netzwerk zu verlassen, ohne die Gesamtstruktur signifikant zu beeinflussen. Die Suche nach Ressourcen oder Informationen erfolgt oft durch Broadcasts oder zufällige Weiterleitungen, was jedoch zu ineffizienten Suchprozessen führen kann, da keine klare Routing-Struktur vorhanden ist. Ein klassisches Beispiel für ein unstrukturiertes Netzwerk ist das Gnutella-Netzwerk, das sich durch seine dezentrale Natur auszeichnet, jedoch bei der effizienten Ressourcenlokalisierung Schwierigkeiten aufgrund des fehlenden organisierten Routings hat [QUELLE].

Strukturierte Peer-to-Peer-Netzwerke hingegen weisen klare Regeln und Algorithmen zur Organisation der Knoten auf. Diese Netzwerke verfügen über eine explizite Organisationsstruktur, sei es eine Ringstruktur, k-bucket basierte Systeme oder andere, die es ermöglichen, effizientes Routing und eine optimierte Ressourcenverwaltung zu erreichen. Durch diese klar definierte Struktur sind strukturierte Netzwerke oft stabiler und

bieten eine effizientere Ressourcenlokalisierung im Vergleich zu ihren unstrukturierten Gegenstücken. Allerdings kann diese Stabilität auf Kosten von Flexibilität und Anpassungsfähigkeit gehen, da Änderungen in der Netzwerktopologie oder hohe Dynamik der Knoten schwerer zu handhaben sind [QUELLE].

Hybride Peer-to-Peer-Netzwerke versuchen, das Beste aus beiden Welten zu vereinen, indem sie Elemente aus strukturierten und unstrukturierten Ansätzen kombinieren. Diese Netzwerke integrieren Aspekte einer festen Organisationsstruktur für bestimmte Netzwerkbereiche, während andere Bereiche eher unstrukturiert sind. Als Beispiel hierfür kann Napster genannt werden, das eine oder mehrere Nodes dazu verwendet, um die Inhalte innerhalb des Netzwerks zu indizieren und zu verwalten, während der tatsächliche Download des Inhalts über eine direkte Verbindung zwischen den Teilnehmern erfolgt (Yang und Garcia-Molina 2001). Ziel ist es, Flexibilität und Effizienz zu optimieren und gleichzeitig eine gewisse Stabilität zu gewährleisten. Diese hybriden Ansätze streben danach, eine ausgewogene Lösung zu bieten, die sowohl die Anforderungen an eine stabile Struktur als auch an eine dynamische und flexible Umgebung erfüllt, je nach den spezifischen Bedürfnissen und Anwendungsfällen [QUELLE].

Um die Peer-to-Peer Funktionalität für das Protokoll dieser Arbeit zu gewährleisten, wird ein bereits existierendes Netzwerkprotokoll verwendet. In die engere Auswahl kamen Chord und Kademlia, welche beide lange Gegenstand intensiver Forschung waren, sowohl in der Industrie als auch in der akademischen Welt (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015, S. 808). Das Chord-Protokoll und das Kademlia-Protokoll sind zwei grundlegend verschiedene Ansätze zur Organisation von Peer-to-Peer-Netzwerken. Beide Protokolle sind strukturiert und bieten eine effiziente Ressourcenlokalisierung, aber sie unterscheiden sich in ihrer Routing-Struktur und der Art und Weise, wie sie die Knoteninformationen verwalten.

Chord basiert auf einer Ringstruktur (siehe Abbildung 1.2), bei der die Knoten in einem Ring angeordnet sind und jeder Knoten für einen bestimmten Schlüsselbereich verantwortlich ist. Die Verbindungen zwischen den Knoten sind durch ihren Platz im Ring definiert, wobei jeder Knoten eine Verbindung zu seinem nächsten Nachbarn im Uhrzeigersinn hat. Bei der Suche nach einem bestimmten Schlüssel durchläuft eine Anfrage einen logarithmischen Pfad im Ring, wobei die Knoten auf dem Weg begrenzte Informationen über andere Knoten behalten, um Anfragen weiterzuleiten. Dieses Modell ist recht einfach und effizient für viele Anwendungsfälle, aber es könnte anfällig sein für Engpässe oder längere Suchzeiten, insbesondere wenn das Netzwerk dynamisch ist und sich die Konfiguration häufig ändert.

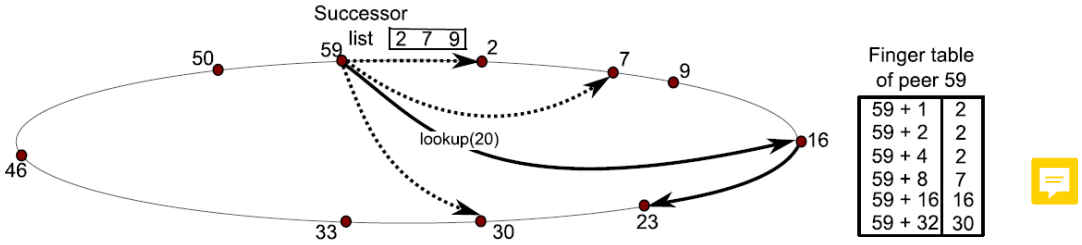

Abbildung 1.2: Visualisierung der Ringstruktur von Chord (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015)

Im Gegensatz dazu verwendet Kademlia eine K-Bucket-Struktur, die in Abbildung 1.3 zu sehen ist, um eine effiziente Verwaltung von Knoteninformationen zu ermöglichen. Die K-Buckets enthalten eine Liste von Knoten für verschiedene Schlüsselbereiche basierend auf ihrer Nähe, die durch XOR-Distanzen der IDs berechnet wird. Die Verbindungen zwischen den Knoten sind asymmetrisch, und jeder Knoten speichert Informationen über andere Knoten in seinen K-Buckets. Bei der Suche nach einem bestimmten Schlüssel erfolgt das Routing durch die XOR-Entfernung, wodurch die nächsten Knoten für diesen Schlüssel gefunden werden. Dieses Verfahren ermöglicht ebenfalls eine logarithmische Anzahl von Schritten für die Suche und bietet eine robuste Struktur, die gut mit dynamischen Netzwerkänderungen umgehen kann.

