Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Entwurf und Evaluierung einer sicheren DHT für dezentrales Voice-over-IP

Diplomarbeit am Institut für Telematik
Prof. Dr. Martina Zitterbart
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Sebastian Mies

Betreuer:
Prof. Dr. Martina Zitterbart
Dipl.-Inform. Ingmar Baumgart

Tag der Anmeldung: 1. April 2006
Tag der Abgabe: 30. September 2006

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 30. September 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Peer-to-Peer Systeme zeichnen sich gegenüber Client-Server Systemen durch ihre dezentrale Struktur aus und wurden insbesondere durch Filesharingbörsen bekannt. Der primäre Unterschied zu klassischen Client-Server Systemen ist, dass jeder Knoten im Netz gleichberechtigt ist und das Netz durch die Interessengemeinschaft von vielen Nutzern am Leben gehalten wird. Redundant ausgelegtes Routing und Stabilisierungsmaßnahmen verhindern den Zusammenbruch des Netzes bei Knotenausfällen.

Ein solches Netz organisiert sich selbst und ist nicht von zentralen Koordinationsinstanzen abhängig. Ein weiterer wichtiger Punkt sind die Kosten. Ein Betreiber muss für Dienste mit hohem Datenaufkommen meist zusätzliche Server bereitstellen, um eine große Last zu verteilen. Einrichtung, Pflege und Anpassung eines solchen Systems sind kosten- und zeitintensive Aufgaben. In Peer-to-Peer Systemen trägt jeder Nutzer mit seinen eigenen Ressourcen zum Erhalt des Netzes und des Dienstes bei, wodurch das Netz mit der Anzahl der Nutzer skaliert. Außerdem werden dadurch die Kosten auf die beteiligten Nutzer verteilt. Zusammenfassend lässt sich feststellen, dass Peer-to-Peer Systeme gut skalieren und kostengünstig sind, nicht von zentralen Instanzen abhängen sowie robust gegenüber Knotenausfällen sind.

Im speziellen Fall von Voice-over-IP kommt bisher das *Session Initiation Protocol* (SIP) zum Einsatz, welches über ein Netz von Servern, sogenannte *SIP-Proxies*, die Verbindung zwischen zwei Teilnehmern herstellt. Hier kann ein Peer-to-Peer System eingesetzt werden um ein dezentrales und autonomes VoIP zu ermöglichen. Die Daten, welche zuvor auf SIP-Proxies abgelegt wurden, wie beispielsweise die Teilnehmerkennungen, werden auf den beteiligten Endsystemen gespeichert.

Eine besondere Herausforderung stellt die Sicherheit eines solchen Systems dar. In Peer-to-Peer Systemen, die heutzutage in Filesharinganwendungen eingesetzt werden, spielen Sicherheitsaspekte eine untergeordnete Rolle. Beispielsweise wird oftmals nicht verhindert, dass ein Teilnehmer die bei ihm gespeicherten Daten zu seinem Vorteil missbrauchen oder fälschen kann. Sollte eine Suchanfrage nach einer Datei unbeantwortet bleiben, obwohl diese im Netz existiert, wird dies vom Nutzer in aller Regel toleriert, da er in den meisten Fällen nicht sicher weiß, ob die

Datei existiert. Bisher wurde in Filesharinganwendungen lediglich Wert darauf gelegt, sogenanntes „free-riding“, ein Herunterladen ohne gleichzeitiges Hochladen von Dateien, zu verhindern.

Die erfolglose Suche eines Kommunikationsteilnehmers bei Voice-over-IP hingegen führt zu der Unerreichbarkeit dieses Teilnehmers. Die Unerreichbarkeit einer Notrufstelle beispielsweise kann schwerwiegende Folgen haben und benötigt deshalb einen sicheren und zuverlässigen Dienst. Daher müssen für Peer-to-Peer Systeme, die für dezentrales VoIP eingesetzt werden, neue Sicherheitskonzepte entworfen werden.

1.1 Ziel dieser Arbeit

Ziel dieser Arbeit ist der Entwurf und die Evaluation eines Peer-to-Peer Systems, welches den Anforderungen von Voice-over-IP gerecht wird. Der Fokus der Arbeit liegt dabei auf den Sicherheitsanforderungen eines solchen Systems. Dazu sollen zunächst mögliche Angriffe analysiert und geeignete Gegenmaßnahmen erarbeitet werden, welche in den Entwurf eines dezentralen Voice-over-IP Systems einfließen. Zusätzlich zu den theoretischen Überlegungen sollen die sicherheitsrelevanten Teile dieses Entwurfs noch mittels Simulationen evaluiert werden.

Als Basis dieser Arbeit sollen das Peer-to-Peer System Kademlia[1] [MM02] und das weit verbreitete Voice-over-IP Protokoll SIP [HSS97] dienen.

1.2 Gliederung dieser Arbeit

Die Arbeit gliedert sich in fünf Teile. Zunächst werden in Kapitel 2 die zum Verständnis der Arbeit erforderlichen Grundlagen erklärt. Dann werden in Kapitel 3 die Anforderungen von Voice-over-IP definiert sowie analysiert auf welche Weise ein Peer-to-Peer System diesen Anforderungen gerecht werden kann. Auf Basis dieser Analyse wird dann in Kapitel 4 ein Peer-to-Peer System entworfen, welches den zuvor aufgestellten Sicherheitsanforderungen gerecht wird.

In Kapitel 5 wird die verwendete Simulationsumgebung OverSim vorgestellt sowie die Stufen der erstellten Implementierung beschrieben. Abschließend werden in Kapitel 6 die Ergebnisse der Simulationen vorgestellt und bewertet.

[1] Kademlia hat sich bereits durch die Filesharingbörsen „eDonkey“ bzw. „emule“ mit mehreren Millionen Teilnehmer bewährt

2. Grundlagen

In diesem Kapitel werden die zum Verständnis dieser Arbeit benötigten Grundlagen erklärt. Die ersten Abschnitte beschäftigen sich zunächst mit der Definition, den Eigenschaften und den verschiedenen Arten von Peer-to-Peer Netzen sowie möglichen Angriffen. Danach wird die Architektur von dezentralen Voice-over-IP Systemen vorgestellt. Hier wird insbesondere Bezug auf das Session Initiation Protocol (SIP) genommen. Abschließend wird das Kademlia Peer-to-Peer System beschrieben, welches die Grundlage dieser Arbeit bildet und einige kryptographische Grundbegriffe erläutert.

2.1 Peer-to-Peer Netze

Der Begriff Peer-to-Peer (P2P) basiert auf der Idee Informationen und Ressourcen in einem Netzwerk von gleichberechtigten Teilnehmern spontan und ohne zentrale Koordination auszutauschen. Im Vergleich zu klassischen Client/Server-Netzen verfügen P2P-Netze über eine bessere Skalierbarkeit und verursachen niedrigere Kosten für den Betreiber. P2P-Netze haben folgende Eigenschaften:

Sharing: Ressourcen und Dienste werden gemeinsam genutzt. Ein Teilnehmer ist somit gleichzeitig Server und Client.

Dezentralisierung: In einem P2P-Netz existiert keine zentrale Instanz, welche die Nutzung von Ressourcen und Diensten verwaltet oder diese zuweist. Auch der Netzaufbau wird nicht zentral kontrolliert oder überwacht. Deshalb spricht man auch von einer „direkten“ Kommunikation zwischen den gleichberechtigten Teilnehmern.

Selbstverwaltung: Jeder Teilnehmer eines P2P-Netzes kann selbst bestimmen, in welcher Quantität und Qualität er Dienste oder Ressourcen anderen Teilnehmern zur Verfügung stellt.

Selbstorganisierung: Ein P2P-Netz organisiert sich selbst. Das heißt, dass die Topologie des Netzwerks durch jeden einzelnen Teilnehmer aufgebaut wird.

Selbstheilend: Verlassen Teilnehmer das Netz, so stabilisiert sich das Netz selbst. Die Lücke, welche diese hinterlassen, schließt sich selbstständig.

Diese Eigenschaften machen P2P-Systeme zu interessanten Netzen, welche sich an dem „Gemeinschaftssinn“ der Teilnehmer orientieren und sich nicht durch zentrale Instanzen abgeschalten lassen. P2P-Netze werden als Overlaynetze realisiert. Diese setzen auf einem bestehenden Netz wie beispielsweise TCP/IP auf.

Damit ein Teilnehmer einem P2P-Netz beitreten kann, muss er bereits einen oder mehrere andere Knoten im Netz kennen. Diese Knoten werden *Bootstrapknoten* genannt. Über diese Knoten kann ein Teilnehmer dann weitere Knoten kennenlernen und gliedert sich selbst in das Netz ein.

Die Konstruktion eines P2P-Netzes lässt sich in zwei Kategorien einteilen: unstrukturierte und strukturierte P2P-Netze.

Bei unstrukurierten P2P-Netzen bilden die Knoten ein zufälliges Netz. Jeder Teilnehmer kennt eine bestimmte Anzahl anderer Teilnehmer, die zufällig gewählt werden. Wird nach einer Ressource oder einem anderen Teilnehmer gesucht, so wird in einem einfachen unstrukurierten Netz die Suchanfrage geflutet, bis der Knoten oder die Ressource gefunden wurde. Es existieren auch Optimierungen der Suche, beispielsweise durch „Random-Walk“. In jedem Fall führt dies dazu, dass unter Umständen eine große Anzahl von Nachrichten benötigt wird, bis eine Suchanfrage beantwortet werden kann. Unstrukturierte P2P Netze werden auch oft als P2P Netze der 1. Generation bezeichnet. Gnutella [Fra] war eines der ersten unstrukturierten P2P Netze.

Strukturierte P2P Netze wie CAN [RFH+00], Chord[SMK+01] oder Kademlia[MM02] besitzen eine klare Struktur. Jeder Knoten steht an einer bestimmten Stelle im Netz, welche über die Knotenkennung (*NodeID*) festgelegt wird. Durch die Struktur sind Suchanfragen gezielter möglich und kommen mit weniger Nachrichten aus. Der übliche Durchmesser des Netzwerkgraphen eines P2P-Netzes, sowie die Länge eines Pfades, liegt in $O(\log N)$ (N ist die Anzahl der Knoten). In dieser Arbeit werden ausschließlich strukturierte P2P-Netze betrachtet.

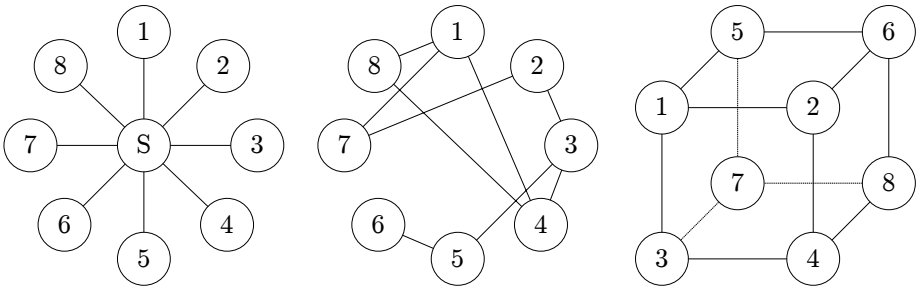


Abbildung 2.1: Netzwerke im Vergleich (von links nach rechts): Client-Server, Unstrukturiertes P2P, Strukturiertes P2P

Strukturierte P2P-Netze bestehen oftmals aus zwei Komponenten: Dem Key-based Routing, welches eine Nachricht an einen gegebenen Knoten weiterleitet, und der verteilten Hashtabelle, welche als Datenablage in einem P2P-Netz dient. In den nächsten beiden Abschnitten werden diese Komponenten erläutert. Dananch wird auf die Sicherheit und mögliche Angriffe auf P2P-Netze eingegangen.

2.1.1 Key-based Routing

Das Key-based Routing (KBR) ermöglicht es, einen Knoten mit einer bestimmten *NodeID* zu finden. Es strukturiert das Netz, indem es der Topologie entsprechend Routingtabellen aufbaut. Die am häufigsten eingesetzten Topologien basieren auf Hyperkubus- und DeBruijn-Graphen.

In Hyperkubus-basierenden Ansätzen haben die Knoten üblicherweise einen Ein- und Ausgangsgrad von $O(\log N)$ und eine normalverteilte Pfadlänge. Bei DeBruijn-basierenden Ansätzen ist der Ein- und Ausgangsgrad sowie die Pfadlänge weitgehend konstant.

Beide Topologien zeichnen sich durch eine maximale Pfadlänge, den Durchmesser des Graphen, von $O(\log N)$ aus. Um die Verlässlichkeit zu erhöhen, bieten einige KBR-Protokolle einen Parameter k an. Dieser gibt an, wie viele redundante Wege im Netz verwaltet werden.

Um den Durchmesser zu verkleinern, werden oft bei jedem Hop b Bits der *NodeID* gleichzeitig verwendet. Dies verringert den Durchmesser des Graphens üblicherweise auf $O(\log_{2^b} N)$.

KBR-Protokolle lassen sich in zwei Klassen aufteilen: *reaktive* und *proaktive* Protokolle. Proaktive KBRs benötigen einen Stabilisierungsmechanismus, welcher die Topologie des Netzes aufrecht erhält. Reaktive KBRs füllen ihre Routingtabellen durch eingehende Suchanfragen und können durch ihre Topologie auf spezielle Stabilisierungsnachrichten verzichten. Sie haben aufgrund dieser Eigenschaft viele Sicherheitsvorteile, wie sich in dieser Arbeit herausstellen wird.

Beispiele für Hyperkubus-basierende KBR sind Chord [SMK+01] (proaktiv), Tapestry [ZKJ01] (proaktiv) und Kademlia [MM02] (reaktiv). Für DeBruijn-basierende seien hier Koorde [KK03] (proaktiv) und Broose [GV04] (reaktiv) erwähnt.

Im nächsten Abschnitt soll nun beschrieben werden, wie Daten auf Basis eines KBRs abgelegt werden können.

2.1.2 Verteilte Hashtabellen

Auf der Basis des Key-based Routing können Daten in einer verteilten Hashtabelle (DHT[1]) abgelegt werden. Jeder Knoten speichert Daten in Form von Schlüssel-Wert Paaren von anderen Knoten.

Die übliche Strategie ist den Hashwert über den Schlüssel zu berechnen und das Schlüssel-Wert Paar dann auf dem Knoten mit der naheliegendsten *NodeID* abzulegen. Um zu verhindern, dass Daten verloren gehen, wenn dieser Knoten das Netz verlässt, werden Replikationsmechanismen eingesetzt, um das Schlüssel-Wert-Paar auf mehreren Knoten redundant zu verteilen. Da die Daten durch die Hashfunktion nicht geographisch nahe, sondern „zufällig“ verteilt abgelegt werden, sind sie vor einem geographisch-lokalen Ausfall geschützt.

Im nächsten Abschnitt wird die Sicherheit von abgelegten Daten und des Routings betrachtet.

[1] Distributed Hashtable

2.1.3 Sicherheit in Peer-to-Peer Netzen

Peer-to-Peer Systeme sind wie jedes Netz zahlreichen möglichen Angriffen ausgesetzt. Diese Angriffe können in drei Kategorien eingeteilt werden, die im Folgenden beschrieben werden. Ausführlichere Informationen sind in [SM02] zu finden.

Angriffe auf das Routing

Bei dieser Art von Angriff wird gezielt versucht das Key-based Routing anzugreifen. Sie haben die Absicht, entweder das Netz zu partitionieren, Anfragen in ein paralleles Netz umzuleiten oder aber das gesamte Netz zu stören. Die einzelnen Angriffe werden hier kurz vorgestellt:

Eclipseangriff: Der Eclipseangriff versucht das Netz in ein oder mehrere Teilnetze zu spalten. Dazu muss der Angreifer gezielt Knoten in das Netz einbringen, so dass er einen Keil zwischen das Netz treibt. Abbildung 2.2 zeigt diesen Angriff.

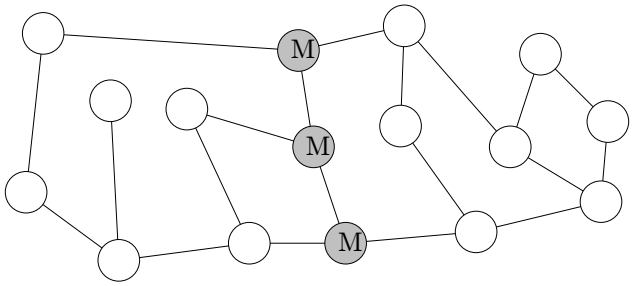


Abbildung 2.2: Der Eclipseangriff

Churnangriff: Bei einem Churnangriff betritt ein Angreifer mit vielen Knoten das Netz und verlässt es nach kurzer Zeit wieder. Dieser Vorgang wird so lange wiederholt, bis sich das Netz durch seine eigenen Stabilisierungsmaßnahmen lahm legt oder partitioniert.