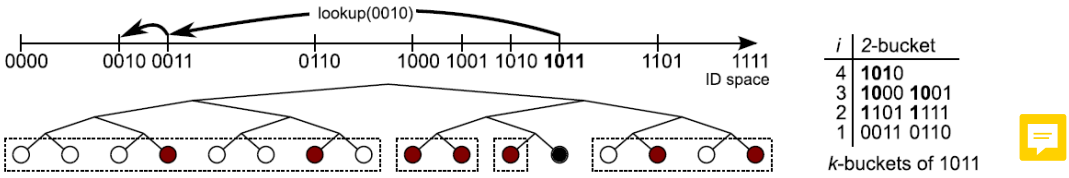

Abbildung 1.3: Visualisierung der Baumstruktur von Kademlia (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015)

Da bei einem Instant-Messaging-Protokoll häufig Teilnehmer das Netzwerk verlassen und neue Teilnehmer dem Netzwerk beitreten, ist es wichtig, dass das Protokoll mit hoher Fluktuation umgehen kann. Diese Fluktuation von Nodes wird als Churn (engl. Abwanderung) bezeichnet. In einer Studie von Medrano-Chávez et al. (Medrano-Chávez, Perez-Cortès und Lopez-Guerrero 2015), welche im hybriden Journal *Peer-to-Peer Networking and Applications* veröffentlicht wurde, wurde die Leistung von Chord und Kademlia in Bezug auf Netzwerkfluktuation untersucht. Die Ergebnisse zeigen, dass Kademlia bei hoher Fluktuation besser abschneidet als Chord. Aus diesem Grund wird Kademlia in

diesem Protokoll als Grundlage für das Auffinden von Teilnehmern und das Routing verwendet.

Um die Problematik mit Firewalls und NAT-Gateways zu lösen, wird das ICE-Protokoll verwendet. ICE ist ein Framework, das mehrere Techniken kombiniert, um eine Verbindung zwischen zwei Endpunkten herzustellen, die sich hinter NATs oder Firewalls befinden. Genaue Details zur Implementierung von ICE folgt in Kapitel 1.3 *Verbindungsmanagement*.

1.2 Identifikation von Teilnehmern

Da es in Peer-to-Peer-Netzwerken keinen zentralen Server gibt, der unter anderem zur Auffindung und Identifikation anderer Teilnehmer verwendet werden kann, müssen die Teilnehmer auf andere Weise identifiziert werden. Durch die Entscheidung, das Kademlia Protokoll zu implementieren, wird die ID des Teilnehmers als Schlüssel für die Speicherung der Teilnehmerinformationen, wie IP-Adresse und Port, verwendet.

Aus der Spezifikation von Kademlia geht hervor, dass die ID einer Node, welche auch als *Kademlia-ID* bezeichnet wird, 160 Bit lang sein muss und aus einer zufälligen Kennung bestehen soll (Maymounkov und Mazières 2002, S. 2). Für das hier entwickelte Protokoll wird der Benutzername des Teilnehmers als ID verwendet. Da Kademlia eine ID mit einer Länge von 160 Bit erwartet, muss der Benutzername auf diese Länge gebracht werden. Dafür wird eine Zeichenbegrenzung von 20 Zeichen festgelegt, was bei einer UTF-8 Codierung einer Bitlänge von 160 Bit entspricht (Yergeau 2003). Falls der Benutzername kürzer als 20 Zeichen ist, wird er mit Nullen aufgefüllt. Der Benutzername muss, wie aus den funktionalen Anforderungen (siehe 1.6.2) zu entnehmen ist, eindeutig sein und kann vom Benutzer bei der Registrierung frei gewählt werden. Durch die Beschränkung der ID auf 20 Zeichen wird die Anzahl der Teilnehmer zwar begrenzt, jedoch ist die Anzahl der möglichen Teilnehmer immer noch sehr groß. Diese Anzahl ist so groß, dass sie in der Praxis nicht erreicht werden sollte und somit die Beschränkung der ID keine Auswirkungen auf die Funktionalität des Protokolls haben sollte. In der Node werden dann Schlüssel-Wert-Paare gespeichert, wobei der Schlüssel die ID des Teilnehmers ist und der Wert die IP-Adresse und der Port. Es wird festgelegt, dass immer Port 49152 verwendet wird, da sich dieser Port im Bereich der dynamischen Ports befindet und somit nicht für andere Anwendungen reserviert ist (Cotton u. a. 2011, S. 20).

Wenn ein Teilnehmer eine Nachricht an einen anderen Teilnehmer senden möchte, muss er dessen IP-Adresse kennen. Um an diese Information zu gelangen, muss zuerst die ID des Teilnehmers in der Blockchain gesucht werden und anschließend eine Peer-

Discovery durchgeführt werden, um die IP-Adresse des Teilnehmers zu erhalten. Eine Peer-Discovery läuft wie folgt ab:

1. Der Teilnehmer, der die Nachricht senden möchte, sendet eine *FIND_NODE*-Nachricht an den Teilnehmer, dessen IP-Adresse er sucht. In dieser Nachricht ist die ID des Teilnehmers enthalten, dessen IP-Adresse gesucht wird.

2. Der Teilnehmer, der die Nachricht erhalten hat, sucht in seiner Routing-Tabelle nach dem Teilnehmer, dessen ID in der Nachricht enthalten ist. Falls der Teilnehmer gefunden wird, wird eine Nachricht an den Teilnehmer gesendet, der die *FIND_NODE*-Nachricht gesendet hat. In dieser Nachricht ist die IP-Adresse des Teilnehmers enthalten.

3. Falls der Teilnehmer nicht in der Routing-Tabelle gefunden wird, wird die *FIND_NODE*-Nachricht an den Teilnehmer weitergeleitet, der der ID des gesuchten Teilnehmers am nächsten ist. Dieser Vorgang wird solange wiederholt, bis der Teilnehmer gefunden wird.

4. Falls der Teilnehmer nicht gefunden wird, wird eine *NODE_NOT_FOUND*-Nachricht an den Teilnehmer gesendet, der die *FIND_NODE*-Nachricht gesendet hat.

5. Der Teilnehmer, der die Nachricht mit der IP-Adresse seines Ziels erhalten hat, speichert diese in seiner Routing-Tabelle ab.

Um das Netzwerk immer aktuell zu halten, wird in regelmäßigen Abständen ein *PING* an alle Teilnehmer gesendet, die in den Routing-Tabellen der Nodes gespeichert sind. Falls ein Teilnehmer nicht antwortet, wird er aus der Routing-Tabelle entfernt.

1.3 Verbindungsmanagement

Das Verbindungsmanagement ist ein essentieller Bestandteil des Protokolls, da es die Grundlage für die Kommunikation zwischen den Teilnehmern bildet. Es ist dafür verantwortlich, dass die Nachrichtenübertragung zwischen den Teilnehmern funktioniert. Dazu gehört der Verbindungsaufbau, die Nachrichtenübertragung und der Verbindungsabbau, welche im Folgenden detailliert beschrieben werden.

1.3.1 Verbindungsaufbau

Bei der Herstellung einer Verbindung zwischen zwei Teilnehmern sind die IP-Adresse und der Port des Zielgeräts entscheidend. Der Port ist durch die Definition in Abschnitt 1.2 festgelegt. Die IP-Adresse des Zielgeräts ist jedoch nicht bekannt. Um diese zu ermitteln, wird das Kademlia-Protokoll verwendet. Sobald die IP-Adresse und der Port des Ziels über Kademlia ermittelt wurden, wird versucht, eine direkte Verbindung herzustellen. Die Präferenz liegt hierbei auf der direkten Verbindung. Ein UDP-Paket wird an die ermittelte Zieladresse gesendet. Es wird erwartet, dass das Ziel innerhalb eines festgelegten Zeitfensters von zwei Sekunden antwortet. Falls eine Antwort innerhalb dieses Zeitrahmens eingeht, wird die Kommunikation über diesen direkten Weg aufgebaut. Falls keine Antwort eintrifft, gibt es eine alternative Methode: das Interactive Connectivity Establishment (ICE) Protokoll. Wenn also keine Antwort vom Ziel über das Kademlia-Netzwerk erhalten wird oder die direkte Verbindung scheitert, tritt das ICE-Protokoll in Kraft. Ursprünglich war die Idee, dass bei einer fehlgeschlagenen direkten Verbindung ein Relay-Server mit dem TURN-Protokoll verwendet werden soll. Doch aus nachfolgendem Zitat ist zu entnehmen, dass TURN nicht die beste Lösung ist, da nicht definiert ist, wie die *relayed transport address*, mit deren Hilfe die Kommunikation über den Relay-Server aufgebaut werden soll, an die Teilnehmer kommuniziert werden soll.

> *A client using TURN must have some way to communicate the relayed transport address to its peers and to learn each peer's IP address and port (more precisely, each peer's server-reflexive transport address; see Section 3). How this is done is out of the scope of the TURN protocol. One way this might be done is for the client and peers to exchange email messages. Another way is for the client and its peers to use a special-purpose „introduction" or „rendezvous" protocol (see RFC 5128 for more details).* (Reddy u. a. 2020, S. 7)

Deshalb wurde sich, wenn die direkte Verbindung nicht funktioniert, für die Verwendung des ICE-Protokolls entschieden, da es für dieses Problem eine Lösung bietet. ICE ist ein Protokoll, das die Verbindungseinrichtung zwischen zwei Geräten ermöglicht, die sich hinter NATs (Network Address Translation) oder Firewalls befinden (Keränen, Holmberg und Rosenberg 2018, S. 6 ff.). Es ist ein Protokoll, das in der Lage ist, sich an sich ändernde Netzwerkbedingungen anzupassen und verschiedene Arten von Verbindungen zu unterstützen, wie beispielsweise direkte Verbindungen, Verbindungen über STUN (Session Traversal Utilities for NAT) oder Verbindungen über TURN (Traversal Using Relays around NAT) (Keränen, Holmberg und Rosenberg 2018, S. 1).

Um eine Verbindung zwischen zwei Geräten aufzubauen, durchläuft ICE drei Phasen: die Sammlung und der Austausch von Verbindungsadressen, die Konnektivitätsprüfung und die Auswahl der am besten geeigneten Verbindungsadresse für die Kommunikation (Keränen, Holmberg und Rosenberg 2018, S. 7 ff.).

Die erste Phase beinhaltet das Sammeln verschiedener potenzieller Verbindungsadressen (auch *Kandidaten* genannt (Keränen, Holmberg und Rosenberg 2018, S. 8)), die für eine zuverlässige Kommunikation benötigt werden, insbesondere wenn einer oder beide Teilnehmer sich hinter NATs oder Firewalls befinden. Zuallererst werden die lokalen oder *Host*-Adressen der beteiligten Geräte berücksichtigt. Diese Adressen repräsentieren die standardmäßigen IP-Adressen und Ports der Geräte innerhalb ihres lokalen Netzwerks. Sie dienen als potenzielle direkte Verbindungswege zwischen den Geräten, falls sie sich im gleichen Netzwerk oder Subnetz befinden. Zusätzlich zu den lokalen Adressen werden reflexive Adressen über STUN (Session Traversal Utilities for NAT) ermittelt. STUN ermöglicht es einem Gerät, seine eigene externe IP-Adresse und den entsprechenden Port zu identifizieren, wie sie von einem NAT-Gerät reflektiert werden (Petit-Huguenin u. a. 2020, S. 4). Diese reflexiven Adressen stellen die externe Sichtbarkeit des Geräts aus der Perspektive des NAT-Geräts dar und helfen bei der Umgehung von NAT-Beschränkungen für den direkten Verbindungsaufbau. Sollte es jedoch aufgrund von restriktiven NAT-Konfigurationen nicht möglich sein, eine direkte Verbindung aufzubauen, kommen Relay-Adressen durch TURN (Traversal Using Relays around NAT) ins Spiel. Durch TURN werden Relay-Server genutzt, um den Datenverkehr zwischen den Geräten zu vermitteln (Reddy u. a. 2020, S. 10 f.). Diese Weiterleitungsadressen dienen als alternative Verbindungsmethode, indem sie den Datenverkehr über den Relay-Server leiten und so die Hindernisse von restriktiven NATs oder Firewalls umgehen. Die kombinierte Nutzung dieser verschiedenen Arten von potenziellen Verbindungsadressen – von lokalen, reflexiven bis hin zu Relay-Adressen – ermöglicht eine Vielzahl von Optionen für die Verbindungseinrichtung zwischen den Geräten. Diese Vielfalt an Adressen gewährleistet, dass selbst in komplexen Netzwerkszenarien wie NATs oder Firewalls verschiedene Wege für eine zuverlässige Kommunikation vorhanden sind. Nachdem alle potenziellen Verbindungsadressen gesammelt wurden, werden sie an den anderen Teilnehmer gesendet. Dieser Prozess wird als *Kandidatenaustausch* bezeichnet und erfolgt unter Verwendung eines Signaling Servers. Der Signaling Server ist für den Austausch der potenziellen Verbindungsadressen zwischen den Teilnehmern zuständig. Aus der Definition von ICE geht nicht hervor, wie die Verbindungsadressen zwischen den Teilnehmern (in ICE als *Agents* bezeichnet) ausgetauscht werden sollen. Wie aus dem nachstehenden Zitat zu verstehen ist, nimmt das Protokoll an, dass die Agents dazu in der Lage sind, doch wie genau das geschehen soll,

wird nicht definiert (Keränen, Holmberg und Rosenberg 2018, S. 7 ff.).