Sybilangriff: Bei einem Sybilangriff versucht ein Angreifer mit möglichst vielen Identitäten gleichzeitig dem Netz beizutreten, bis er einen bestimmten Prozentsatz aller Knoten hält. Damit kann der Angreifer die vollständige Kontrolle über gespeicherte Daten erlangen oder bestimmte Knoten aus dem Netz ausschließen. Auch die vollständige Zerstörung der Netztopologie ist möglich.

Angriffe auf die Datenspeicherung

Diese Angriffe beziehen sich auf die Datenspeicherung der DHT. Ein Knoten kann seine lokal gespeicherten Daten jederzeit verändern, löschen oder gar nicht erst speichern. Dies führt zu Datenverlust oder manipulierten Daten. Wird ein Replikationsmechanismus verwendet, so kann ein Angreifer durch gezielte Wahl der *NodeID* versuchen alle Knoten zu kontrollieren, welche die Replikate zu einem bestimmten Schlüssel-Wert-Paar speichern.

Angriffe auf das Underlay

Der einfachste Angriff im Underlay ist der Denial-of-Service Angriff. Ein Knoten wird schlichtweg mit Anfragen überhäuft, bis dieser seinen Dienst einstellt. Ein weiterer Angriff ist der *Man-in-the-Middle* Angriff. Abbildung 2.3 zeigt diese Konstellation. Möchten Alice und Bob miteinander kommunizieren, so kann diese Verbindung über einen bösartigen Dritten (Eve) geroutet werden. Dieser kann die gesamte Kommunikation zwischen Alice und Bob mitlesen sowie übertragene Nachrichten modifizieren.

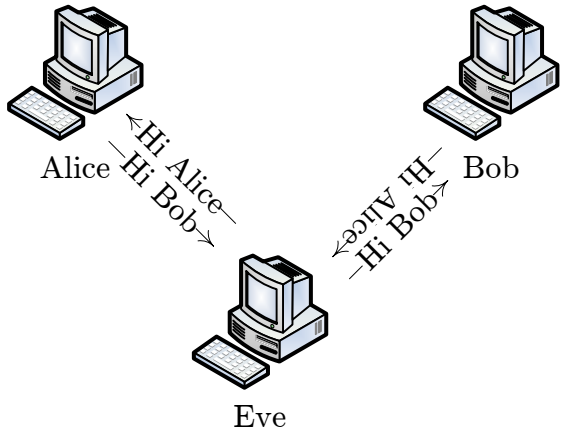


Abbildung 2.3: Der *Man-in-the-Middle* Angriff

2.2 Dezentrales Voice-over-IP

Um dezentrales Voice-over-IP zu erläutern, wird in diesem Abschnitt zunächst das *Session Initiation Protocol* [HSS97] vorgestellt. Danach wird ein mögliches dezentralisiertes Voice-over-IP Netz vorgestellt und dessen Vor- und Nachteile diskutiert.

Session Initiation Protocol (SIP)

Das *Session Initiation Protocol* (SIP) ist der defacto-Standard für Voice-over-IP Netze. Es wurde von der IETF[2] in den RFCs[3] 3261, 3515 und 3485 standardisiert. SIP übernimmt folgende Aufgaben:

1. Lokalisierung eines Endpunkts (Client)
2. Namensregistrierung eines Endpunkts
3. Weiterleiten von Nachrichten
4. Kommunikationssitzungen aushandeln

Es besteht aus folgenden Komponenten:

SIP Proxy: Der SIP-Proxy übernimmt das Weiterleiten von Daten und die Lokalisierung eines Endpunkts. Mehrere SIP-Proxys können vernetzt werden, so dass ein redundant ausgelegtes Netz entsteht.

[2]Internet Engineering Task Force
[3]Request For Comment

SIP Registrar: Ein SIP-Registrar hält Nutzerinformationen. Meist wird ein SIP Registrar in einem Proxy integriert.

SIP Client: Ein SIP Client ist ein Endpunkt in einem SIP-Netz und wird auch *User Agent* (UA) genannt. Er kann sich an einem SIP-Proxy registrieren und über das SIP-Netz Sitzungen aufbauen oder sich direkt mit einem anderen Teilnehmer verbinden.

SIP nutzt die das *Domain Name System* um Namen aufzulösen sowie UDP um Nachrichten auszutauschen. Es wird lediglich verwendet um eine Sitzung aufzubauen. Sobald die Sitzung aufgebaut ist, kommunizieren die Teilnehmer direkt miteinander. Man kann somit von einer Peer-to-Peer *ähnlichen* Struktur reden. Abbildung 2.4 zeigt die SIP-Architektur.

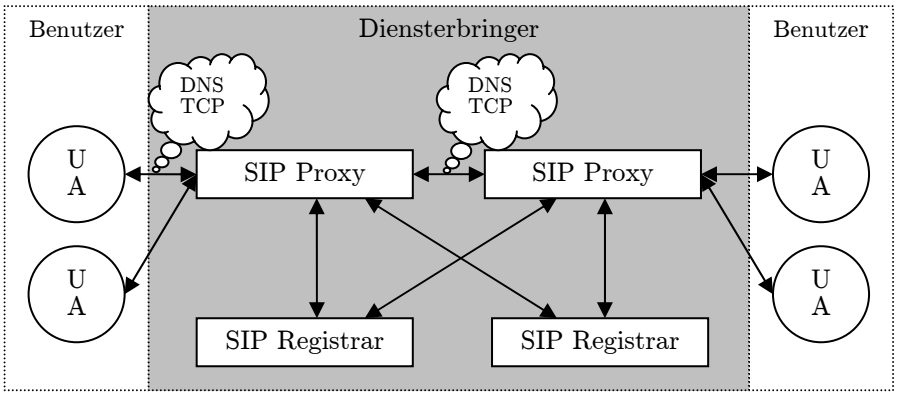


Abbildung 2.4: Die SIP-Architektur

Zur Aushandlung der Sitzungsparameter wird für VoIP das *Session Description Protocol* [HJ97] (SDP) verwendet. Hier werden beispielsweise Codecs für die Sprachübertragung ausgehandelt.

Die Übertragung der Sprachdaten geschieht schließlich über das *Realtime Transfer Protocol* [SCFJ98] (RTP). Abbildung 2.5 zeigt den Aufbau einer Sitzung und das Aushandeln von Kommunikationsparametern.

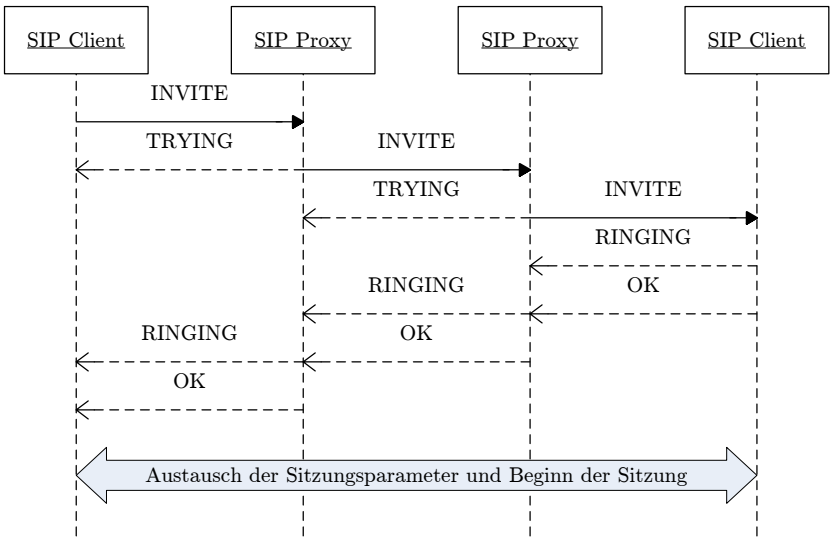


Abbildung 2.5: Aufbau einer SIP-Sitzung

Die SIP-Architektur ist zwar nahezu ein Peer-to-Peer System, da SIP-Proxies verteilt und redundant ausgelegt sind, die Namensauflösung geschieht aber immernoch über zentrale Dienste wie DNS. Auch ist das Netz ohne Administration durch einen Netzbetreiber nicht realisierbar.

Im nächsten Absatz werden die Charakteristika eines möglichen dezentralen SIP-Systems vorgestellt.

Dezentrales Voice-over-IP (P2P-SIP)

Bei dezentralem Voice-over-IP wird kein administrierter Server verwendet. Die Ablage von Nutzerdaten muss bei den Nutzern selbst erfolgen. Jeder Teilnehmer übernimmt gleichzeitig die Datenspeicherung und die Weiterleitung von Nachrichten anderer Nutzer. Der primäre Unterschied zum gewöhnlichen SIP ist, dass kein DNS-Server verwendet wird. Informationen, welche zuvor auf den SIP-Proxies sowie den SIP-Registrars abgelegt wurden, müssen auf die Teilnehmer verteilt werden.

Hierzu kann eine verteilte Hashtabelle verwendet werden, welche Daten verteilt speichert. Das Key-based Routing übernimmt die Lokalisation eines Teilnehmers. Abbildung 2.6 zeigt eine mögliche Architektur eines P2P-SIP-Systems.

Viele verteile Hashtabellen sind unsicher, da Daten von einzelnen Teilnehmern manipuliert werden können. Die Lokalisation von Teilnehmern ist damit nicht immer gewährleistet.

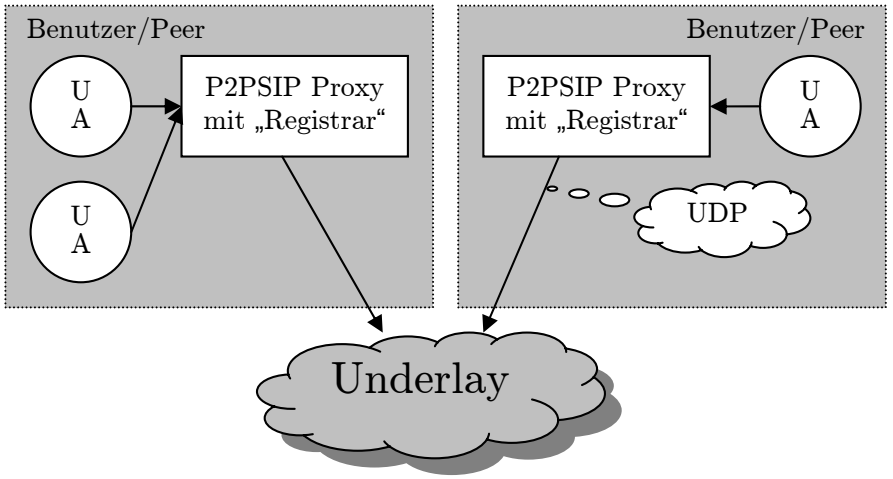


Abbildung 2.6: Die **P2PSIP**-Architektur

2.3 Kademlia

Kademlia [MM02] ist ein strukturiertes Peer-to-Peer System. Im Gegensatz zu Chord [SMK+01] ist das Protokoll reaktiv, das heißt, die Routingtabelle wird durch Suchanfragen anderer Knoten gefüllt und stabilisiert. Ein expliziter Stabilisierungsmechanismus ist durch die symmetrische XOR-Topologie Kademlias nicht notwenig. Der Vorteil ist, dass diese Art der Wartung weniger Sicherheitsrisiken mit sich bringt, da bösartige Knoten nur schwer die Routingtabelle eines Knotens beeinflussen können. Durch einen parallelen, asynchronen Lookup werden Verzögerungen minimiert und ausgefallene Knoten in den Routingtabellen kompensiert.

Der Grund für die Wahl von Kademlia in dieser Arbeit sind nicht nur dessen Sicherheitsaspekte, sondern auch die Tatsache, dass es das erste Peer-to-Peer System ist, welches sich bereits in der Praxis bewährt hat. Es funktioniert in der Filesharingtauschbörsen „Overnet“ bzw. „eMule“ seit einigen Jahren sehr zuverlässig mit mehreren Millionen Knoten.

Kademlia nutzt wie viele andere verteilte Hashtabellen einen *ID*-Raum von 160-Bit. Knoten, die dem Netz beitreten, verwenden somit eine 160-Bit *NodeID*. Daten, in Form von Schlüssel-Wert Paaren, welche in der verteilten Hashtabelle abgelegt werden sollen, werden immer auf den naheliegendsten Knoten zum Hashwert des Schlüssels gespeichert.

Die Routingtabelle von Kademlia hat eine deutliche Ähnlichkeit zu der von Tapestry [ZKJ01], was nicht weiter verwunderlich ist, da beide präfix basierend sind. Der Unterschied ist, dass durch die XOR-Metrik keine zusätzlichen Routingtabellen benötigt werden, wenn der Lookup nahe an einen Knoten kommt, da die XOR-Metrik bei gleichem Präfix auch den Suffix berücksichtigt.

In dem nächsten Abschnitten wird zunächst die Topologie der XOR-Metrik erklärt. Es folgen die Beschreibungen des Key-based Routings sowie der verteilten Hashtabelle Kademlias.

2.3.1 Kademlia XOR-Metrik

Jede *NodeID* und jede *ID* eines Schlüssels sind bei Kademlia 160 Bit lang. Um den Abstand d zwischen zwei *ID*s zu bestimmen, verwendet Kademlia die XOR-Operation $d : (x, y) \rightarrow x \oplus y$. Die XOR-Operation bildet damit eine nicht-euklidische Metrik und es gilt:

$$\forall\ x : d(x, x) = 0 \text{ (reflexiv)}$$

$$\forall\ x, y : d(x, y) > 0, \text{ falls } x \neq y \text{ (positiv definit)}$$

$$\forall\ x, y : d(x, y) = d(y, x) \text{ (symmetrisch)}$$

Insbesondere erfüllt die XOR-Operation auch die Dreiecksungleichung, da $d(x, y) \oplus d(y, z) = d(x, z)$ und $\forall\ a \geq 0, b \geq 0 a + b \geq a \oplus b$ gilt, gilt auch:

$$\forall\ x, y : d(x, y) + d(y, z) \geq d(x, z)$$

Dies wird klar, wenn die binäre Addition von Zahlen, ohne Berücksichtigung des Übertrags, betrachtet wird.

Diese Metrik lässt sich in einem binären Trie[4] veranschaulichen. Wird ein vollständig gefüllter binärer Trie mit allen 160 Bit *ID*s betrachtet, so ist der Abstand zwischen zwei *ID*s gerade die Höhe des kleinsten Untertries, welcher beide *ID*s enthält.

Eine weitere Eigenschaft der XOR-Metrik ist, dass für jeden Abstand $d(x, y)$ zu einer *ID* x genau ein y existiert. Somit ist die Metrik *unidirektional*.

2.3.2 Kademlia Key-based Routing

Das Kademlia Key-based Routing besteht aus drei Teilen: Dem Aufbau der Routingtabelle, der Wartung und einem Algorithmus für das Suchen einer *NodeID* (Lookup). In den folgenden Absätzen werden diese Teile erklärt.

Aufbau der Routingtabelle

Kademlias Routingtabelle besteht aus Listen mit k Einträgen. Eine solche Liste wird k-Bucket genannt. Jeder Eintrag der Liste umfasst IP-Adresse, Port und *NodeID*. Durch die XOR-Metrik kann die Routingtabelle als Präfixbaum dargestellt werden. Der Parameter k gibt an, wieviele redundante Wege in der Routingtabelle gehalten werden sollen.

Zunächst besteht die Routingtabelle aus einem einzigem k-Bucket, welcher den gesamten Raum der *NodeID*s umfasst bzw. überdeckt. Ist dieser Bucket voll, so wird er aufgeteilt, wenn er die eigene *NodeID* überdeckt. Auf diese Weise entsteht bei 2^{160} Knoten eine Routingtabelle mit $i = 0 \ldots 159$ k-Buckets, welche den Wertebereich 2^i bis 2^{i+1} abdecken.

Allerdings kann nicht sichergestellt werden, dass jeder Knoten seine k Nachbarn kennt. Deshalb wird der k-Bucket auch dann aufgeteilt, wenn sich der k-Bucket in dem kleinsten Teilbaum, welcher die eigene *NodeID* abdeckt, befindet.

Abbildung 2.7 zeigt den schrittweisen Aufbau der Routingtabelle.

[4] Ein Trie ist ein Baum, in dem nur dessen Blätter Daten beinhalten.