> *In a typical ICE deployment, there are two endpoints (ICE agents) that want to communicate. [...]. ICE assumes that the agents are able to establish a signaling connection between each other.* (Keränen, Holmberg und Rosenberg 2018, S. 7)

Eine eigene Definition für den Austausch der Verbindungsadressen zwischen den Teilnehmern über einen Signaling Server zu erstellen, würde den Rahmen dieser Arbeit übersteigen. Es sei aber gesagt, dass der Server die Benutzernamen der Teilnehmer kennt und die Verbindungsadressen der Teilnehmer anhand der Benutzernamen an die jeweiligen Teilnehmer sendet. Ein Beispiel (Diagramm drin lassen?):

- Alice möchte eine Nachricht an Bob senden
- Alice kennt den Benutzernamen von Bob
- Alice sendet eine Nachricht an den Signaling Server mit dem Benutzernamen von Bob
- Bob sendet seine Verbindungsadressen über den Signaling Server an Alice
- Alice sendet ihre Verbindungsadressen über den Signaling Server an Bob
- Alice und Bob haben nun die Verbindungsadressen des jeweils anderen

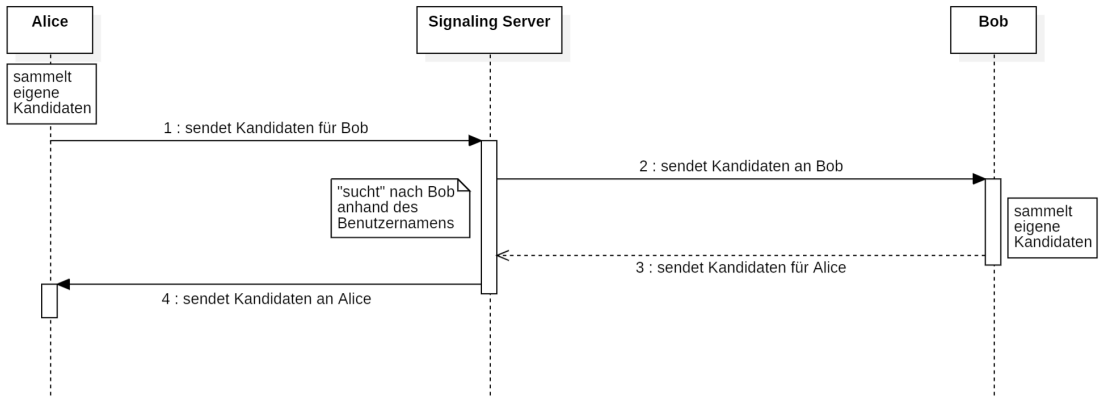

Abbildung 1.4: Kandidatenaustausch über einen Signaling Server

Man könnte außerdem argumentieren, dass Kademlia durch die Verwendung von ICE ersetzt wird, da auch ICE die lokale IP-Adresse und den Port des Empfängers ermittelt.

Doch Kademlia wird bewusst weiterhin verwendet, da es die Auslastung vom Signaling Server so gering wie möglich hält, indem der erste Versuch des Verbindungsaufbaus immer über Kademlia erfolgt. Erst wenn dieser fehlschlägt, wird das ICE-Protokoll verwendet, wodurch die Anzahl der Verbindungsversuche über den Signaling Server minimiert wird.

TODO: UML-Diagramm für Verbindungsaufbau erstellen

In Phase zwei werden Konnektivitätsprüfungen durchgeführt. Diese Prüfungen dienen dazu, die Eignung und Zuverlässigkeit der gesammelten Verbindungswege zwischen den Geräten zu bewerten, um die bestmögliche Verbindung für eine erfolgreiche Kommunikation zu identifizieren. Die Reihenfolge der Konnektivitätsprüfungen wird durch einen Prioritätsalgorithmus bestimmt, der die Verbindungsadressen nach ihrer Priorität ordnet. Aus der Dokumentation von ICE geht hervor, dass die Priorität einer Verbindungsadresse durch die folgende Formel berechnet wird (Keränen, Holmberg und Rosenberg 2018, S. 22):

$$\text{priority} = 2^{24} \cdot (\text{type preference}) + 2^{8} \cdot (\text{local preference}) + 2^{0} \cdot (256 - \text{component ID}) \quad (1.1)$$

Jeder Kandidat hat eine Priorität, die durch die Formel 1.1 berechnet wird. Die Priorität wird durch die drei Werte in der Formel bestimmt: Typpräferenz, lokale Präferenz und Komponenten-ID. Die Typpräferenz ist ein Wert, der die Art der Verbindungsadresse angibt, und dessen Wert sich zwischen 0 und 126 befinden muss. Es gibt, wie bereits in Phase eins aufgezeigt, drei verschiedene Typen von Verbindungsadressen: Host-Adressen, Peer-reflexive Adressen und Relay-Adressen. Die Werte für die Typpräferenz, die in der Dokumentation von ICE für die Berechnung der Priorität empfohlen werden, sind die folgenden: 126 für Host-Adressen, 100 für Peer-reflexive Adressen und 0 für Relay-Adressen. Wichtig ist, dass der Wert 0 für Relay-Adressen nicht bedeutet, dass Relay-Adressen nicht verwendet werden sollten, sondern dass sie die niedrigste Priorität haben. Für das Protokoll dieser Arbeit werden diese Werte übernommen. Für die lokalen Präferenzen wird ein Wert von 65535 empfohlen, um die Verwendung von lokalen Adressen zu priorisieren. Auch dieser Wert wird übernommen. Der dritte und letzte Wert der Formel ist die Komponenten-ID, die dazu dient, verschiedene Datenströme oder Komponenten innerhalb eines einzelnen Kandidaten zu unterscheiden. Das bedeutet, dass ein Kandidat mehrere Komponenten haben kann, die jeweils eine eigene Komponenten-ID haben. Dadurch können beispielweise Bild-Daten, Audio-Daten und Text-Daten innerhalb eines einzelnen Kandidaten über verschiedene Komponenten gesendet werden. Da in dem Protokoll dieser Arbeit nur ein Datenstrom verwendet wird - und zwar Text - wird der

Komponenten-ID der Wert 1 zugewiesen.

Wenn alle Kandidaten ihre Priorität erhalten haben, werden sie nach ihrer Priorität sortiert und die Konnektivitätsprüfung beginnt. Dieser Prozess beinhaltet den Versuch, Verbindungen aufzubauen und Datenverkehr über verschiedene potenzielle Wege zu senden und zu empfangen. Durch das Senden von Probe-Paketen über jede potenzielle Verbindungsadresse wird geprüft, ob die Kommunikation erfolgreich erfolgen kann. Dabei werden die drei verschiedenen Arten von Adressen (lokale, reflexive und Relay-Adressen) verwendet, um verschiedene Möglichkeiten zu testen, wie die Geräte miteinander kommunizieren können. Die Probe-Pakete werden über UDP (User Datagram Protocol) gesendet, da es ein verbindungsloses Protokoll ist und somit vor dem Senden keine Verbindung aufgebaut werden muss (Postel 1980, S. 1). Dies ermöglicht es, die Erreichbarkeit der Verbindungsadressen zu testen, ohne eine Verbindung aufzubauen. Die Probe-Pakete werden an die Verbindungsadressen gesendet und die Antwort wird überwacht. Wenn eine Antwort empfangen wird, wird die Verbindungsadresse als erreichbar angesehen. Wenn keine Antwort empfangen wird, wird die Verbindungsadresse als nicht erreichbar angesehen. Die Probe-Pakete werden in regelmäßigen Abständen gesendet, um die Stabilität der Verbindungsadressen zu testen. Wenn die Probe-Pakete über einen längeren Zeitraum nicht beantwortet werden, wird die Verbindungsadresse als nicht stabil angesehen.

Mit dem Abschluss der Konnektivitätsprüfungen beginnt die dritte und letzte Phase. Mit den Ergebnissen werden Kandidaten-Paare gebildet, die die Verbindungsadressen der beiden Geräte enthalten. Während dieses Schritts werden die gesammelten Kandidaten (lokale IP-Adressen, Ports und durch STUN oder TURN-Server erhaltene Reflexionen) in verschiedenen Konstellationen kombiniert. Jede dieser Kombinationen bildet ein Paar, das als möglicher Kommunikationsweg zwischen den Geräten dienen könnte. Die gebildeten Paare werden auf Redundanz kontrolliert und dann nach ihrer Priorität sortiert, die aus den einzelnen Prioritäten der Kandidaten berechnet wird. Aus dieser Reihenfolge wird dann die sogenannte *Checkliste* erstellt, die die Paare in der Reihenfolge ihrer Priorität enthält. Auch die Kandidaten-Paare erhalten mit Hilfe einer Formel eine Priorität, die in der Dokumentation von ICE auf Seite 31 definiert ist (Keränen, Holmberg und Rosenberg 2018, S. 31). Anschließend wird damit begonnen, eine Verbindung über das Paar mit der höchsten Priorität aufzubauen. Wenn die Verbindung erfolgreich aufgebaut werden kann, wird die Kommunikation über dieses Paar fortgesetzt. Wenn die Verbindung nicht erfolgreich aufgebaut werden kann, wird das Paar mit der nächsthöheren Priorität ausgewählt und der Prozess wird wiederholt. Dieser Prozess wird fortgesetzt, bis eine Verbindung erfolgreich aufgebaut werden kann. Wenn keine Verbindung aufgebaut werden kann, wird die Kommunikation über das Paar mit der höchsten Priorität fortgesetzt,

das eine Verbindung über den Relay-Server verwendet.

Ein großer Vorteil von ICE ist die Flexibilität, die es ermöglicht, sich an sich ändernde Netzwerkbedingungen anzupassen. Falls sich während der Kommunikation Netzwerkparameter ändern - beispielsweise durch einen Wechsel zwischen Wi-Fi und Mobilfunknetzwerken - kann ICE dynamisch neue Kandidatenadressen identifizieren und die Verbindungen anpassen, ohne die laufende Kommunikation zu unterbrechen.

TODO: Vielleicht hier auch ein Diagramm?

1.3.2 Nachrichtenübertragung

Wenn die am besten geeignete Verbindungsadresse eine lokale Adresse ist, wird eine direkte Verbindung zwischen den Geräten aufgebaut. Wenn die am besten geeignete Verbindungsadresse eine reflexive Adresse ist, wird eine direkte Verbindung zwischen den Geräten aufgebaut, indem die reflexive Adresse als Zieladresse verwendet wird. Wenn wiederum die am besten geeignete Verbindungsadresse eine Relay-Adresse ist, wird eine Verbindung über den Relay-Server aufgebaut, indem die Relay-Adresse als Zieladresse verwendet wird. Der Nachrichtenaustausch erfolgt über die ausgewählte Verbindungsadresse. Wobei hier mit *Verbindung aufbauen* gemeint ist, dass die Nachrichten über die Verbindungsadresse gesendet und empfangen werden können. Aus Sicht der Transportschicht wird durch die Verwendung von UDP keine Verbindung aufgebaut, denn UDP ist ein verbindungsloses Protokoll. Das bedeutet, dass keine Verbindung aufgebaut werden muss, um die Nachrichten zu senden und zu empfangen. Die Nachrichten werden über die Verbindungsadresse gesendet und empfangen. Aus Applikationssicht wird jedoch eine Verbindung aufgebaut, da die Nachrichten zwischen den Teilnehmern ausgetauscht werden können. Ein Nachteil bei der Verwendung von UDP ist, dass die Nachrichten nicht zuverlässig zugestellt werden. Was bedeutet, dass Nachrichten verloren gehen können, ohne dass der Absender oder Empfänger davon erfährt. Das Transportprotokoll TCP (Transmission Control Protocol) hingegen ist zuverlässig, da es eine Verbindung aufbaut und sicherstellt, dass die Nachrichten erfolgreich zugestellt werden (Eddy 2022, S. 36). Im Falle von Paketverlusten ist auf der gleichen Seite definiert, dass TCP in der Lage ist, die verlorenen Pakete zu erkennen und erneut zu senden. Doch eine mögliche Verwendung von TCP hat auch Nachteile: ein Nachteil ist, dass es eine Verbindung aufbauen muss, bevor die Nachrichten gesendet werden können. Dies kann zu einer höheren Latenz führen. Ein weiterer Nachteil ist, dass die Verbindung aufrechterhalten werden muss, um die Nachrichten zu senden und zu empfangen. Dies führt zu einem höheren Ressourcenverbrauch, da die Verbindung aufrechterhalten werden muss, auch wenn keine Nachrichten

gesendet werden. Auch bei NATs hat TCP Nachteile. NATs schließen die Verbindungen nach einer gewissen Zeit, wenn keine Daten übertragen werden. Dies kann dazu führen, dass die Verbindung geschlossen werden, bevor die Nachrichten gesendet werden können. Bei Instant Messaging zählt die Geschwindigkeit, mit der die Nachrichten gesendet und empfangen werden mehr, als die Zuverlässigkeit der Nachrichten. Daher ist es besser, die Nachrichten schnell zu senden und zu empfangen, auch wenn dies bedeutet, dass die Nachrichten eventuell nicht zuverlässig zugestellt werden. Und auch in Verbindung mit NATs stellt sich UDP als die bessere Wahl heraus. Aus diesen Gründen wurde UDP als Transportprotokoll für die Nachrichtenübertragung gewählt.