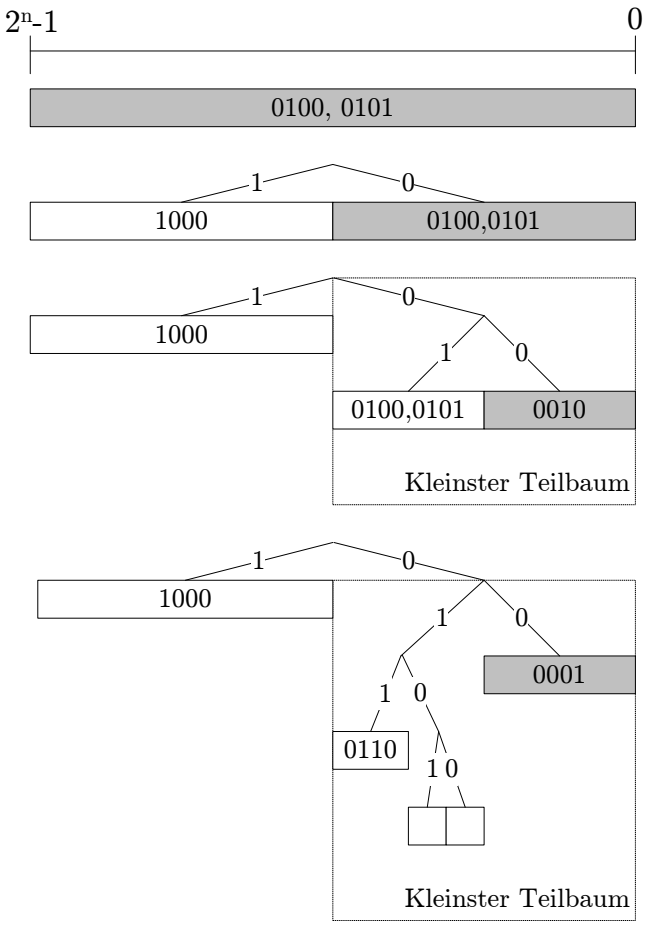


Abbildung 2.7: Die Kademlia Routingtabelle eines Knotens mit *NodeID*= 0. Der k-Bucket, welche die eigene *NodeID* überdeckt, ist grau hinterlegt.

Um den Durchmesser des Overlay-Graphen und damit die maximale Pfadlänge zu verkleinern, benutzt Kademlia den Parameter b. Er gibt an, wieviele Bits der *NodeID* in einem Schritt berücksichtigt werden sollen. Für die Routingtabelle bedeutet dies, dass ein k-Bucket außerdem aufgeteilt wird, wenn für die Tiefe d (im Baum) eines k-Buckets in der Routingtabelle $d \leq b$ gilt.

Damit gilt die allgemeine Aufteilungsregel: Ein k-Bucket wird aufgeteilt, wenn sich der k-Bucket im kleinsten Teilbaum, welcher die eigene *NodeID* überdeckt, befindet oder $d \leq b$ gilt.

Sei n die Anzahl der Bits der *NodeID*, b die Anzahl der zu berücksichtigenden Bits, k die Anzahl der Einträge in dem k-Bucket und N die Anzahl der Knoten im Netz. Dann ist die zu erwartende Größe der Routingtabelle

$$O(k \cdot (2^b - 1) \cdot \log_{2^b} N)$$

und der Durchmesser liegt in

$$O(\log_{2^b} N)$$

Damit andere Knoten Suchanfragen formulieren können, stellt die Routingtabelle einen RPC-Aufruf `FIND_NODE` zur Verfügung. Dieser gibt zu einer gegebenen *NodeID* die k naheliegendsten Knoten aus der Routingtabelle zurück.

Wartung der Routingtabelle

Die Routingtabelle wird durch eingehende Anfragen gefüllt. Ist ein Bucket voll, so wird er wie im vorherigen Absatz beschrieben aufgeteilt. Ist der k-Bucket vollständig gefüllt und kann *nicht* aufgeteilt werden, so wird der neue Knoten verworfen. Wenn ein Knoten bereits bekannt ist, so wird er in dem k-Bucket an das Ende der Liste verschoben. Dieses Vorgehen realisiert eine *Least-Recently-Used* (LRU) Strategie. Dies hat den Vorteil, dass ältere Knoten immer am Anfang eines k-Buckets stehen und damit schneller überprüft werden kann, ob ein Knoten eines k-Buckets das Netz verlassen hat.

Sollte ein k-Bucket für längere Zeit keinen Zugriff haben, so wird ein Lookup auf eine zufällige *ID* aus dem k-Bucket durchgeführt. Dies führt dazu, dass der k-Bucket aufgefrischt wird.

Um dem Netz beizutreten, wird ein Lookup auf die eigene *NodeID* durchgeführt. Dadurch erhalten die Nachbarn des Knotens Information über den neuen Knoten und die k-Buckets werden durch die Antworten anderer Knoten gefüllt. Dieser Vorgang wird jede Stunde wiederholt, um die Präsenz im Netz zu gewährleisten.

Lookup einer *ID*

Der Lookup einer *ID* geschieht iterativ und parallel asynchron. Zunächst wird ein `FIND_NODE` auf dem eigenen Knoten ausgeführt. Auf α der zurückgegebenen Knoten (auch Kontakte genannt) werden dann gleichzeitig `FIND_NODE` RPCs aufgerufen. Die zurückgegebenen Knoten werden einer aufsteigend nach Abstand zur Ziel *ID* sortierten Liste mit k Einträgen hinzugefügt. In der Liste wird jeder bereits befragte Knoten markiert. Antwortet keiner der Knoten, so werden weitere α Knoten befragt, bis alle k Knoten befragt wurden.

Antwortet ein Knoten auf eine Anfrage nicht, so wird er aus dem entsprechenden k-Bucket entfernt.

Steht die gesuchte *ID* am Anfang der Liste, so ist die Suche beendet und die *NodeID* wurde gefunden. Ist die Liste vollständig gefüllt, und konnte keiner der Anfragen weitere Knoten hinzufügen, so ist die Suche beendet und die Liste umfasst die k naheliegendsten Knoten zu der *ID*.

Abbildung 2.8 zeigt die Lookupliste während des Auffindens der $ID = 240$. Zunächst werden die naheliegendsten Knoten aus der eigenen Routingtabelle eingefügt. Dann wird der Knoten mit der $ID = 1298$ nach den naheliegendsten Knoten gefragt und markiert. Als nächstes wird der Knoten mit der $ID = 2390$ gefragt. Dieser gibt deutlich naheliegendere Knoten zurück, was dazu führt, dass die gesamte Liste neu gefüllt wird. Der gleiche Vorgang wiederholt sich, wenn im nächsten Schritt der Knoten mit der $ID = 500$ gefragt wird. In den letzten beiden Schritten verändert sich die Liste durch weitere Anfragen nicht mehr und die Suche ist beendet. Die Liste beeinhaltet nun die k Nachbarn zu der $ID = 240$.

2.3.3 Verteilte Hashtabelle

Die verteilte Hashtabelle wird bei Kademlia durch zwei RPCs realisiert: `STORE` und `FIND_VALUE`. Der `STORE` RPC speichert ein Schlüssel-Wert Paar auf einem Knoten,

1298
78457
12932324
234239488

1298
2390
4359
3249

500
645
124
24

260
124
50
24

250
124
50
24

Abbildung 2.8: Der Zustand der Lookupliste für die *ID*= 240, welche keine *Node-ID* repräsentiert, in Schritten von links nach rechts.

`FIND_VALUE` funktioniert wie `FIND_NODE`, nur dass er ein Schlüssel-Wert Paar zurückgibt, falls der Knoten dieses gespeichert hat.

Jedes Schlüssel-Wert Paar enthält ein `Time-to-Live` Attribut, welches angibt, wie lange ein Schlüssel-Wert Paar gespeichert werden soll. In der Arbeit zu Kademlia wird eine Lebenszeit von einer Stunde vorgeschlagen. Dies verhindert Karteileichen in der verteilten Hashtabelle. Es hat allerdings zur Folge, dass die Daten regelmäßig erneuert werden müssen.

Will ein Knoten ein Schlüssel-Wert Paar ablegen, so führt dieser einen Lookup auf den Hashwert des Schlüssels aus. Danach werden über den `STORE` RPC die Daten auf den naheliegendsten Knoten abgelegt. Um zu vermeiden, dass Daten verloren gehen, wenn dieser Knoten ausfallen sollte, kann das Schlüssel-Wert Paar auf den k Nachbarn repliziert werden.

Die Replikation hat auch den Vorteil, dass „Daten-HotSpots“ vermieden werden. Da alle Lookups zu dem gleichen Pfad konvergieren, ist es sehr wahrscheinlich, dass Suchanfragen auf die replizierenden Knoten verteilt werden.

2.4 Kryptographie

In diesem Abschnitt werden kryptographische Grundbegriffe, welche in dieser Arbeit verwendet werden, kurz erläutert. Ausführlichere Informationen können unter [Gol95] nachgelesen werden.

Einwegfunktionen

Einwegfunktionen sind bijektive Funktionen, welche sich einfach berechnen lassen, aber nur mit sehr viel Aufwand invertierbar sind. Bis heute ist nicht bewiesen, ob solche Funktionen wirklich existieren. Sie bilden aber seit mehreren Jahren eine solide Basis für Kryptosysteme, welche ausgiebig analysiert worden sind. Deshalb kann mit hoher Wahrscheinlichkeit angenommen werden, dass Einwegfunktionen wirklich existieren.

Kryptographische Hashfunktionen

Kryptographische Hashfunktionen sind Funktionen, welche die genannten Einwegeigenschaften besitzen und zusätzlich folgenden Anforderungen genügen:

Datenkompression: Eine Hashfunktion sollte deutlich kleinere Werte liefern als ihre Eingangsdaten.

Kollisionsfreiheit: Es darf nur mit sehr viel Aufwand möglich sein, den selben Hashwert zu zwei oder mehreren verschiedenen Eingabedaten zu generieren.

Konsistente kryptographische Hashfunktionen haben zusätzlich immer eine feste Ausgangsgröße. Am weitesten verbreitet sind die SHA1 und MD5 Hashfunktionen.

Symmetrische Verschlüsselung

Die symmetrische Verschlüsselung benutzt einen gemeinsamen Schlüssel um Daten zu Ver- und Entschlüsseln. Die Güte eines Verschlüsselungsalgorithmus wird anhand statistischer Daten über das Chiffrat ermittelt. So sollte dieses nicht von Zufallszahlen unterscheidbar sein.

Die symmetrische Verschlüsselung verhindert das Mitlesen von Daten durch einen Dritten, wenn sich zwei Parteien auf einen symmetrischen Schlüssel geeinigt haben und die Verschlüsselung verwenden.

Symmetrische Verschlüsselungsverfahren sind gegenüber asymmetrischen Verfahren relativ effizient. Eines der bekanntesten symmetrischen Verschlüsselungsverfahren ist AES. Dies ist ein blockorientiertes Verfahren. Das heißt, es können nur Blöcke gleicher Länge verschlüsselt werden.

Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselungsverfahren nutzen zwei Schlüssel. Einen Öffentlichen und einen Privaten. Der öffentliche Schlüssel wird benutzt, um Daten zu verschlüsseln. Diese Daten können dann nur noch vom Inhaber des privaten Schlüssels entschlüsselt werden.

Realisiert werden diese Verfahren über Einwegfunktionen, welche mit dem privaten Schlüssel leicht invertierbar sind. Solche Funktionen sind beispielsweise die Faktorisierung großer Primzahlen oder die Berechnung des diskreten Logarithmus. Wie bei den Einwegfunktionen, ist es bisher noch nicht gelungen zu beweisen, dass beispielsweise die Faktorisierung von Primzahlen ausschließlich in exponentieller Zeit lösbar ist. Dennoch wurden bisher noch keine Algorithmen gefunden, welche diese beiden Probleme in polynomieller Zeit lösen können.

Zwei asymmetrische Verschlüsselungsverfahren sind weit verbreitet: Der RSA Algorithmus, welcher auf großen Primzahlen beruht, sowie ein Verfahren auf Basis elliptischer Kurven (ECC), welches auf dem diskreten Logarithmus beruht. Beide Kryptosysteme wurden über mehrere Jahre analysiert und für sicher befunden.

Asymmetrische Verschlüsselungsverfahren bestehen aus drei Primitiven: Schlüsselgenerierung, Verschlüsselung mit dem öffentlichen Schlüssel und Entschlüsselung mit dem privaten Schlüssel.

Digitale Signaturen

Digitale Signaturen sind ein Mittel, um die Integrität von Nachrichten sicherzustellen. Sie sind mit den asymmetrischen Verschlüsselungsverfahren verwandt. Hier wird der öffentliche Schlüssel verwendet, um die Integrität bzw. die Signatur einer Nachricht zu überprüfen, und der Private, um eine Nachricht zu signieren.

Für gewöhnlich wird nicht die gesamte Nachricht signiert, sondern lediglich ein kryptographischer Hashwert der Nachricht. Dies setzt voraus, dass die verwendete Hashfunktion solide ist.

Zwei Verfahren sind weit verbreitet: Der DSA Algorithmus auf Basis großer Primzahlen und seine Abwandlung auf Basis elliptischer Kurven ECDSA. Letzterer bietet deutlich kleinere Signaturgrößen bei gleicher Sicherheit. Die Verifizierung einer Signatur dauert dafür etwas länger.

Digitale Signaturen bestehen wie die asymmetrischen Verfahren aus drei Primitiven: Schlüsselgenerierung, Erzeugen von Signaturen und der Verifizierung von Signaturen.

Kryptographische Puzzle

Kryptographische Puzzle werden häufig verwendet um Denial-of-Service Angriffe zu verhindern. Mit einer Lösung eines kryptographischen Puzzles kann ein Nutzer nachweisen, dass er eine bestimmte Rechenzeit aufgewendet hat. Sie werden üblicherweise mit kryptographischen Hashfunktionen realisiert.

Es ist bekanntlich schwer eine Kollision des Hashwerts zu berechnen. Ein kryptographisches Puzzle kann somit beispielsweise die Berechnung einer partiellen Kollision (die Übereinstimmung eines Präfixes über eine gegebene Anzahl Bits) von zwei Hashwerten darstellen. In dieser Arbeit werden mehrere Kryptopuzzles vorgestellt und ausführlich erläutert.

3. Analyse

In diesem Kapitel werden zunächst die Anforderungen an ein Overlay-Netz für dezentrales Voice-over-IP definiert. Anhand dieser Definition werden dann sicherheitsrelevante Aspekte des Key-based Routings sowie der verteilten Hashtabelle analysiert.

3.1 Anforderungen

Dezentrales Voice-over-IP (P2P-VoIP) stellt hohe Sicherheitsanforderungen an die verwendete DHT. So ist beispielsweise in Filesharinganwendungen das Auffinden eines bestimmten Teilnehmers nicht sehr wichtig. Vielmehr steht im Vordergrund, dass eine Datei auffindbar ist. Je mehr Leute eine Datei freigegeben haben, desto geringer ist die Wahrscheinlichkeit, dass eine Datei nicht gefunden wird. Bei P2P-VoIP ist eine erfolglose Suche äquivalent zu einem Ausfall des Telefons. Deshalb sollen hier die Anforderungen eines Teilnehmers an das Netz definiert werden:

Authentifizierung: Jeder Teilnehmer muss sich gegenüber anderen Teilnehmern authentifizieren können, damit sichergestellt ist, dass ein Anrufer mit der richtigen Person verbunden ist.

Erreichbarkeit: Jeder Teilnehmer muss trotz einer gewissen Anzahl bösartiger Knoten im Netz mit hoher Wahrscheinlichkeit erreichbar sein.

Lokalisation: Jeder Teilnehmer muss im Netz auffindbar sein. Dies impliziert, dass ein Teilnehmer eine Kennung an andere Teilnehmer weitergeben kann und über diese erreichbar ist. Es sei hier angemerkt, dass sich natürliche Namen besser weitergeben lassen als beispielsweise eine sehr lange Zahlenfolge.

Viele DHTs wurden in der Vergangenheit auf ihre Resistenz gegen eine hohe Teilnehmerfluktuation[1] überprüft. Diese Eigenschaft ist bei einer DHT für VoIP nahezu vernachlässigbar, denn es ist unwahrscheinlich, dass viele Teilnehmer nur für kurze Zeit erreichbar sein wollen. Die Zeit welche ein Verbindungsaufbau benötigt wird

[1] vgl. Englisch: Churn

selbst bei mehreren Sekunden nicht als störend empfunden. Dies gibt einem Suchvorgang genügend Freiraum, um mehr auf Sicherheit als auf schnellen Lookuperfolg zu achten.

Die Anforderung der Lokalisierbarkeit impliziert nicht nur die Existenz einer permanenten eindeutigen Teilnehmerkennung, sondern erfordert auch einen Namensdienst, welcher einen natürlichen Namen auf eine Teilnehmerkennung abbildet.

In dem nächsten Abschnitt werden Möglichkeiten diskutiert und analysiert, wie *NodeID*s so gewählt werden können, dass die geforderte Authentifizierbarkeit anhand einer *NodeID* garantiert ist.

3.2 Wahl der *NodeID* für den Netzbeitritt

Die Wahl der *NodeID*, mit welcher sich ein Teilnehmer in einem Overlay-Netz registriert, ist von großer Sicherheitsrelevanz. Kann eine Teilnehmer eine *NodeID* frei wählen, so könnte ein Angreifer mehrerer dieser *NodeID*s dazu verwenden, um durch gezielte Platzierung von Knoten einen anderen Knoten vom Netz abzutrennen. Wird angenommen, dass die *NodeID* nicht frei wählbar ist, so bleibt einem Angreifer die Möglichkeit so lange *NodeID*s zu generieren bis er mit den daraus resultierenden Knoten einen Großteil des Netzes beherrscht.