1.3.3 Verbindungsabbau

Da es wie bereits erwähnt keine Verbindung gibt, die aufgebaut werden muss, um die Nachrichten zu senden und zu empfangen, gibt es auch keinen Verbindungsabbau. Die Nachrichtenübertragung kann jederzeit gestoppt werden, indem einfach keine Nachrichten mehr gesendet werden.

1.4 Nachrichtenformat

Jedes Instant-Messaging-Protokoll benötigt die Definition eines Nachrichtenformats, das die Struktur der Nachrichten festlegt, die zwischen den Teilnehmern ausgetauscht werden. Im Folgenden wird das Nachrichtenformat für das Instant-Messaging-Protokoll beschrieben.

1.4.1 Nachrichtenkopf

Nachdem eine Verbindung zwischen zwei Teilnehmern hergestellt werden konnte, kann die Nachricht vom Sender an den Empfänger gesendet werden. Der Nachrichtenkopf enthält die folgenden Informationen:

- Quell-IP-Adresse und Quell-Port
- Ziel-IP-Adresse und Ziel-Port
- Prüfsumme
- Timestamp
- Signatur

- Sequenznummer

Die Quell-IP-Adresse und der Quell-Port stammen vom Sender, die Ziel-IP-Adresse und der Ziel-Port enthalten die IP-Adresse und den Port des Empfängers. Die Prüfsumme wird aus dem Nachrichteninhalt berechnet und dient der Integritätsprüfung. Der Timestamp enthält die aktuelle Zeit, zu der die Nachricht gesendet wird. Jede Nachricht wird aus Sicherheitsgründen mit dem statischen privaten Schlüssel des Senders signiert und kann somit vom Empfänger verifiziert werden. Die Sequenznummer spielt bei der Verwendung von UDP eine wichtige Rolle, da UDP keine Garantie für die Reihenfolge der Nachrichten bietet. Somit kann der Empfänger die Nachrichten anhand der Sequenznummer in die richtige Reihenfolge bringen.

1.4.2 Nachrichteninhalt

Der Nachrichteninhalt enthält die folgenden Informationen:

- Nachrichtenlänge
- Nachrichteninhalt

Der Nachrichtentyp gibt an, um welche Art von Nachricht es sich handelt. Die Nachrichtenlänge gibt die Länge des Nachrichteninhalts an. Der Nachrichteninhalt enthält die eigentliche Nachricht, die vom Sender an den Empfänger gesendet wird.

1.5 Sicherheit

Quellenangaben, falls nicht schon in Grundlagen gemacht

Um die Kommunikation zwischen den Teilnehmern zu schützen, wird sowohl asymmetrische als auch symmetrische Verschlüsselung verwendet. Die asymmetrische Verschlüsselung wird für die Authentifizierung und die symmetrische Verschlüsselung für die Ende-zu-Ende-Verschlüsselung der Nachrichten verwendet. Die asymmetrische Verschlüsselung wird mit Hilfe von Public-Key-Kryptographie realisiert. In dem von Whitfield Diffie und Martin Hellman 1976 veröffentlichten Paper *New Directions in Cryptography* (Diffie und Hellman 1976) wurde die Public-Key-Kryptographie erstmals beschrieben. Darin wird die Problematik der symmetrischen Verschlüsselung beschrieben, dass ein Schlüssel für die Verschlüsselung und Entschlüsselung verwendet wird und dieser Schlüssel zu Beginn der Kommunikation über einen unsicheren Kanal zwischen den Teilnehmern ausgetauscht werden muss. Die Lösung dieses Problems ist die Public-Key-Kryptographie.

Bei der Public-Key-Kryptographie wird ein Schlüsselpaar generiert, das aus einem öffentlichen und einem privaten Schlüssel besteht. In diesem Protokoll wird bei der Registrierung eines Teilnehmers ein statisches Schlüsselpaar generiert. Der öffentliche Schlüssel dieses Schlüsselpaars wird gemeinsam mit dem Benutzernamen in der Blockchain hinterlegt und der private Schlüssel wird lokal auf dem Gerät des Teilnehmers gespeichert.

Um die Ende-zu-Ende-Verschlüsselung zu realisieren, wird ein Diffie-Hellman Schlüsselaustausch durchgeführt, der auf elliptischen Kurven basiert und daher auch Elliptic-Curve-Diffie-Hellman-Schlüsselaustausch (oder kurz ECDH) genannt wird. Als Grundlage für einen Schlüsselaustausch muss sich auf eine gemeinsame elliptische Kurvengleichung geeinigt werden(Wong 2023, S. 118). Für das hier entwickelte Protokoll wird die Funktion *X448* verwendet, die in RFC 7748 *The X25519 and X448 Elliptic Curves* (Langley, Hamburg und Turner 2016) spezifiziert ist.

Der Diffie-Hellman-Schlüsselaustausch basiert auf der Annahme, dass es keine effiziente Methode gibt, um den diskreten Logarithmus zu berechnen. Der Diffie-Hellman-Schlüsselaustausch basiert auf Gruppen, die aus einem Generator und Wahlweise einer Primzahl oder einem elliptischen Kurvenpunkt bestehen. Die Gruppen werden so gewählt, dass der diskrete Logarithmus in diesen Gruppen schwer zu berechnen ist. Der diskrete Logarithmus ist das inverse Element der Exponentialfunktion. Das bedeutet, dass der diskrete Logarithmus die Lösung der Gleichung $g^x = y$ ist, wobei g der Generator, x der diskrete Logarithmus und y das Ergebnis der Exponentialfunktion ist. Die Schwierigkeit des diskreten Logarithmus ist, dass es keine effiziente Methode gibt, um ihn zu berechnen. Die Sicherheit des Diffie-Hellman-Schlüsselaustauschs basiert auf der Annahme, dass es keine effiziente Methode gibt, um den diskreten Logarithmus zu berechnen (Wong 2023, S. 105-121).