Weiterhin bestimmt die *NodeID* auch die Authentifizierbarkeit eines Knotens, denn ein Knoten sollte erstens eindeutig identifizierbar sein und zweitens soll ausgeschlossen werden, dass ein anderer Knoten die *NodeID* eines Knotens verwenden oder generieren kann.

Aus diesem Grund werden im Folgenden Strategien zur Generierung von *NodeID*s unter den Anforderungen der Authentifizierbarkeit, der Integritätsprüfung sowie deren möglichen Missbrauch durch einen Sybilangriff analysiert.

3.2.1 Generierung der Teilnehmerkennung

In der Arbeit des Chord-Overlays [SMK+01] wurde vorgeschlagen, die Teilnehmerkennung aus einem kryptographischen Hashwert über IP $\oplus$ Port zu generieren. Dadurch wird verhindert, dass ein Knoten seine *NodeID* frei wählen kann, da jeder Knoten den Hashwert überprüfen kann.

Dieses Vorgehen hat den Nachteil, dass bei dynamisch zugewiesenen IP-Adressen, wie es bei vielen DSL Anschlüssen der Fall ist, sich diese *NodeID* meist jede 24 Stunden ändert. Eine Authentifizierung eines Teilnehmers über diese *NodeID* ist deshalb nicht oder nur sehr eingeschränkt möglich.

Durch die Anforderung der Authentifizierbarkeit liegt es nahe die Knotenkennung kryptographisch zu generieren. Deshalb wurde von E. Sit und R. Morris [SM02] vorgeschlagen die *NodeID* als Hashwert eines öffentlichen Schlüssels zu generieren. Über ein Challenge-Response Verfahren kann damit jeder Knoten authentifiziert werden. Zwar ermöglicht dieses Verfahren die Authentifizierung eines Knotens, hindert aber keinen Angreifer daran einen Man-in-the-Middle Angriff im Underlay durchzuführen. Dadurch ist die Integriät der Nachrichten nicht gewährleistet.

Aus diesem Grund wird in [See06] vorgeschlagen, einen öffentlichen Schlüssel im Sinne der digitalen Signatur zu verwenden und dessen Hashwert als *NodeID* zu benutzen. Über jede mit diesem Schlüsselpaar signierte Nachricht kann so ein Knoten

authentifiziert werden und es wird zusätzlich die Integrität der Nachricht sichergestellt.

Es verhindert allerdings nicht, dass Nachrichten mitgeschnitten werden, da die Nachrichten unverschlüsselt übertragen werden. Dies ist jedoch in den meisten Fällen kein großer Nachteil, denn die meisten Informationen sind für einen Angreifer uninteressant sind wie sich später in der Analyse der Lookups herausstellen wird.

Die hier beschriebenen Maßnahmen ermöglichen die Authentifikation eines Knotens sowie die Integritätsprüfung der Nachrichten. Durch die Hashfunktion über den öffentlichen Schlüssel wird außerdem sichergestellt, dass eine *NodeID* nicht frei gewählt werden kann. Es wird davon ausgegangen, dass sich bösartige sowie gutartige Knoten gleichverteilt im *NodeID*-Raum wiederfinden.

Im nächsten Abschnitt wird nun behandelt, wie die Anzahl der *NodeID*s pro Teilnehmer eingeschränkt werden kann, um den daraus resultierenden Sybilangriff zu erschweren.

3.2.2 Erschweren des Sybilangriffs

Einer schwer verhinderbarer Angriff auf ein dezentrales Netzwerk ist der Sybilangriff. Bei diesem tritt ein Angreifer unter einer Vielzahl von *NodeID*s dem Netz bei mit der Absicht das Routing oder Daten zu beeinflussen. Der Erfolg eines solchen Angriffs hängt primär davon ab, wie schwer es für einen Angreifer ist in den Besitz einer Vielzahl von *NodeID*s zu gelangen.

Douceur hat bewiesen [Dou02], dass es keine dezentrale Möglichkeit gibt diesen Angriff zu verhindern. Es existieren aber Möglichkeiten, mit Hilfe von Kryptopuzzles diesen Angriff deutlich zu erschweren.

Unter Berücksichtigung der im vorherigen Abschnitt beschriebenen Erzeugung von *NodeID*s, lässt sich folgendes Kryptopuzzle [ANL01] formulieren:

1. Generiere Schlüsselpaar k_{priv}, k_{pub}
2. Berechne $P := H \circ H(k_{pub})$ und überprüfe, ob die ersten c-bits von P den Wert 0 haben, wenn nicht gehe zu 1.
3. Berechne die *NodeID* $ID := H(k_{pub})$

Dabei bezeichne $H(x)$ eine kryptographisch sichere *Hashfunktion*. Dieses Kryptopuzzle besitzt einen durchschnittlichen Generierungsaufwand von

$$T_{gen} \approx 2^{\frac{c}{2}} \cdot (T_{hash} + T_{key}) \in O(2^c)$$

und den Verifizierungsaufwand von

$$T_{verify} = T_{hash} \in O(1)$$

Der Wert c kann so gewählt werden, dass das Kryptopuzzle in einer bestimmten Zeit, beispielsweise in einer Stunde, auf einem durchschnittlich ausgestatteten Rechner gelöst werden kann. Über längere Sicht kann c erhöht werden, wenn sich die Rechenleistung gängiger Computer erhöht hat. Dies erfordert bei dem oben vorgestellten

Kryptopuzzle allerdings die Generierung einer neuen *NodeID*. In dem Entwurf dieser Arbeit wird ein Kryptopuzzle vorgestellt, bei welchem die Generierung der *NodeID* von der Lösung des Kryptopuzzles abgekoppelt wird.

Douceur kommt zu dem Schluss, dass durch die parallelisierbaren Kryptopuzzles und den linearen Zusammenhang zwischen Rechenleistung und der Anzahl der generierten *NodeID*s hier nur ein kleiner Schutz besteht. Allerdings lässt er außer Acht, dass für einen erfolgreichen Sybilangriff, ein bestimmter Prozentsatz an Knoten in das Netz eingebracht werden muss. Die Anzahl der Knoten im Netz steigt allerdings in der Bootstrapping-Phase superlinear in Abhängigkeit der Zeit. Dadurch muss der Angreifer stets neue Ressourcen einsetzen, um den Prozentsatz an bösartigen Knoten zu halten. Diese Anforderungen können in der Entstehungsphase eines Netzes durchaus exponentiell steigen.

Der Nachteil von Kryptopuzzles ist, dass sie bei der in dieser Arbeit angenommenen Generierung von *NodeID*s den Raum der möglichen öffentlichen Schlüssel stark einschränken. Aus diesem Grund muss darauf geachtet werden, dass gegebenenfalls die Anzahl der Bits des öffentlichen Schlüssels um dem Parameter c erweitert wird.

Eine weitere Möglichkeit den Sybilangriff zu erschweren beschreibt Jochen Dinger in [DH06]. Sein Ansatz basiert auf der Annahme, dass ein Angreifer nur selten an viele IP-Adressen kommt, deren Präfixe sich nicht ähneln bzw. gleich sind. Sie werden tatsächlich durch die zentrale Vergabestelle von IP-Adressen beschränkt (ICANN). Ob dies einen Angreifer wirklich hindert an ausreichend viele IP-Adressen zu gelangen, ist nicht Thema der Publikation. In der Arbeit wird es einem Knoten nur erlaubt dem Netz beizutreten, wenn der Präfix der IP-Adresse das Maximum der im Netz zugelassenen Präfixe nicht überschreitet. Dies wird erreicht, indem zu jedem Präfix ein Zähler, welcher dessen Vorkommen zählt, in der DHT gespeichert wird.

Dieser Ansatz scheint zunächst vielversprechend, allerdings setzt er voraus, dass die Zähler in der DHT auch sicher gespeichert werden. Weiterhin schränkt dieser Ansatz Teilnehmer, welche sich hinter einem NAT-Router befinden und dem P2P-System beitreten wollen, stark ein.

Um den Sybilangriff aufgrund der IP-Adresse zu erschweren, ist es sinnvoller, das Vorkommen einer IP-Adresse oder deren Präfix in der Routingtabelle zu beschränken. Zusätzlich können die Lookups der gleichen Beschränkung unterliegen. Das heißt, es wird bei jedem Pfad kontrolliert, dass ein Präfix nur einmal oder in einer beschränkten Anzahl vorkommt. Dieses Vorgehen vermeidet die aufwändige Datenspeicherung und schränkt die Anzahl ähnlicher IP-Adressen in gleicher Weise ein. Im Bezug auf Kademlia bleibt dessen reaktive Natur außerdem weiterhin erhalten.

Es lässt sich somit zusammenfassen: Kryptopuzzles erschweren den Sybilangriff ausreichend unter der Annahme, dass die Netzwerkgröße superlinear in Abhänigkeit der Zeit steigt. Das Ausschließen von ähnlichen IP-Adressen in der Routingtabelle und beim Lookup ist eine weitere *optionale* Möglichkeit den Sybilangriff noch weiter zu erschweren. Die vollständige Verhinderung eines Sybilangriffs ist dennoch nicht möglich. Insbesondere in der Entstehungsphase, in welcher sich nur wenige Knoten im Peer-to-Peer Netz befinden, ist es unmöglich zu verhindern, dass ein Angreifer schnell einen großen Prozentsatz der Knoten kontrollieren kann. Mit diesem Problem wird sich der nächste Abschnitt beschäftigen.

3.2.3 Der Sybilangriff während des Netzaufbaus

Jedes Peer-to-Peer Netz besteht in der Entstehungsphase, dem sogenannten *Bootstrapping*, nur aus sehr wenigen Knoten. In dieser Phase ist ein Sybilangriff sehr leicht möglich, da ohne weiteres in linearer Zeit genügend *NodeID*s generiert werden können. Eine rein dezentrale Generierung der *NodeID*s reicht hier offensichtlich nicht aus. Aus diesem Grund müssen *NodeID*s zentral generiert werden bis eine ausreichende Knotenanzahl (beispielsweise >1 Million) erreicht ist.

Eine Möglichkeit wäre, die *NodeID*s zusätzlich von einer vertrauenswürdigen Instanz, einer sogenannten *Certificate Authority* (CA), unterschreiben zu lassen. Dies wird auch in [SM02] vorgeschlagen. Eine ähnliche, aber vereinfachte Variante soll hier vorgestellt werden:

Jeder Peer-to-Peer Client wird mit einer Menge öffentlicher Schlüssel ausgeliefert. Diesen Schlüsseln wird allgemein vertraut. Ein *Supervisor*, welcher die privaten Schlüssel verwaltet, kann nun *NodeID*s signieren und sicherstellen, dass pro Teilnehmer nur wenige *NodeID*s vergeben werden. Um *NodeID*s zu signieren muss der *Supervisor* zunächst prüfen, ob der Teilnehmer auch in dem Besitz des privaten Schlüssels der *NodeID* ist. Dies kann er durch ein *Challenge-Response* Verfahren ermitteln. Beispielsweise kann er dem Knoten eine Zufallszahl übergeben, welche der Teilnehmer mit seinem privaten Schlüssel signieren muss. Um Replay-Angriffe zu vermeiden, wird diese Zufallszahl jeweils nur einmal benutzt. Man spricht hier allgemein von sogenannten *Nonces*.

Sollte einer der privaten Schlüssel des *Supervisors* bekannt werden, so kann der dazugehörige öffentliche Schlüssel gesperrt werden. Dies wird erreicht, indem der *Supervisor* einen gültigen privaten Schlüssel nutzt um eine „Sperrnachricht“ zu signieren und im Netz zu verbreiten. Jeder Knoten, welcher diese Nachricht erhält, löscht nun beide öffentliche Schlüssel: den des bekanntgewordenen privaten Schlüssels und den der „Sperrnachricht“. Damit ist sichergestellt, dass mit dem bekannt gewordenen privaten Schlüssel kein Missbrauch betrieben werden kann. Da es unwahrscheinlich ist, dass ein privater Schlüssel schon während der Aufbauphase bekannt wird, ist dies aber eher als Optimierung anzusehen.

Wurde die *NodeID* des Teilnehmers signiert, so kann es diese Signatur jeder Nachricht anhängen und damit beweisen, dass diese *NodeID* vertrauenswürdig generiert wurde. Hat das Netz eine ausreichende Größe erreicht, so kann auf diese „überwachte Signatur“ verzichtet werden. Um festzustellen wie groß das Netz ist, wird im Entwurfskaptiel dieser Arbeit ein Verfahren vorgestellt, mit welchem sich die Netzgröße ausreichend genau abschätzen lässt.

Der Vorteil dieses Verfahrens ist, dass der *Supervisor* nach dem Signieren der *NodeID* keinen zusätzlichen Wartungsaufwand mehr hat. Dieses Vorgehen ist für die zeitlich meist recht kurze Aufbauphase völlig ausreichend.

In dem nächsten Abschnitt werden Sicherheitsaspekte des Routings, basierend auf der in diesem Abschnitt vorgestellten Wahl der *NodeID*, vorgestellt.

3.3 Sicherheitsaspekte des Routings

Dieser Abschnitt befasst sich mit der Sicherheitsanalyse des Routings und der Lookup Operationen, welche für die Erreichbarkeit und Auffindbarkeit eines Teilnehmers

von großer Bedeutung sind. Dazu werden zunächst mögliche Angriffe und deren Gegenmaßnahmen bei Verwendung der zuvor analysierten *NodeID*s, diskutiert.

In dieser Arbeit soll eine DHT vorgestellt werden, welche auf Kademlia basiert. Die Analyse dieses Abschnitts bezieht sich aus diesem Grund immer auf das Kademlia Key-based Routing.

3.3.1 Angriffe auf das Routing

In diesem Unterabschnitt werden Angriffe und Gegenmaßnahmen auf das Routing Kademlias erläutert. Es stellt sich heraus, dass das Kademlia Protokoll sehr resistent gegenüber einer Vielzahl von Angriffen ist.

Eclipseangriff: Durch gezielte Wahl der *NodeID* können Knoten derart in der Netztopologie positioniert werden, dass mehrere andere Knoten aus dem Netz ausgeblendet werden. Dazu muss der Angreifer wissen, welche *NodeID*s diese Knoten referenzieren oder kennen. Abgesehen davon, dass sich die *NodeID*s in dieser Arbeit nicht frei wählen lassen, sind diese Informationen nicht vorhersehbar, denn die Kademlia-Routingtabelle referenziert nur Knoten, wenn ein k-Bucket nicht voll sein sollte. Dieses Verhalten ist für einen Angreifer nicht vorhersehbar. Deshalb ist das Kademlia-Routing resistent gegen diesen Angriff.

Churnangriff: Bei einem Churnangriff betritt ein Angreifer mit vielen Knoten das Netz und verlässt es nach kurzer Zeit wieder. Dieser Vorgang wird so lange wiederholt, bis sich das Netz durch seine eigenen Stabilisierungsmaßnahmen lahm legt oder partitioniert. Das Kademlia-Netz ist auch hier resistent, da ein Netzbeitritt oder das Verlassen des Netzes keine Stabilisierungsmaßnahmen benötigt. Kurzlebige Knoten erreichen nie die nötige Aufmerksamkeit des Netzes, um mit diese, Angriff Erfolg zu haben.

Rückgabe bösartiger Routinginformationen: Ein bösartiger Knoten kann auf eine Routinganfrage entweder keine Knoten zurückgeben oder aber ausschließlich weitere bösartige Knoten zurückgeben. Werden keine Knoten zurückgegeben, so verläuft dieser Angriff im Sand, denn der Angreifer wird als inaktiver Knoten angesehen und nach kurzer Zeit aus der Routingtabelle entfernt. Viel effektiver ist es für den Angreifer ausschließlich verwandte bösartige Knoten zurückzugeben, um zu versuchen das Opfer in ein paralleles Netz zu ziehen. Diesem Angriff kann durch einen parallelen, disjunkten Mehrwege-Lookup entgegengewirkt werden, wie später bei der Analyse deutlich wird.

3.3.2 Kademlia Routing

Kademlia ist ein reaktives Protokoll. Jeder Knoten kümmert sich um seine eigene Präsenz im Netz, welches damit sehr lose verbunden ist. Diese Eigenschaft hat bereits viele Vorteile bezüglich der Sicherheit. So lassen sich nur schwer Einträge in der Routingtabelle gezielt platzieren und jeder Knoten kann selbst entscheiden, ob er einen bestimmten Knoten in die Routingtabelle aufnimmt oder nicht.

Eine der Konstruktionsregeln ist, dass jeder Knoten in die Routingtabelle eingefügt wird, wenn der entsprechende Bucket nicht voll sein sollte. Diesen Umstand könnte sich ein Angreifer zu nutze machen, wenn er gezielt diese Buckets versucht zu füllen.

Allerdings sind durch das Kryptopuzzle die nah bei einem Knoten liegenden Einträge gut geschützt, denn für einen Angriff müsste eine *NodeID* generiert werden, welche dem Präfix des Opfers entspricht. Dieser Aufwand steigt exponentiell mit der Länge des Präfixes. Im Gegenzug ist die Wahrscheinlichkeit gering, dass ein Knoten einen langen Präfix mit einem anderen teilt, so dass genügend weit entfernte Knoten zum Füllen der Buckets vorhanden sind und die Buckets damit nur beim Netzbeitritt für kurze Zeit leer sind. Während dieser Zeit ist es sinnvoll lediglich aktiv befragte Knoten in die „höheren“ k-Buckets aufzunehmen.

Die Routingtabelle wird durch eingehende Anfragen und Antworten gefüllt. Auch die durch FIND_NODE gefundenen Knoten werden im ursprünglichen Design der Routingtabelle hinzugefügt. Dies kann dazu führen, dass ein Angreifer einen Lookup mit ausschließlich fehlerhaften IP-Adressen zurückgibt, um das Netz zu schwächen. Deshalb darf ein solcher Knoten erst nach einem signierten Ping in die Routingtabelle aufgenommen werden.

3.3.3 Erfolgswahrscheinlichkeiten des Knotenlookups

Kademlia ist wie CAN *k-node-connected*. Das heißt der durch Kademlia erzeugte Graph enthält mit hoher Wahrscheinlichkeit je k knotendisjunkte Pfade zwischen paarweise verschiedenen Knoten. Durch die Wahl der *NodeID* kann angenommen werden, dass alle bösartigen Knoten in den Routingtabellen gleichverteilt sind, sodass jeder zufällig gewählte Knoten aus der Routingtabelle mit der Wahrscheinlichkeit m bösartig ist.

Dadurch ist die Wahrscheinlichkeit, dass sich unter den Knoten auf einem Pfad der Länge h kein bösartiger Knoten befindet $(1-m)^h$. Werden d knotendisjunkte Pfade zwischen zwei festen Knoten gleichzeitig gewählt, so ist die Wahrscheinlichkeit das ein Pfad ohne bösartigen Knoten gefunden wird:

$$\Pr[\text{„Knotenlookup erfolgreich“}] = P'_K(m,h,d) := 1 - \left(1-(1-m)^h\right)^d$$

Abbildung 3.1 zeigt diese Wahrscheinlichkeit mit unterschiedlichen Parametern. Sie zeigt, dass ab etwa 15% bösartiger Knoten die Zahl der Hops erheblich mehr Einfluss auf das Gelingen eines Lookups hat als die Anzahl der disjunkten Pfade. Auch wird der Vorteil, welcher durch die disjunkten Pfade erzielt wird, für große k immer geringer. Als sinnvolle Werte können somit beispielsweise $d = 8$ und $h = 1 \ldots 6$ angenommen werden. In diesem Bereich ist sichergestellt, dass für $m \leq 0.25$ immer noch 80% der Lookups gelingen.

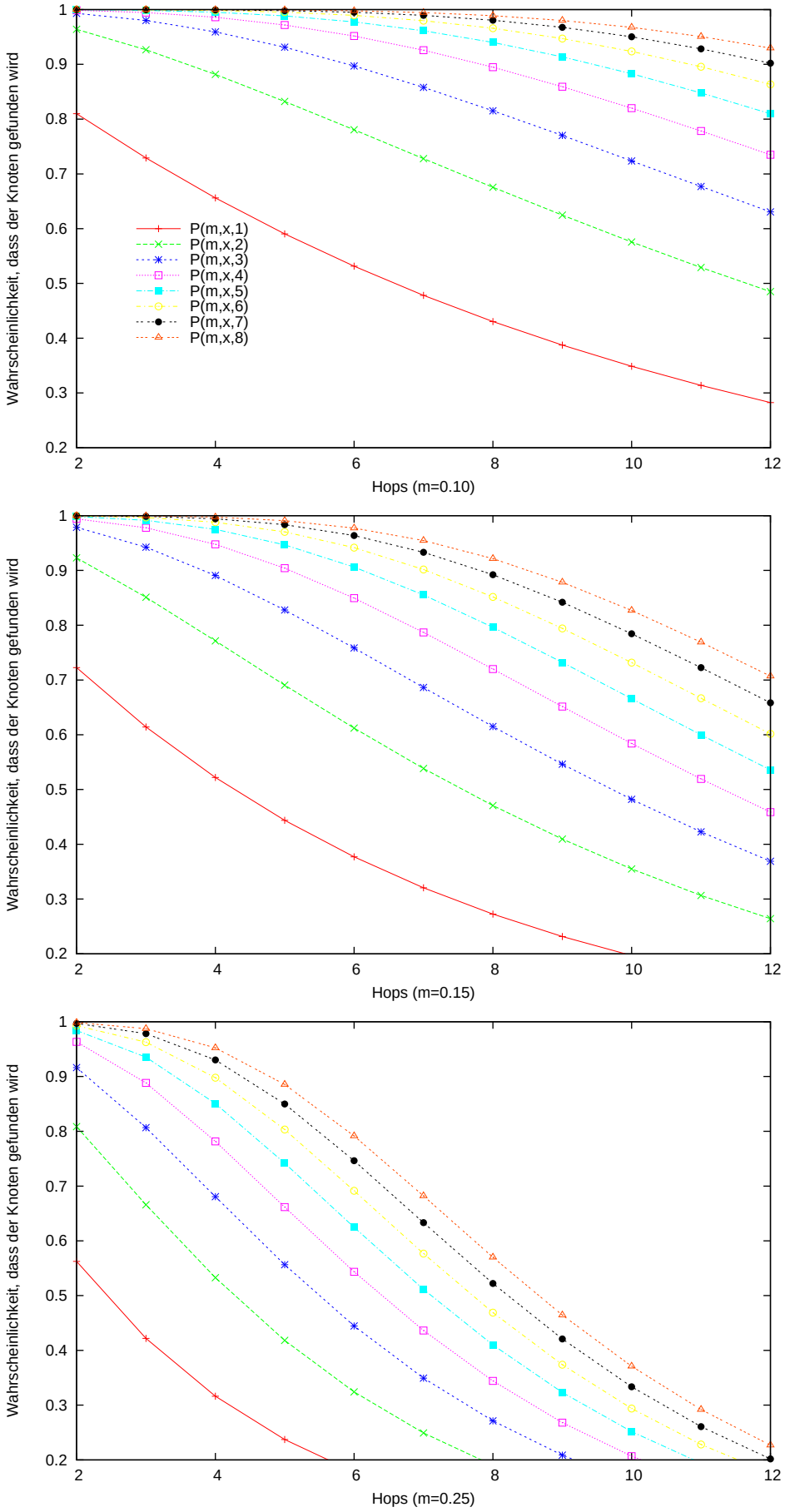


Abbildung 3.1: Wahrscheinlichkeit, dass ein Knotenlookup mit $m = 0.10, 0.15, 0.25$ und $d = 1, \ldots, 8$ gelingt

Diese Wahrscheinlichkeiten gelten nur für gleichlange Pfade. Bei hyperkubischen Topologien, wie es bei Kademlia der Fall ist, ist die Hopverteilung normalverteilt. Darum wird die Wahrscheinlichkeit für eine Hopverteilung (h_x) verallgemeinert:

Sei $(h_x) \in [0,1]$, $x \in \mathbb{N}$ die normalisierte Häufigkeit der Pfade mit x Hops, m der Anteil bösartiger Knoten, n die Anzahl der Replikate bzw. Nachbarn, d die Anzahl der knotendisjunkten Pfade und c eine hopkorrigierende Konstante. Dann gilt:

$$\Pr[\text{„Knotenlookup ist erfolgreich“}] =$$

$$P_K := \sum_{i=1}^{|(h_x)|} \left(h_i \cdot \left(1 - \left(1 - (1-m)^{i-c} \right)^d \right) \right)$$

Die Konstante c wird im Falle des Knotenlookups $c = 1$ gewählt, da davon ausgegangen werden kann, dass der letzte Hop zu einem gutartigen Knoten führt. Im Falle des Datenlookups wird $c = 0$ gesetzt, da der letzte Hop zu einem bösartigen Knoten führen kann.

Diese Analyse gilt für das gezielte Auffinden eines Knotens. Für die Replikation von Daten, welche später in Abschnitt 3.4.3 diskutiert wird, wird zusätzlich ein Lookup der n Nachbarschaftsknoten eines Hashwerts benötigt. Im Kapitel 4 wird durch eine Ergänzung der Kademlia Routingtabelle gezeigt, dass lediglich ein Knoten aus der Nachbarschaft gefunden werden muss, um die gesamte Nachbarschaft eines Hashwerts zu kennen. Deshalb gelten diese Wahrscheinlichkeiten auch für den Nachbarschaftslookup.

3.4 Sicherheitsaspekte der verteilten Hashtabelle

Die verteilte Hashtabelle soll in dieser Arbeit eine sichere Ablage einer *NodeID* unter einem lesbaren Namen ermöglichen. Damit ist es möglich eine Art dezentralen DNS[2] zu realisieren um einen P2P-VoIP Teilnehmer nicht unter einem 160 Bit Schlüssel suchen zu müssen.

Im ersten Teil werden zunächst mögliche Angriffe auf die Datenspeicherung analysiert und erläutert. Dann folgt eine Analyse der möglichen Replikationsräume und deren Eignung zur Realisierung redundanter Speicherung von Daten.

Im letzten Teil wird die Integriät durch die Replikation von Daten mit Hilfe von Mehrheitsentscheidungen in einem bösartigen Umfeld analysiert.

In dieser Arbeit wird davon ausgegangen, dass ein Schlüssel-Wert Paar immer unter dem Hashwert des Schlüssels gespeichert wird. Auf die Speicherung des echten Schlüssels wird aus Sicherheitsgründen explizit verzichtet. Es wird weiterhin davon ausgegangen, dass eine Kollision von Hashwerten einer kryptographischen *Hashfunktion* nahezu unmöglich ist.

Um die nächsten Abschnitte zu verstehen, wird folgendes definiert: Daten werden von dem Knoten gespeichert, dessen *NodeID* dem Schlüssel am nächsten liegt. Dieser Knoten heißt *designierter Knoten* eines Schlüssel-Wert Paares. Ein Knoten, welcher ein Replikat dieses Paares speichert heißt *replizierender Knoten*. Es wird weiterhin davon ausgegangen, dass jeder Schlüssel und die damit verbundenen Daten genau einmal auf jedem Knoten gespeichert werden können. Änderungen oder das Löschen von Daten werden nur dem Autor des Schlüssel-Wert-Paares gestattet.

[2]Domain Name Service

3.4.1 Angriffe auf die Datenspeicherung

In diesem Abschnitt werden Angriffe auf die Datenspeicherung in der verteilten Hashtabelle, sowie ihre Gegenmaßnahmen diskutiert. Dadurch, dass mittels der *NodeID* die Integrität einer Nachricht überprüft werden kann, brauchen Angriffe, welche durch das Peer-to-Peer Netz sowie durch das Underlay möglich werden, hier nicht diskutiert werden.

Ein Knoten, welcher ein Schlüssel-Wert-Paar ablegen will, heißt im folgenden *Autor*.

Verändern, Löschen, Abweisen von Daten: Jeder Knoten kann Daten löschen, verändern oder abweisen. Diesem Angriff kann durch Replikation vorgebeugt werden. Wird ein Schlüsselpaar über n-Knoten repliziert, so kann durch eine Mehrheitsentscheidung bei $\frac{n}{2} - 1$ modifizierten, gelöschten oder veränderten Daten immernoch bestimmt werden, welche Daten ursprünglich abgelegt wurden.

Angriff über die Suche des designierten Knotens: Sollen Daten abgelegt werden, so geschieht dies unter dem Hashwert des Schlüssels. Nach diesem Hashwert muss gesucht werden damit das Schlüssel-Wert Paar auf dem designierten Knoten gespeichert werden kann. Angenommen der designierte Knoten ist gutartig, so kann jeder Knoten auf dem Pfad zu dem designierten Knoten behaupten, dass er sich in der unmittelbaren Nachbarschaft des Knotens befindet. Die Daten werden damit auf einem falschen Knoten abgelegt. Zusätzlich kann sich der Angreifer den Hashwert zu nutze machen, indem er in einem zweiten Schritt, ein anderes Schlüssel-Wert-Paar auf dem richtigen designierten Knoten ablegt. Damit wurde die Datenspeicherung für den Autor der Daten für immer verwehrt. Abbildung 3.2 zeigt diesen Angriff in einem Sequenzdiagramm.

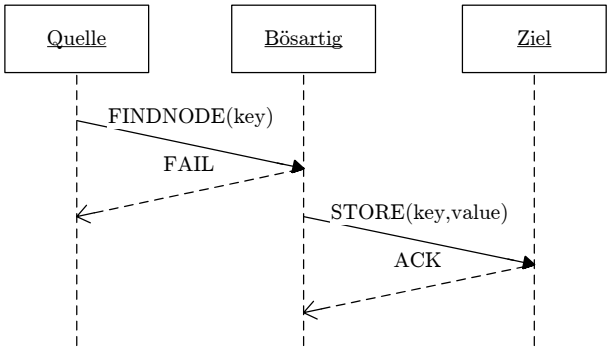


Abbildung 3.2: Sequenzdiagramm eines Angriffs über die Suche des designierten Knotens

Replikationsangriff: Hier gelten die gleichen Probleme und Maßnahmen wie sie bereits aus dem Angriff über die Suche des designierten Knotens bekannt sind. Hinzu kommt, dass, falls einer der Knoten in dem Replikationsraum bösartig ist, dieser bei allen anderen Knoten seine Daten unter dem Hashwert speichern kann. Damit schlägt die Mehrheitsentscheidung über die Replikate fehl, falls es ihm gelingt auf der Mehrheit der Knoten seine Daten zu speichern.

Um diesen Angriff zu vereiteln, wird zunächst der Wert auf allen replizierenden Knoten gespeichert und in einer zweiten Runde der genaue Schlüssel bekannt gegeben. Dies verhindert wirkungsvoll den Replikationsangriff. Abbildung 3.3 zeigt diesen Angriff in einem Sequenzdiagramm.

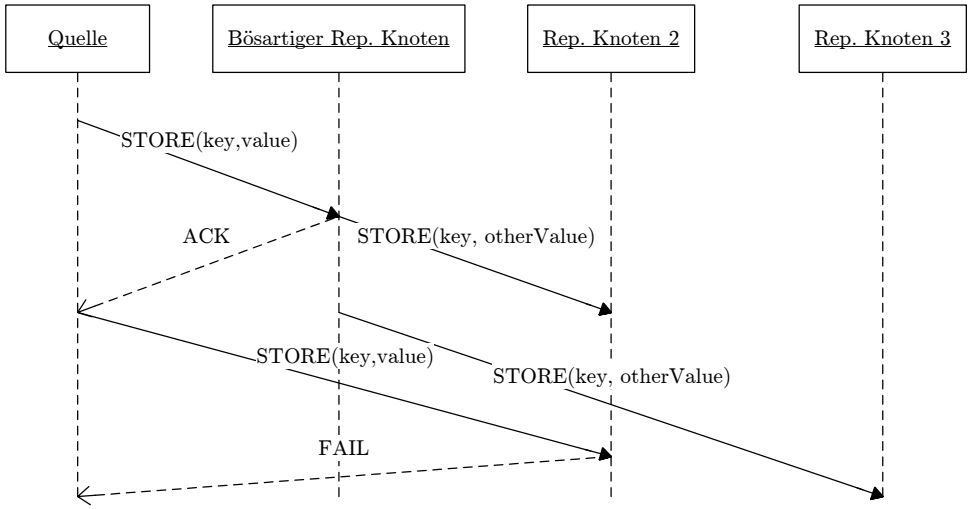


Abbildung 3.3: Sequenzdiagramm eines Replikationsangriffs

Beeinflussung von Mehrheitsentscheidungen: Ein Angreifer kann seine *NodeID*s so wählen, dass er alle Knoten hält, welche für einen bestimmten Schlüssel und dessen Replikate zuständig sind. Dies setzt voraus, dass der Angreifer seine *NodeID* frei wählen kann. Bei den hier vorgestellten *NodeID*s ist dies nur sehr schwer möglich. Falls der Angreifer würde versuchen die *NodeID*s durch einen *brute-force-Angriff* zu generieren, ist die Wahrscheinlichkeit, dass er einen Knoten der einem gegebenen Schlüssel am nächsten liegt, $\frac{1}{N-1}$ (N ist die Anzahl der Knoten im Netz). Dies ist intuitiv klar, denn mit N Knoten gibt es $N-1$ Zwischenräume, in welchen die *NodeID* liegen kann. Somit ist der Erwartungswert der Anzahl der zu generierenden *NodeID*s $E(t) \approx k \cdot (N-1)$ bei k-Replikaten.

Dieser Aufwand ist weitaus höher als der für einen erfolgreichen Sybilangriff benötigte. Genauere Abschätzungen und Analysen dieses Sachverhalts findet man in [DC05].