Bei einem Verbindungsaufbau wird vom Sender ein neues Schlüsselpaar generiert, welches aus flüchtigen Schlüsseln besteht. Flüchtige Schlüssel werden in der Kryptographie auch ephemeral keys genannt und sind Schlüssel, die nur für eine kurze Zeit existieren. Der Sender generiert also einen flüchtigen privaten Schlüssel und einen flüchtigen öffentlichen Schlüssel. Der flüchtige öffentliche Schlüssel wird zusammen mit der ID des Senders und einem Zeitstempel in einer Nachricht, welche mit dem statischen privaten Schlüssel des Senders signiert wird, an den Empfänger gesendet. Der Empfänger kann die Nachricht mit dem öffentlichen Schlüssel des Senders, welcher mittels Smart Contract aus der Blockchain ausgelesen werden kann, entschlüsseln und somit die Authentizität des Senders verifizieren. Dadurch kann der Empfänger sicher sein, dass die Nachricht vom Sender stammt und nicht von einem Angreifer manipuliert wurde. Mit Hilfe des Zeitstempels kann der Empfänger außerdem feststellen, ob der flüchtige öffentliche Schlüssel

noch gültig ist. Der Empfänger extrahiert den flüchtigen öffentlichen Schlüssel aus der Nachricht des Senders und ist somit im Besitz des flüchtigen öffentlichen Schlüssels des Senders. Der Empfänger generiert ebenfalls einen flüchtigen privaten Schlüssel und einen flüchtigen öffentlichen Schlüssel. Der flüchtige öffentliche Schlüssel wird zusammen mit der ID des Empfängers und einem Zeitstempel in einer Nachricht, welche mit dem statischen privaten Schlüssel des Empfängers signiert wird, an den Sender gesendet. Der Sender kann die Nachricht mit dem öffentlichen Schlüssel des Empfängers entschlüsseln und somit die Authentizität des Empfängers verifizieren. Nun sind beide Teilnehmer im Besitz des flüchtigen öffentlichen Schlüssels des jeweils anderen Teilnehmers und können in Kombinationen mit ihrem eigenen flüchtigen privaten Schlüssel den gemeinsamen geheimen Schlüssel berechnen. Dieser gemeinsame geheime Schlüssel wird für die symmetrische Verschlüsselung der Nachrichten verwendet. Der gemeinsame geheime Schlüssel wird nicht über einen unsicheren Kanal übertragen, sondern von den Teilnehmern selbst berechnet. Dadurch ist es nicht möglich, dass ein Angreifer den gemeinsamen geheimen Schlüssel abfangen kann. Der gemeinsame geheime Schlüssel wird nur für eine kurze Zeit verwendet und danach verworfen. Für das hier entwickelte Protokoll wird die Gültigkeit des gemeinsamen geheimen Schlüssels auf 10 Minuten festgelegt. Nach Ablauf dieser Zeit muss erneut ein Diffie-Hellman-Schlüsselaustausch durchgeführt werden, um einen neuen gemeinsamen geheimen Schlüssel zu generieren. Falls also ein Angreifer irgendwie in den Besitz des gemeinsamen geheimen Schlüssels gelangt, kann er maximal 10 Minuten lang Nachrichten entschlüsseln. Nach Ablauf dieser Zeit ist der gemeinsame geheime Schlüssel ungültig und der Angreifer kann keine Nachrichten mehr entschlüsseln.

Diagramm erstellen

Durch die Verwendung von Public-Key-Kryptographie und Diffie-Hellman Schlüsselaustausch ist es möglich, dass sich die Teilnehmer gegenseitig authentifizieren können und einen gemeinsamen geheimen Schlüssel generieren können, ohne dass dieser über einen unsicheren Kanal übertragen werden muss. Dadurch ist es möglich, dass die Teilnehmer sicher miteinander kommunizieren können.

1.6 Integration von Blockchain

1.6.1 Auswahl der Blockchain

Für die Integration der Blockchain in das Protokoll wurde zunächst eine geeignete Blockchain gesucht. Dabei wurde sich auf die beiden bekanntesten Blockchains, Bitcoin und Ethereum, beschränkt. Da Bitcoin eine reine Kryptowährung ist und keine Smart Con-

tracts unterstützt, wurde sich für Ethereum entschieden. Ethereum ist eine Blockchain, die zwar auch eine Kryptowährung, Ether, besitzt, aber zusätzlich auch Smart Contracts unterstützt. Der Grund für die Existenz einer Währung auf der Blockchain ist, dass die Smart Contracts, die auf der Blockchain ausgeführt werden, mit Ether bezahlt werden müssen (Antonopoulos und Wood 2018, S. 2). Das bedeutet für die Teilnehmer des Protokolls, dass sie Ether besitzen müssen, um sich registrieren zu können. Für die zwei weiteren Funktionen, die die Blockchain im Protokoll übernimmt, ist keine Währung notwendig, da es sich um reine Lesezugriffe handelt, welche keine Kosten verursachen. Die beiden Funktionen sind die Suche nach einem Benutzernamen und die Suche nach einem öffentlichen Schlüssel anhand eines Benutzernamens. Diese beiden Funktionen werden in Abschnitt 1.6.2 (Registrierung) und Abschnitt 1.6.3 (Kommunikation) beschrieben. Die Blockchain wird also nur für die Registrierung der Teilnehmer und für die Suche nach Benutzernamen und öffentlichen Schlüsseln verwendet.

1.6.2 Registrierung

Um das Protokoll zu nutzen, muss sich jeder Nutzer zunächst auf der Blockchain registrieren. Dazu muss ein Smart Contract auf der Blockchain ausgeführt werden, der die Registrierung des Nutzers durchführt. Dieser Smart Contract wird mit dem Benutzernamen und dem statischen öffentlichen Schlüssel aufgerufen. Der statische öffentliche Schlüssel wird bei der Registrierung, bevor der Smart Contract angestoßen wird, festgelegt und kann nicht mehr geändert werden. Der Smart Contract erstellt einen neuen Eintrag in der Blockchain, der den Benutzernamen und den öffentlichen Schlüssel des Nutzers enthält. Um Dopplungen in der Blockchain zu vermeiden, wird der Benutzername nicht als Key in der Blockchain gespeichert, sondern als Value. Als Key wird die Adresse des Benutzers verwendet. Dadurch wird sichergestellt, dass jeder Benutzer nur einmal auf der Blockchain registriert werden kann. Diese Funktion wird nur einmalig bei der Registrierung aufgerufen. Sollte ein Nutzer seinen Benutzernamen ändern wollen, muss er sich mit dem neuen Benutzernamen und dem öffentlichen Schlüssel eines neu erzeugten statischen Schlüsselpaars erneut registrieren. Die Funktion im Smart Contract könnte wie folgt aussehen:

```
1 // Function to register a user
2 function registerUser(string memory _username,
3     bytes memory _publicKey) external {
4         // Check if the username and public key are not empty
5         require(bytes(_username).length > 0,
6             "Username cannot be empty");
```

```
7           require(_publicKey.length > 0,
8               "Public key cannot be empty");
9
10          // Check if the username is not already taken
11          require(usernames[_username] == address(0),
12              "Username is already taken");
13
14          // Check if the user is not already registered
15          require(!users[msg.sender].isRegistered, "User is already
                registered");
16
17          // Add the user to the users mapping with the user's
18          // username as the key
19          users[msg.sender] = User(_username, _publicKey, true);
20
21          // Store the username to address mapping
22          usernames[_username] = msg.sender;
23
24          // Emit the UserRegistered event with the user's address,
25          // username and public key
26          emit UserRegistered(msg.sender, _username, _publicKey);
27  }
```