Insertion Denial-of-Service: Ein Angreifer kann beliebig viele Daten in der DHT speichern. Damit kann er Knoten überschwemmen, bis diese den Dienst einstellen. Um dies zu vermeiden, lassen sich wie in [Bac] erneut Kryptopuzzles einsetzen. Dadurch wird zwar der Insertion Denial-of-Service Angriff erschwert, aber nicht ausgeschlossen.

Deshalb muss ein Knoten, welcher eine Aufforderung zum Speichern von Daten bekommt, überprüfen, wieviele Daten der Autor bereits in der Hashtabelle abgelegt hat. Außerdem muss der Autor zunächst in seiner Nachbarschaft bekanntgeben, dass er einen bestimmten Schlüssel speichern will.

Erst wenn mehr als die Hälfte der Nachbarn zustimmen, dass der Schlüssel gespeichert werden darf, gilt die Speicherung eines Schlüssel-Wert-Paar durch

die Nachbarn als autorisiert. Jeder Knoten der eine Aufforderung zum Speichern von Daten bekommt, kann über die Nachbarschaft des Autors prüfen, ob die Speicherung autorisiert wurde.

Das Lösen des Kryptopuzzles stellt die Entschädigung für die benötigte Bandbreite des Netzes dar. Deshalb kann dieses Kryptopuzzle deutlich einfacher als das der *NodeID* gewählt werden. Eine durchschnittliche Lösungszeit von 2 Stunden erscheint als angemessen für den Überprüfungsaufwand über die Nachbarschaft des Autors. Dieses Verfahren wird in Abschnitt 4.3.1 ausführlicher erläutert.

Natürlich sind zusätzlich zu den hier genannten Angriffen auch bis heute unbekannte Angriffe möglich. Die hier beschriebenen Szenarien sollten allerdings die meisten „offensichtlichen“ Angriffe abdecken.

Um die Integrität sowie die Verfügbarkeit von Daten in einer verteilten Hashtabelle sicherzustellen, werden Daten über mehrere Knoten repliziert. Auf welchen Knoten diese Replikate abgelegt werden, soll im nächsten Abschnitt diskutiert werden.

3.4.2 Replikationsräume

Replikationsräume sind Räume, welche alle möglichen Hashwerte enthalten, an denen Replikate zu einem gegebenen Schlüssel S abgelegt werden können. Zwei dieser Räume sollen in dieser Arbeit diskutiert werden:

Ablage im Raum der Hashrunden

Die Daten werden über Hashrunden des Schlüssels verteilt. Dieser Raum wird hiert nicht benutzt und soll nur der Vollständigkeit halber diskutiert werden. Er hat den Vorteil, dass einem Replikationsangriff vorgebeugt werden kann indem zunächst die Hashrunden berechnet werden und dann rückwärts die Daten auf den jeweiligen replizierenden Knoten abgelegt werden. Im letzten Schritt erhält der designierte Knoten den Hashwert der ersten Hashrunde.

Die Idee dabei ist, dass eine bessere Verteilung im Zahlenraum der *NodeID*s erreicht wird. Dadurch wird es dem Angreifer erschwert Knoten in jeder Nachbarschaft zu positionieren. Dies macht allerdings nur Sinn, wenn der Angreifer seine *NodeID* frei oder sehr leicht in einem zusammenhängenden Nachbarschaftscluster wählen kann. Durch die hier behandelte Wahl der *NodeID* ist dies nicht möglich.

Weiterhin ist durch diese Maßnahme für jedes Replikat ein Lookup erforderlich, was den Bandbereitenbedarf unnötig in die Höhe treibt.

Ablage im Raum der Nachbarschaftsknoten

Die Daten werden in der Nachbarschaft des designierten Knotens gespeichert. Dieser Raum wird in der vorliegenden Arbeit durchgängig benutzt. Er bietet den Vorteil, dass nur ein Lookup der Nachbarschaft benötigt wird um eine Mehrheitsentscheidung zu fällen. Zwischen den Nachbarn kann eine enge Beziehung gepflegt werden, so dass fehlerhafte Daten erkannt und korrigiert werden können.

Es ist klar, dass der Nachbarschaftsraum am idealsten geeignet ist Replikate zu speichern. Im nächsten Abschnitt wird die Sicherstellung der Integrität der replizierten Daten analysiert.

3.4.3 Integrität von Daten durch Replikation

In diesem Abschnitt wird die Integrität von replizierten Daten analysiert. Um modifizierte Daten von den ursprünglichen Daten unterscheiden zu können, werden Mehrheitsentscheidungen benötigt:

Sind mehr als die Hälfte der Replikate durch bösartige Knoten in der gleichen Weise modifiziert worden, so gehen die ursprünglich gespeicherten Daten verloren, denn die Mehrheitsentscheidung würde zu Gunsten der bösartigen Daten ausfallen. Dies ist der schlimmste anzunehmende Fall, denn werden die Daten jeweils in einer anderen Weise modifiziert, wird zu Gunsten der ursprünglichen Daten entschieden.

Da bösartige Knoten durch die Wahl der *NodeID* zufällig verteilt sind, ist die Wahrscheinlichkeit, dass eine Mehrheitsentscheidung zu Gunsten der ursprünglichen Daten ausfällt ein Bernoulliexperiment mit n Versuchen (n ist die Anzahl der Nachbarn eines Knotens und gleichzeitig die Anzahl der Replikate). Damit gilt:

Sei m der Anteil der bösartigen Knoten im Netz, dann ist die Wahrscheinlichkeit, dass eine Mehrheitsentscheidung zu Gunsten der ursprünglichen Daten ausfällt, über eine Menge mit n Knoten gegeben durch:

$$\Pr[\text{„Mehrheitsentscheidung zu Gunsten der ursprünglichen Daten“}] =$$

$$P_B(m,n) := 1 - \sum_{i=\frac{n}{2}}^{n} \binom{n}{i} m^i \, (1-m)^{n-i}$$

Die Abweichung vom Erwartungswert $E(X) = m \cdot n$ ist für $n > 48$ ausreichend klein, so dass die Anzahl der bösartigen Knoten unter den n Knoten näherungsweise mit $m \cdot n$ angenommen werden kann. Verdeutlicht wird dies auch in Abbildung 3.4.

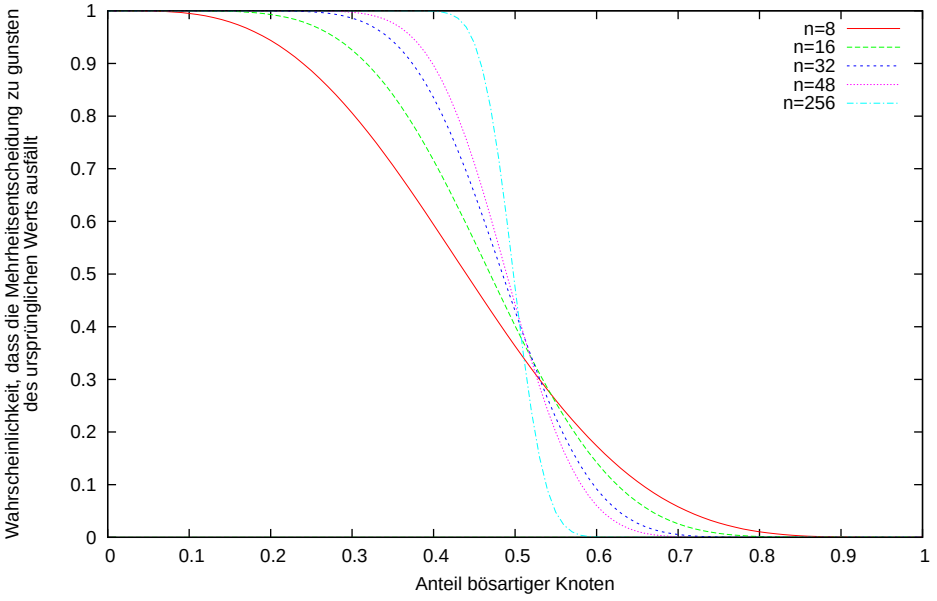


Abbildung 3.4: Mehrheitsentscheidung über die Replikate mit $n = 8, 16, 32, 48, 64$

Im Gegensatz zu einem Lookup eines Knotens bei dem nach einer konkreten *NodeID* gesucht wird, muss bei dem Lookup von Daten die Nachbarschaft des designierten

Knotens gefunden werden. Der Unterschied besteht darin, dass die gesuchte *NodeID* auf Authentizität überprüft werden kann, die Nachbarschaft aber nur sehr schwer zu verifizieren ist, denn jeder bösartige Knoten kann behaupten er befinde sich in der Nachbarschaft der Daten. Weiterhin kann nicht davon ausgegangen werden, dass der designierte Knoten gutartig ist. So muss der Knotenlookup um einen Lookup der Nachbarschaft erweitert werden. Für ihn gelten die gleichen Erfolgswahrscheinlichkeiten, mit dem Unterschied, dass bei gleicher Netzwerkgröße für gewöhnlich ein Hop weniger benötigt[3] wird. Dieser Lookup der Nachbarschaft im Entwurf dieser Arbeit vorgestellt. Betrachtet man gleichzeitig die Erfolgswahrscheinlichkeit der Mehrheitsentscheidung, so gilt:

$$\Pr[\text{„Datenlookup erfolgreich“}] = P_D(m, n, h, d) := P_K(m, h, d) \cdot P_B(m, n)$$

Damit ergibt sich, unter der Annahme einer konstanten Pfadlänge, die Funktion:

$$P_D(m, n, h, d) := \left(1 - \left(1 - (1-m)^h\right)^d\right) \cdot \left(1 - \sum_{k=\frac{1}{2}n}^{n} \binom{n}{k} m^k (1-m)^{n-k}\right)$$

Diese Funktion gibt die Erfolgswahrscheinlichkeit an, falls Nachbarschaftslookup und Mehrheitsentscheidung stochastisch unabhängig sind. In Wirklichkeit sind sie das nicht immer, denn jeder Lookup berücksichtigt auch mindestens d Knoten aus der Nachbarschaft. Allerdings ist die Funktion genau genug, um hier als Abschätzung zu genügen.

In Abbildung 3.5 sieht man nun die Wahrscheinlichkeit P_D für $n \in \{8, 16, 32\}$ mit angenommenen $h = 6$ und $d = 8$. Es wird deutlich, dass die zuvor für sinnvoll erachteten Anzahl an Replikaten für $n > 16$ nur sehr gering für das Gesamtergebnis des Lookups von Bedeutung ist. Aus diesem Grund wird für doe folgenden Untersuchungen die Zahl der Replikate auf $n = 16$ begrenzt.

[3] Dies gilt, da der Hop zum designierten Knoten wegfällt

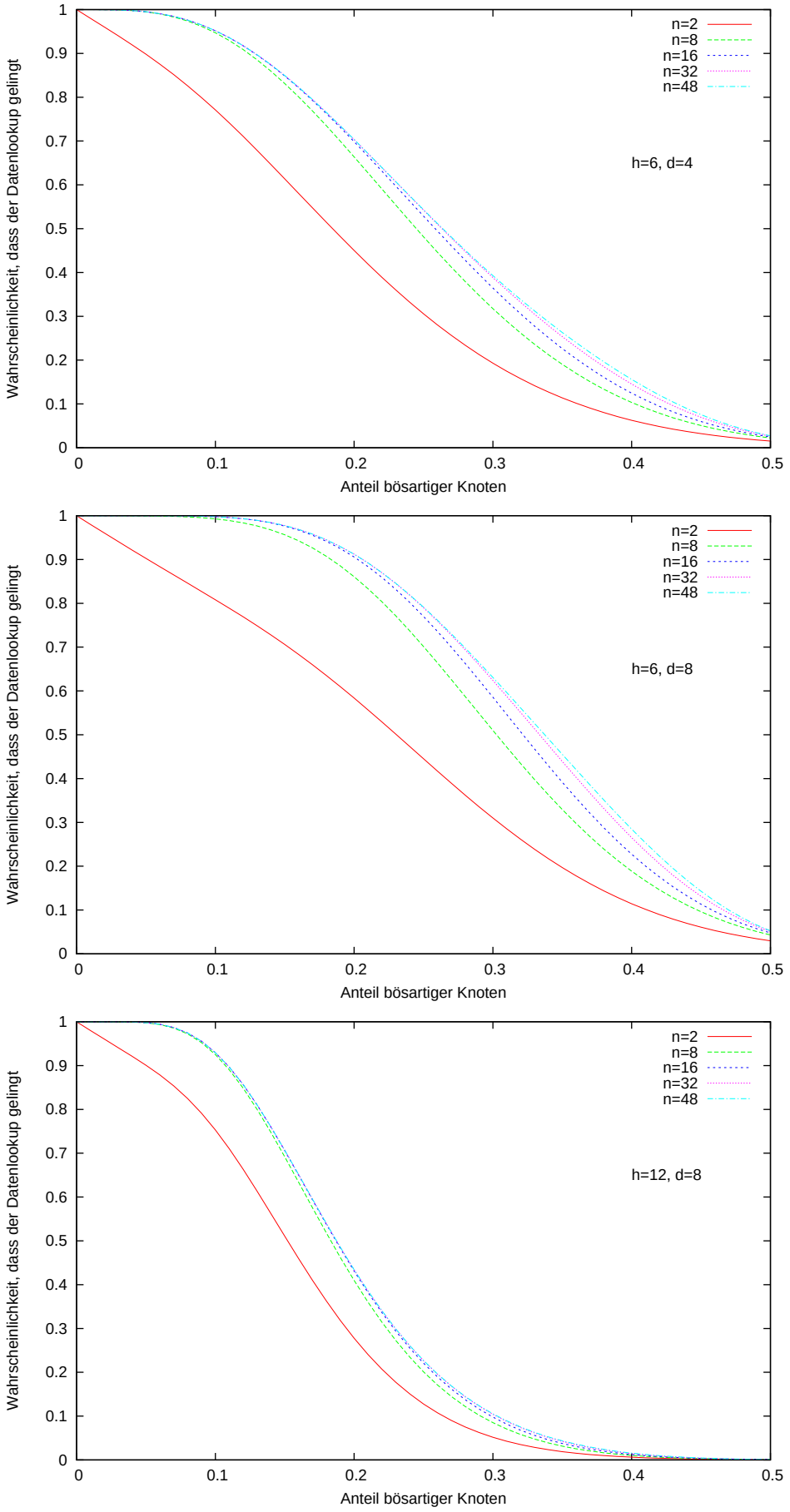


Abbildung 3.5: Wahrscheinlichkeit, dass ein Datenlookup gelingt für $n \in \{8, 16, 32\}$, $h = 6$, $d = 4$ und $h = 6, 12$, $k = 8$

4. Entwurf

Die Architektur des hier vorgestellten Peer-to-Peer Systems besteht aus 4 Teilen: Sicherheitsschicht, Key-based Routing (Kademlia), verteilte Hashtabelle und Anwendungschicht. Im Folgenden werden die Aufgaben jeder Schicht kurz vorgestellt:

Sicherheitsschicht: Die Sicherheitsschicht ist für die Authentifizierung der Knoten und die Integritätsprüfung der Nachrichten zuständig. Besteht eine Nachricht die Sicherheitsprüfung nicht, so wird sie mit einem Anonym-Flag gekennzeichnet und weitergereicht. Außerdem übernimmt die Sicherheitsschicht die Verwaltung von Timeouts und *Remote-Procedure-Calls* (RPCs).

Key-based Routing: Das Key-based Routing dient dem sicheren Auffinden von IP-Adresse und Port zu einer gegebenen *NodeID* oder von einer Nachbarschaft eines Hashwertes. In dieser Arbeit wird Kademlia als Basis des Key-based Routings verwendet.

Verteilte Hashtabelle: Die verteilte Hashtabelle übernimmt die Speicherung, Replikation und Sicherung von Daten. An dieser Stelle werden Mehrheitsentscheide gefällt und überprüft, ob Daten von einem Knoten abgelegt werden dürfen.

Anwendungsschicht: Die Anwendungsschicht besteht aus einem Namensdienst, welcher eine URI[1] auf eine *NodeID* abbildet, sowie einem Voice-over-IP SIP-Proxy, welcher Anrufe an *User Agents* weitergibt.

Abbildung 4.1 zeigt dieses Modell noch einmal in der Übersicht.

4.1 Sicherheitsschicht

Die Sicherheitsschicht überprüft eingehende und signiert ausgehende Nachrichten. Dies geschieht transparent für die höheren Schichten. Bei ausgehenden Nachrichten wird von den darüberliegenden Schichten, wie dem Key-based Routing und der verteilten Hashtabelle, lediglich angegeben, ob die Nachricht anonym oder signiert

[1] Universal Resource Identifier

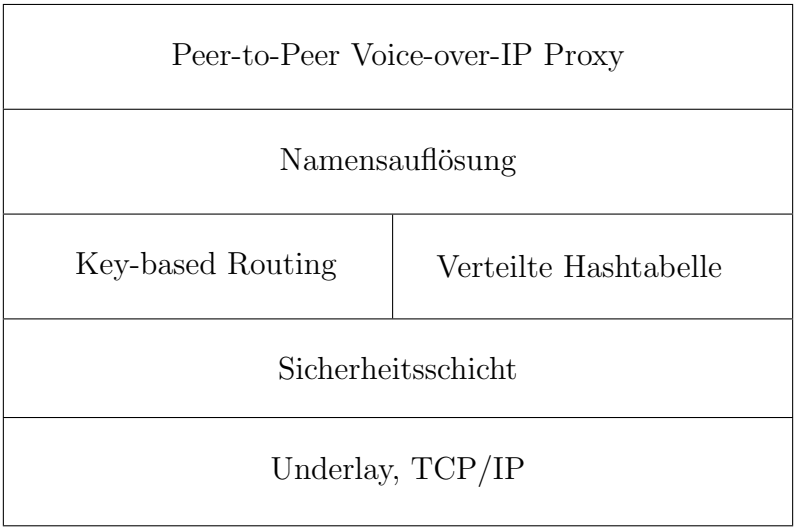


Abbildung 4.1: Architektur des Peer-to-Peer Systems

versendet werden soll. Umgekehrt wird bei eingehenden Nachrichten die Signatur überprüft und das Ergebnis der Prüfung an die höheren Schichten weitergeleitet.

Signaturen enthalten in dieser Arbeit immer auch den öffentlichen Schlüssel mit welchem die Signatur generiert wurde. Es werden unterschieden zwei unterschiedliche Signaturtypen unterschieden:

Schwache Signatur: Die schwache Signatur sichert nicht den gesamten Inhalt der Nachricht, sondern lediglich *IP-Adresse*, *Port* und *Zeitstempel*. Der Zeitstempel gibt an, bis zu welchem Zeitpunkt diese Signatur gültig ist. Dies verhindert Replay-Angriffe, falls eine andere Person, beispielsweise durch eine Neueinwahl und dynamisch zugewiesener Adresse, in den Besitz der IP-Adresse gelangt. Der Zeitstempel kann grob gewählt werden, sodass Abweichungen von Systemuhren zu keinem frühzeitigen Erreichen des Zeitstempels führen. Dadurch dass sich IP-Adresse und Port selten ändern, muss diese Signatur nur kurz vor dem Überschreiten des Zeitstempels aktualisiert werden. Diese Signatur wird benutzt, um `FIND_NODE` Anfragen zu signieren, damit der angefragte Knoten sicher weiß, dass die *NodeID* zu der entsprechenden IP-Adresse und Port gehört. Da `FIND_NODE` Nachrichten die häufigsten Nachrichten im Netz sind, ist diese Signatur sehr gut für mobile Endgeräte geeignet. Sie lassen sich einfach im Voraus auf einem leistungsstärkeren Rechner berechnen und dann auf ein mobiles Endgerät übertragen.

Starke Signatur: Die starke Signatur sichert den gesamten Inhalt einer Nachricht. Dies ist sinnvoll, wenn Nachrichten vor einem *Man-in-the-Middle* Angriff geschützt werden sollen. Hierzu gehören Nachrichten der Datenspeicherung und des Aufbaus von Sitzungen. Replay-Angriffe werden durch die in RPCs enthaltenen Nonces vermieden.

Durch diese zwei Signaturtypen werden Knoten über ihre *NodeID* authentifiziert und die Integrität der Nachrichten sichergestellt. Um einen Sybilangriff zu erschweren, müssen die Signaturen mit einem Kryptopuzzle versehen oder von einer vertrauenswürdigen Instanz unterschrieben werden:

Überwachte Signatur: Wurde der öffentliche Schlüssel zusätzlich von einer vertrauenswürdigen Instanz unterschrieben, so heißt diese Signatur *überwachte Signatur*. Wie in Abschnitt 3.2.3 diskutiert, wird diese Signatur nur benötigt, um den Sybilangriff zu erschweren, wenn das Netz eine geringe Zahl (<1Mio.) Knoten aufweist. In Abschnitt 4.2.5 wird ein Verfahren vorgestellt, welches erlaubt die Netzgröße abzuschätzen. Mit Hilfe dieser Abschätzung lässt sich dann entscheiden, ob diese Signatur überprüft wird oder nicht.

Kryptopuzzle Signatur: In Abschnitt 3.2.2 wurde bereits ein Kryptopuzzle und dessen Eigenschaften vorgestellt. Dieses Kryptopuzzle jedoch einen Nachteil: Steigen die Anforderungen an das Kryptopuzzle, so muss ein neuer Schlüssel generiert werden. Dies kann umgangen werden, indem *zusätzlich* ein Kryptopuzzle verwendet wird, welches von der Generierung des Schlüsselpaars abgekoppelt ist. Dazu wird zunächst der Hashwert über den öffentlichen Schlüssel, $ID := H(s_{publ})$, gebildet. Dieser Hashwert wird solange mit einem zufälligen Wert X mit Hilfe der XOR-Operation, $P := X \oplus ID$, verknüpft, bis die ersten c-Bits von $H(P)$ Null sind. So wurde das Kryptopuzzle durch X gelöst. ID und X dienen nun als Beweis, dass das Kryptopuzzle gelöst wurde. Abbildung 4.2 zeigt das statische und das dynamische Kryptopuzzle im Vergleich. Auf das statische Kryptopuzzle kann nicht verzichtet werden, denn es wird benötigt um sicherzustellen, dass die freie Wahl der *NodeID* und der damit verbundene mögliche Eclipseangriff erschwert werden.

Da für das Key-based Routing eine Vielzahl von kleinen Nachrichten signiert werden, stellt sich noch die Frage, wie groß solche Signaturen sind und welches Signaturverfahren am besten für diese Nachrichten geeignet ist. Wird von den bekannten Signaturverfahren DSA und ECDSA ausgegangen, so lassen sich die Signaturlängen wie folgt berechnen:

	DSA (1024 bit)	ECDSA (162 bit)
Kryptopuzzle Signatur	188 Bytes	84 Bytes
Überwachte Signatur	336 Bytes	128 Bytes
Überwachte Kryptopuzzle Signatur	356 Bytes	148 Bytes

Bei der Berechnung wurde angenommen, dass die Kurvenparameter der elliptischen Kurve in einem Byte kodiert werden, welches eine der Kurvendomänenparameter der NIST auswählt. Eine ECDSA-Signatur inklusive öffentlichem Schlüssel hat damit die Größe von (163+163+162) Bit = 61 Bytes. In der obigen Tabelle wurde deshalb mit den Kurvenparametern eine Größe von 64 Bytes angenommen. Eine DSA-Signatur besitzt die Größe von 168 Bytes (320-Bit Signatur + 1024-Bit öffentlicher Schlüssel). Da diese Signaturen jeder Nachricht angehängt werden und diese meist sehr klein sind, ist hier der ECDSA-Algorithmus eindeutig zu bevorzugen.

4.2 Key-based Routing

Um die in der Analyse besprochenen Sicherheitsmerkmale umzusetzen, muss das Key-based Routing von Kademlia um eine Nachbarschaftsverwaltung und einen Mehrwege-Lookup erweitert werden. Zusätzlich muss mit hoher Wahrscheinlichkeit sichergestellt sein, dass jeder Nachbar symmetrisch k' andere Nachbarn kennt und die Nachbarschaftsbeziehungen regelmäßig gepflegt werden.

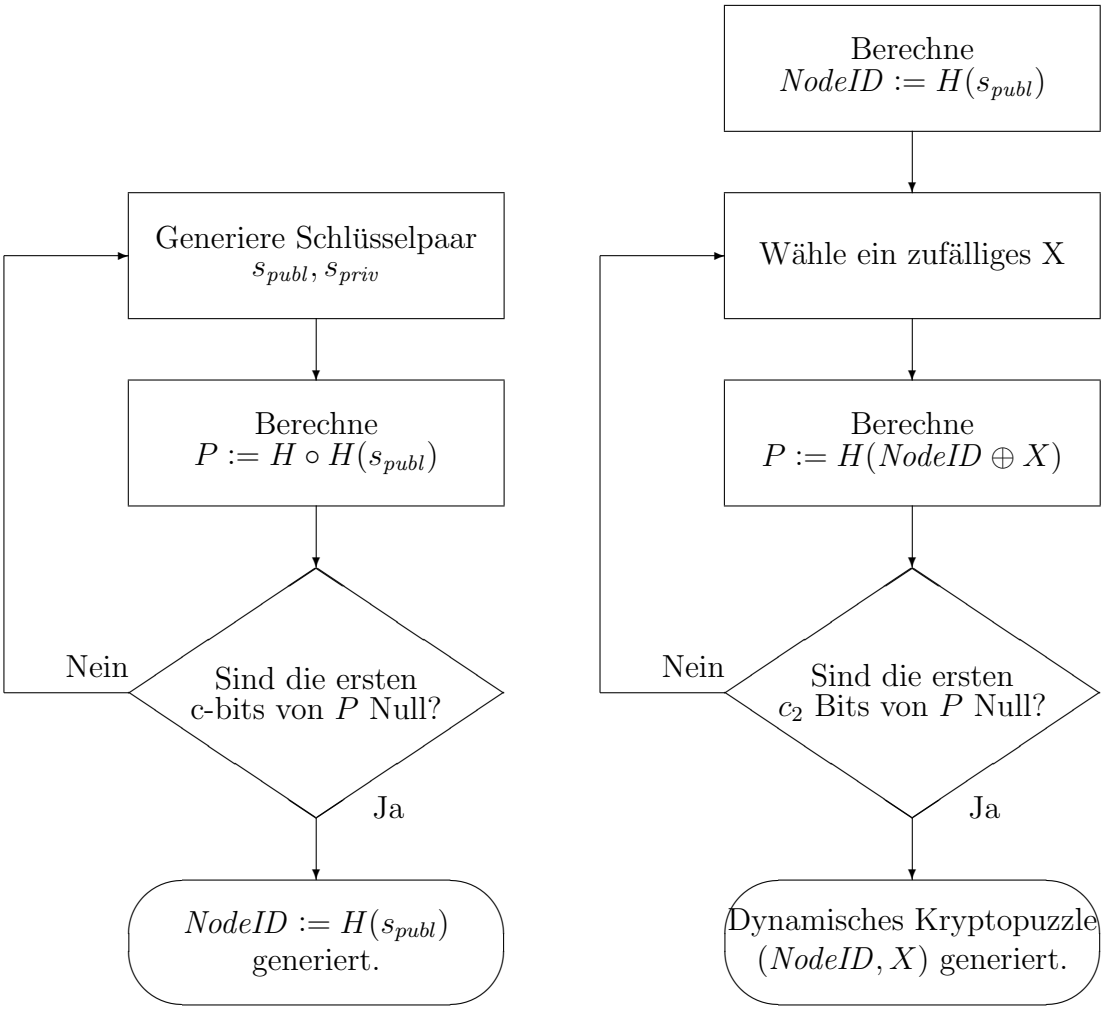


Abbildung 4.2: Statisches (links) und dynamisches (rechts) Kryptopuzzle

4.2.1 Aufbau der Routingtabelle

Die Routingtabelle besteht aus k-Buckets. Der $(i \cdot (2^b - 1) + j)$-te k-Bucket speichert k Kontakte mit dem Abstand

$$d_{\oplus}(x, y) \in [j \cdot 2^{160-(i+1)\cdot b}, (j+1) \cdot 2^{160-(i+1)\cdot b})$$

mit $0 < i \leq \frac{n}{b}$ und $0 \leq j < 2^b$. Der Parameter b gibt an, wieviele Bits des Präfixes auf einmal betrachtet werden sollen, n ist die Zahl der Bits der *NodeID*, und k ist die Anzahl der Kontakte pro Bucket. Abbildung 4.3 zeigt eine beispielhafte Routingtabelle.

110101	101111 101101	leer	leer	001011 001010	leer	leer	leer	leer	001011 001010
11xxxx	**10xxxx**	**01xxxx**	**0011xx**	**0010xx**	**0001xx**	**000011**	**000010**	**000001**	**Nachbarn**

Abbildung 4.3: Routingtabelle eines Knotens mit *NodeID* $= 0$ und $b = 2$, $n = 6$, $k = 2$

Dies entspricht einer Kademlia Routingtabelle wie sie in Abschnitt 2.3.2 vorgestellt wird. Allerdings fehlt die Berücksichtigung des kleinsten Teilbaums. Dadurch ist nicht mehr sichergestellt, dass jeder Knoten seine k' Nachbarn kennt.

Deshalb wird zusätzlich zu den k-Buckets eine sortierte Nachbarschaftsliste geführt. Dies hat den Vorteil, dass ein Knoten mit hoher Wahrscheinlichkeit feststellen kann, ob er bereits die komplette Nachbarschaft eines gesuchten Schlüssels kennt. Denn damit gelten die Vorüberlegungen aus Abschnitt 3.3.3 der Analyse auch für den Lookup einer Nachbarschaft.

Es stellt sich die Frage, wie groß eine solche Tabelle sein muss, damit ein Knoten X die k' Nachbarn eines Schlüssels S kennt, wenn sich S unter Xs k' Nachbarn befindet. Dies ist in dem Fall, dass *NodeID*s linear, also mit konstantem Abstand zwischen den Knoten generiert werden, kein Problem. Es muss lediglich sichergestellt sein, dass sich die Nachbarschaft ausreichend überlappt.

Für zufällig gewählte *NodeID*s muss gezeigt werden, dass sich mit hoher Wahrscheinlichkeit unter einer gewissen Anzahl δ von Nachbarn von X die k' Nachbarn von S wiederfinden. Dieser Beweis wurde bereits in der Arbeit von Gai und Viennot zu der Broose-DHT [GV04] geführt. Demnach ist es mit hoher Wahrscheinlichkeit ausreichend, $\delta = 2 \cdot c \cdot k'$ mit $c = 3.5$ Nachbarn zu kennen. Da dieser Beweis recht kurz gefasst und durch die Verwendung der Chernoff-Schranken [HR90] ungenau ist, wird er an dieser Stelle leicht modifiziert nachvollzogen:

Die XOR-Metrik hat die Eigenschaft, dass für x fest und für alle $d_{\oplus}(x, y)$ genau ein y existiert. Werden nun *NodeID*s zufällig gewählt, so ist die Wahrscheinlichkeit, dass eine *NodeID* y kleiner als eine *NodeID* x ist durch $\frac{x}{2^n}$ gegeben (n ist die Zahl der Bits der *NodeID*). Sei nun N die Anzahl der Knoten im Netz, dann ist die *durchschnittliche* Distanz über die XOR-Metrik zwischen zwei benachbarten Knoten $\frac{2^n}{N}$. Wird der Abstand

$$d_N(\mu) = \mu \cdot \frac{2^n}{N}$$

von einem Knoten x zu einem Knoten y betrachtet, so ist die zu erwartende Anzahl $N(x, y)$ der Knoten zwischen x und y gerade $E[N(x, y)] = \mu$ und die Wahrscheinlichkeit, dass ein Knoten zwischen x und y liegt, beträgt $\frac{\mu}{N}$.

Da die *NodeID*s zufällig gewählt werden, handelt es sich bei diesem Sachverhalt um ein Bernulli-Experiment. Daraus folgt, dass die tatsächliche Anzahl der Knoten zwischen x und y von dem Erwartungswert μ abweichen. Die Wahrscheinlichkeit, dass sich zwischen x und y mit dem Abstand $d_N(ck)$ weniger als k' Knoten befinden, ist damit gegeben durch:

$$\Pr[N(x, y) < k'] = \sum_{i=0}^{k-1} \binom{N}{i} \left(\frac{ck'}{N}\right)^i \left(1 - \frac{ck'}{N}\right)^{(N-i)}$$

Diese Wahrscheinlichkeit ist für kleine $k > 0$ berechenbar, denn es gilt:

$$\binom{n}{k} = \prod_{i=1}^{k} \frac{n + 1 - i}{i}$$

Das heißt, die Verwendung einer Chernoff-Schranke, welche in [GV04] benutzt wird, ist nicht nötig. Damit ist das Ergebnis dieser Wahrscheinlichkeitsrechnung genauer und es lassen sich folgende Wahrscheinlichkeiten in Abhängigkeit von c und k' mit $N = 10^{10}$ berechnen:

	$c = 1.5$	$c = 2.0$	$c = 2.5$	$c = 3.0$
$k' = 8$	$0.8950 \cdot 10^{-1}$	$1.0000 \cdot 10^{-2}$	$0.7786 \cdot 10^{-3}$	$0.4750 \cdot 10^{-4}$
$k' = 16$	$0.3440 \cdot 10^{-1}$	$0.6600 \cdot 10^{-3}$	$0.5464 \cdot 10^{-5}$	$0.2590 \cdot 10^{-7}$
$k' = 20$	$0.2187 \cdot 10^{-1}$	$0.1763 \cdot 10^{-3}$	$0.4791 \cdot 10^{-6}$	$0.6352 \cdot 10^{-9}$
$k' = 32$	$0.5925 \cdot 10^{-2}$	$0.3617 \cdot 10^{-5}$	$0.3506 \cdot 10^{-9}$	$0.1022 \cdot 10^{-13}$

Der Rest des Beweises kann analog zu [GV04] fortgesetzt werden. Insbesondere wird gezeigt, dass die Größe $\delta = 2ck'$ der Nachbarschaftstabelle ausreicht, um die oben genannte Bedingung mit hoher Wahrscheinlichkeit zu erfüllen.