Listing 1.1: Registrierung eines Nutzers auf der Blockchain

1.6.3 Kommunikation

Für die Kommunikation mit anderen Teilnehmern, benötigt man die ID des anderen Teilnehmers im Kademlia-Netzwerk. Da die ID im Kademlia-Netzwerk dem Benutzernamen entspricht, muss der Benutzername des anderen Teilnehmers bekannt sein. Dazu wird dieser Benutzername auf der Blockchain gesucht, um zu kontrollieren, ob dieser bereits registriert ist. Hierfür wird eine Funktion im Smart Contract auf der Blockchain aufgerufen, die den Benutzernamen als Parameter erhält. Der Smart Contract sucht dann in der Blockchain nach dem Benutzernamen und gibt ihn, falls vorhanden, zurück. Sollte kein Benutzer mit diesem Benutzernamen auf der Blockchain vorhanden sein, wird ein leerer String zurückgegeben, der darauf hinweist, dass der gesuchte Benutzer nicht registriert ist und somit keine Verbindung aufgebaut werden kann. Der Rückgabewert muss von der Applikation, die dieses Protokoll nutzt, abgefangen werden. Die Funktion könnte wie folgt aussehen:

```
// Function to verify a user's username
function verifyUsername(string memory _username) external view
    returns (string memory) {
        // Try to get the address associated with the username
        address userAddress = usernames[_username];

        // Check if the address is empty
        require(userAddress != address(0),
            "User does not exist.");

        // Get the user's details
        User storage user = users[userAddress];

        // Return the user's username
        return (user.username);
}
```

Listing 1.2: Suche nach einem Benutzernamen auf der Blockchain

Ist der Benutzername auf der Blockchain vorhanden und es kommt zum Verbindungsaufbau wie in Abschnitt 1.3 (Verbindungsmanagement) beschrieben, wird der öffentliche Schlüssel des Benutzers ausgelesen. Dafür wird die nachstehende Funktion genutzt:

```
// Function to get a user's public key
function getUserPublicKey(string memory _username) external view
    returns (bytes memory) {
        // Try to get the address associated with the username
        address userAddress = usernames[_username];

        // Check if the address is empty
        require(userAddress != address(0),
            "User does not exist.");

        // If the address is not empty, get the user
        // details using the address
        User storage user = users[userAddress];

        // Return the user's public key
        return (user.publicKey);
}
```

Listing 1.3: Suche nach einem öffentlichen Schlüssel auf der Blockchain

Um Redundanz im Code zu vermeiden, wäre es sinnvoll das Suchen nach der Adresse des übergebenen Benutzernamens, die Prüfung ob die Adresse leer ist und das Auslesen der Benutzerdaten aus dem Mapping in eine eigene Funktion auszulagern, welche nur von innerhalb des Smart Contracts aufgerufen werden kann. Aus Gründen des besseren Lese- und Verständnisflusses wurde dies hier nicht umgesetzt.

Durch das Speichern des öffentlichen Schlüssels auf der Blockchain kann jeder Teilnehmer die öffentlichen Schlüssel der anderen Teilnehmer finden und die Nachrichten, die er erhält, mit dem öffentlichen Schlüssel des Absenders verifizieren. Dadurch kann sichergestellt werden, dass die Nachrichten tatsächlich vom angegebenen Absender stammen und nicht von einem anderen Teilnehmer gesendet wurden, der sich als jemand anderes ausgibt.

Literatur

Antonopoulos, Andreas M. und Gavin Wood (2018). *Mastering Ethereum. Building Smart Contracts and DApps*. 1. Auflage. O´Reilly Media. ISBN: 978-1-491-97194-9.

Cotton, Michelle u. a. (Aug. 2011). *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*. RFC 6335. DOI: 10.17487/RFC6335. URL: https://www.rfc-editor.org/info/rfc6335.

Diffie, Whitfield und Martin Hellman (Nov. 1976). „New Directions in Cryptography". In: *IEEE Transactions on Information Theory* 22.6, S. 644–654. DOI: 10.1109/TIT.1976.1055638.

Eddy, Wesley (Aug. 2022). *Transmission Control Protocol (TCP)*. RFC 9293. DOI: 10.17487/RFC9293. URL: https://www.rfc-editor.org/info/rfc9293.

Keränen, Ari, Christer Holmberg und Jonathan Rosenberg (Juli 2018). *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal*. RFC 8445. DOI: 10.17487/RFC8445. URL: https://www.rfc-editor.org/info/rfc8445.

Langley, Adam, Mike Hamburg und Sean Turner (Jan. 2016). *Elliptic Curves for Security*. RFC 7748. DOI: 10.17487/RFC7748. URL: https://www.rfc-editor.org/info/rfc7748.

Luntovskyy, Andriy und Dietbert Gütter (2020). *Moderne Rechnernetze. Protokolle, Standards und Apps in kombinierten drahtgebundenen, mobilen und drahtlosen Netzwerken*. 1. Auflage. Springer Vieweg Wiesbaden. ISBN: 978-3-658-25617-3. DOI: https://doi.org/10.1007/978-3-658-25617-3.

Maymounkov, Petar und David Mazières (2002). *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*. URL: https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf.

Medrano-Chávez, Adan G., Elizabeth Perez-Cortès und Miguel Lopez-Guerrero (Sep. 2015). „A performance comparison of Chord and Kademlia DHTs in high churn scenarios“. In: *Peer-to-Peer Networking and Applications* 8.5. DOI: 10.1007/s12083-014-0294-y.

Petit-Huguenin, Marc u. a. (Feb. 2020). *Session Traversal Utilities for NAT (STUN)*. RFC 8489. DOI: 10.17487/RFC8489. URL: https://www.rfc-editor.org/info/rfc8489.

Postel, Jon (Aug. 1980). *User Datagram Protocol*. RFC 768. DOI: 10.17487/RFC0768. URL: https://www.rfc-editor.org/info/rfc768.

Reddy, Tirumaleswar u. a. (Feb. 2020). *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC 8656. DOI: 10.17487/RFC8656. URL: https://www.rfc-editor.org/info/rfc8656.

Wong, David (2023). *Kryptografie in der Praxis. Eine Einführung in die bewährten Tools, Frameworks und Protokolle*. 1. Auflage. dpunkt.verlag. ISBN: 978-3-86490-939-9.

Yang, Beverly und Hector Garcia-Molina (Jan. 2001). *Comparing Hybrid Peer-to-Peer Systems*. URL: http://infolab.stanford.edu/~byang/pubs/hybridp2p_long.pdf.

Yergeau, François (Nov. 2003). *UTF-8, a transformation format of ISO 10646*. RFC 3629. DOI: 10.17487/RFC3629. URL: https://www.rfc-editor.org/info/rfc3629.