Ein Problem tritt allerdings auf, wenn die Nachbarschaften aller Knoten nicht den gesamten *ID* Raum abdecken. Diese Lücken enstehen, wenn alle rechtsseitigen (kleineren) bzw. linksseitigen (größeren) *ID*s näher an einem Knoten liegen. Da die *NodeID*s zufällig erzeugt werden, ist die Wahrscheinlichkeit, dass sich ein Knoten linksseitig bzw. rechtsseitig befindet und näher ist, im Durchschnitt $p \approx 0.5$. Damit ergeben sich die die Wahrscheinlichkeiten in Abhängigkeit der Anzahl der Nachbarn:

k'	2	4	8	16	32
P	0.25	0.0625	$3.9 \cdot 10^{-3}$	$1.5 \cdot 10^{-5}$	$2.3 \cdot 10^{-10}$

Diesem Problem kann entgegengewirkt werden, indem entweder k' ausreichend groß gewählt wird, oder aber zwei getrennte Nachbarschaftslisten, für linksseitige und rechtsseitige Nachbarn, geführt werden.

In Abschnitt 3.4.3 wird die sinnvolle Anzahl von 16 Replikaten festgelegt. Da die Replikate in der Nachbarschaft abgelegt werden sollen, muss $k' = 16$ gewählt werden und es ergibt sich mit $c = 2.5$, $\delta = 80$. Die Wahrscheinlichkeit, dass Lücken im *ID* Raum entstehen ist mit $1.5 \cdot 10^{-5}$ ausreichend klein.

Es wurde zuvor beschrieben, dass ein k-Bucket aus k Kontakten besteht. Was ein Kontakt beeinhaltet wurde noch nicht definiert. Dies soll jetzt nachgeholt werden. In dieser Arbeit werden 3 Arten von Kontakten unterschieden:

Underlay-Kontakt Der Underlay-Kontakt enthält alle Informationen, damit die unter dem KBR liegende Netzwerkschicht eine Nachricht an einen anderen Knoten schicken kann. Im Falle von TCP/IP sind dies IP-Adresse und Port.

Overlay-Kontakt Der Overlay-Kontakt enthält alle die Informationen aus dem Underlay-Kontakt und enthält zusätzlich noch die Overlay-*NodeID* des Knotens.

Kademlia-Kontakt Der Kademlia-Kontakt enthält alle Informationen aus dem Overlay-Kontakt und enthält zusätzlich folgende Attribute:

- `credits` gibt die Vertrauenswürdigkeit mit `credits` $= 1 \ldots c_{max}$ an. Ein Kontakt ist nicht vertrauenswürdig, wenn `credits` $= 1$ ist.
- `stale` gibt an, wie oft dieser Kontakt auf eine Nachricht nicht geantwortet hat. Erhält ein Knoten eine Nachricht von einem bekannten Kontakt, so wird dieser Wert auf Null gesetzt.

- `lastseen` ist ein Zeitstempel, der angibt, zu welchem Zeitpunkt die letzte Nachricht von diesem Kontakt empfangen wurde.

Dieser Kontakt wird in den k-Buckets sowie in der Nachbarschaftsliste abgelegt.

Wie im ursprünglichen Kademlia besitzt auch diese Routingtabelle einen `FIND_NODE` RPC, welcher die k naheliegendsten Knoten zu einem Schlüssel S zurückgibt. Da, wie zuvor erläutert, genau bestimmt werden kann, ob bereits die k Nachbarn zu dem Schlüssel S verfügbar sind, wird dies in der RPC-Antwort durch ein Flag angezeigt.

4.2.2 Vertrauensbeziehungen der Routingtabelle

Bei VoIP besteht ein soziales Netz zwischen den Teilnehmern. Jeder Teilnehmer lernt nach kurzer Zeit andere Teilnehmer kennen, welchen er vertraut. Diese Teilnehmer werden in einer Kontaktliste abgelegt und bilden *transitiv* fortgesetzt ein *soziales Netz* eines Teilnehmers.

Die Einträge dieser Liste können entweder manuell, über Benutzerinteraktion, oder automatisch, etwa wenn ein längeres Gespräch mit einem Teilnehmer geführt worden ist, vorgenommen werden. Dies ist die Aufgabe des Peer-to-Peer SIP-Proxies bzw. des SIP-Clients und soll hier nicht genauer erläutert werden.

Im Folgenden wird ein vertrauenswürdiger Kontakt „Bekanntschaft“ genannt. Kennt ein Teilnehmer einen anderen Teilnehmer über n Bekanntschaften, so heißen sie *Bekanntschaften n-ten Grades*. Abbildung 4.4 zeigt das soziale Netz eines Knotens Q.

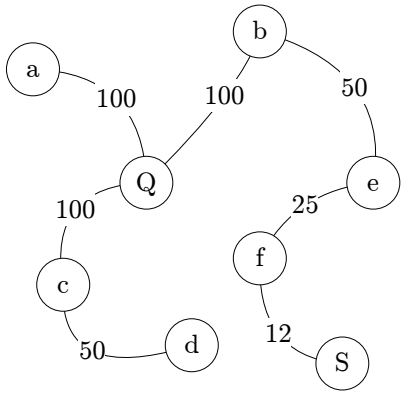


Abbildung 4.4: Das soziale Netz des Knotens Q

Es liegt nahe dieses soziale Netz zu nutzen, um das Key-based Routing weiter gegen einen Sybilangriff zu schützen. Im Kademlia-Kontakt ist zu diesem Zweck bereits das Attribut `credits` vorgesehen. Dieses Attribut wird genutzt, um die Vertrauenswürdigkeit eines Kontakts anzugeben. Sobald ein vertrauenswürdiger Kontakt in die Routingtabelle aufgenommen wird, kann dieses Attribut entspechend gesetzt werden. Somit kann das Routing vertrauenswürdige Kontakte gegenüber den naheliegendsten Kontakten bei einem Lookup stärker berücksichtigen.

Bekanntschaften n-ten Grades können zusätzlich dazu verwendet werden, die Routingtabelle mit mehr vertrauenswürdigen Kontakten zu füllen. Der Vorteil ist, dass

dadurch das Key-based Routing über überwiegend vertrauenswürdige Kontakte führt und ein Sybilangriff weiter erschwert wird. Er kann sogar verhindert werden, wenn in jedem nicht-leeren Bucket mindestens ein vertrauenswürdiger, mit `credits` > 1, Kontakt vorhanden ist.

Damit ein „vertrauenswürdiges Routing“ möglich ist, muss der `FIND_NODE` RPC, welcher die k naheliegendsten Knoten zu einem gegebenen Schlüssel S aus der Routingtabelle zurückgibt, angepasst werden. So werden diese k-naheliegendsten Knoten zusätzlich durch die Funktion

$$d_T : (s, y) \rightarrow \frac{d_{\oplus}(s, y) \cdot c_{max}}{\texttt{credits}(y)}$$

aufsteigend sortiert. Dabei ist s die gesuchte *NodeID* und x die *NodeID* einer der naheliegendsten Knoten.

Sollte sich s bereits unter den naheliegendsten Knoten befinden, so ist durch d_T sichergestellt, dass dieser immer am Anfang der Liste steht. Da die Funktion die naheliegendsten Knoten lediglich anders sortiert, wird das Routingverhalten mit hoher Wahrscheinlichkeit nur wenig beeinflusst. Abbildung 4.5 zeigt das Overlaynetz mit einem vertrauenswürdigen Pfad und dem gewöhnlichen Overlay-Pfad von Q nach S.

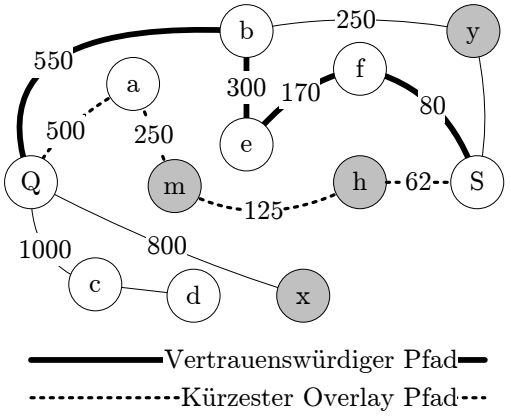


Abbildung 4.5: Overlay-Pfade unter Berücksichtigung des sozialen Netzes

Zusätzlich zu `FIND_NODE` wird ein `FIND_TRUSTED_NODE` benötigt um vertrauenswürdige Kontakte n-ten Grades zu erhalten. Dabei werden nicht die naheliegendsten Knoten, sondern eine durch den RPC gegebene Anzahl von Kontakten, deren Wert an `credits` höher ist als eine gegebene Schwelle, zurückgegeben. Es ist empfehlenswert, den vertrauenswürdigen Kontakten n-ten Grades, mit n>1, quadratisch abfallend weniger `credits` zu geben als den Kontakten ersten Grades. Denkbar wäre eine Halbierung in jedem Schritt. Weiterhin sei angemerkt, dass durch das „Six Degrees of Separation“ oder „Small World“ [Kle00, Kle01] -Phänomen empirisch jede Person über sechs andere Personen mit jeder Person bekannt ist. Aus diesem Grund ist es naheliegend, dass ein vertrauenswürdiges Routing, ähnlich dem Web-of-Trust, mit hoher Wahrscheinlichkeit funktioniert.

4.2.3 Wartung der Routingtabelle und Netzbeitritt

Bezüglich der Routingtabelle beibt als letzter Punkt noch die Wartung der Routingtabelle. Dies geschieht wie bei Kademlia durch eingehende Nachrichten, welche sich wie folgt kategorisieren lassen:

1. Signierte Anfragen durch RPCs anderer Knoten.
2. Signierte Antworten auf RPCs des eigenen Knotens.
3. Unsignierte Nachrichten.

Jede dieser Nachrichten enthält den Overlay-Kontakt des Absenders, welcher sich bei TCP/IP aus IP-Adresse, Port und der *NodeID* zusammensetzt. Bei signierten Nachrichten ist außerdem durch die Sicherheitsschicht klar, dass IP-Adresse und Port wirklich zu der *NodeID* gehören.

Ein Kontakt wird bei signierten Antworten auf RPCs des eigenen Knotens sofort in die Routingtabelle aufgenommen, wenn der entsprechende k-Bucket nicht voll sein sollte oder er sich in der Nachbarschaft des eigenen Knotens befindet. Enthält eine Antwort weitere Kontakte, welche in die Routingtabelle aufgenommen werden könnten, so wird auf diesen Knoten die `PING`-RPC aufgerufen. Sie werden dann mit der Antwort auf den RPC in die Routingtabelle aufgenommen. Dies stellt sicher, dass jeder Kontakt in der Routingtabelle authentifiziert wird.

Ein Kontakt aus einer signierten Anfrage wird nur bei einer gewissen Nähe zu dem lokalen Knoten in die Routingtabelle aufgenommen, beispielsweise wenn die Präfixe über 32 Bit beider *NodeID*s übereinstimmen. Dies verhindert, wie in Abschnitt 3.3.2 beschrieben, dass ein bösartiger Knoten einen leeren k-Bucket mit Kontakten fluten kann.

Kontakte aus unsignierten Nachrichten werden in keinem Fall in die Routingtabelle aufgenommen.

Ist ein Kontakt bereits in einem k-Bucket der Routingtabelle vorhanden, so wird der Kontakt aktualisiert. Das bedeutet, dass `stale` auf Null gesetzt wird, `lastseen`, IP-Adresse und Port aktualisiert werden und der Kontakt an das Ende des k-Buckets verschoben wird. Alte Kontakte stehen damit immer am Anfang des k-Buckets.

Sollte sich der Kontakt in der Nachbarschaftsliste des Knotens befinden, so wird gleichermaßen vorgegangen, allerdings behält der Kontakt seinen sortierten Platz in der Nachbarschaftsliste.

Die Routingtabelle wird analog zu Kademlia aktualisiert. Dies geschieht regelmäßig: Nach einem Aktualisierungsintervall $t_a \approx 1$ Stunde wird periodisch ein Nachbarschaftslookup auf die eigene *NodeID* durchgeführt, welches die k Nachbarn der Nachbarschaftsliste aktualisiert.

Um mindestens δ Nachbarn zu kennen, werden in einem zweiten Schritt k' der naheliegendsten Nachbarn nach ihrer Nachbarschaft gefragt. Ist unter den zurückgegebenen Kontakten ein Knoten nicht in der eigenen Nachbarschaftsliste, so wird auf diesem Knoten der `PING`-RPC aufgerufen. Nach Erhalt einer Antwort auf den RPC wird dieser Kontakt dann in die Nachbarschaftsliste aufgenommen.

Nach jedem Intervall $t_p = 30$ Minuten wird allen Nachbarn periodisch eine `PING` Nachricht gesendet, welche im verstrichenen Interval nicht kontaktiert wurden. Dies ist einerseits notwendig, um den eigenen Knoten bei seiner Nachbarschaft bekannt zu halten. Andererseits werden auf diese Weise ausgefallene Knoten aus der Nachbarschaftsliste entfernt.

Da sich in dem hier vorgestellten Design die k-Buckets nur sehr langsam durch passive Anfragen anderer Knoten füllen, wird zusätzlich nach jedem Aktualisierungsintervall ein Lookup ausgehend von den Nachbarschaftsknoten auf das Komplement der eigenen *NodeID* durchgeführt. Dies führt dazu, dass ein Pfad quer über die k-Buckets beschritten werden muss, und füllt damit bei jedem Hop einen k-Bucket mit mindestens einem Kontakt.

Der Netzbeitritt wird, wie bei Kademlia, lose ausgeführt. Das heißt, es werden die Bootstrap-Knoten beginnend mit der ersten Aktualisierung in die Routingtabelle eingetragen. Nach kurzer Zeit füllt sich dann zunächst die Nachbarschaftsliste und zunehmend, auch die k-Buckets. Mit jedem Lookup eines Knotens steigt die Präsenz im Netz.

Zusammenfassend werden nach jedem Aktualisierungsinterval t_a folgende Operationen ausgeführt:

1. Nachbarschaftslookup der eigenen *NodeID* um die k naheliegendsten Knoten zu aktualisieren.
2. Aktualisierung der Nachbarschaftsliste, indem die naheliegendsten k Knoten nach ihren k Nachbarn gefragt werden und damit fehlende Knoten in die eigene Nachbarschaftsliste aufgenommmen werden.
3. Lookup des Komplements der eignen *NodeID* ausgehend von den k-Nachbarschaftsknoten um die k-Buckets zu füllen.

Punkt 1 und 2 lassen sich als `FIND_NODE` Overlay-RPC-Aufruf bei der eigenen Nachbarschaft zusammenfassen. Das vereinfacht später die Implementierung der Wartungsalgorithmen.

In diesem Abschnitt wird ein Nachbarschaftslookup verwendet um die k Nachbarn zu finden. Im nächsten Abschnitt soll dieser Lookup genauer beschrieben werden.

4.2.4 Lookup eines Knoten oder einer Nachbarschaft

Der Lookup eines Knotens oder einer Nachbarschaft wird, wie bei Kademlia, iterativ durchgeführt. Sei Q der eigene Knoten und S ein Hashwert. Dann sucht Q zunächst die k naheliegendsten Kontakte zu S aus seiner eigenen Routingtabelle[2]. Aus dieser Kontaktmenge werden $d(< k)$ Kontakte genutzt um parallel knotendisjunkte Pfade zu dem Zielknoten S zu finden. Dazu wird für jeden Kontakt i parallel folgender *Pfadlookupalgorithmus* ausgeführt:

Sei A die Menge der bereits angefragten Kontakte über alle parallelen Pfadlookups und $X := \{i\}$ die Menge der zu befragenden Kontakte dieses *Pfadlookups*. Dann führe folgenden Algorithmus aus:

[2]Dies entspricht einem `FIND_NODE` auf dem eigenen Knoten

1. Fordere von α Kontakten $Y \subset X \setminus A$ die k naheliegendsten Kontakte zu S an und setze $A := Y \cup A$. Antwortet keiner der α Knoten, wiederhole den Vorgang mit einem weiteren Kontakt aus $X \setminus A$. Ist kein Kontakt mehr in der Menge $X \setminus A$, warte eine Zeit t_{fail}. Antwortet in dieser Zeit kein kontaktierter Knoten, so ist der Pfadlookup fehlgeschlagen.

2. Die zeitlich erste Menge F der k naheliegendsten Kontakte zu S, welche von einem Knoten zurückgegeben wird, ist die neue Menge der zu befragenden Knoten $X := F$ und Schritt 1 wird wiederholt. Eventuell später eintreffende Antworten werden ignoriert. Behauptet der Knoten, dass bereits alle k Nachbarn in F enthalten sind, und handelt es sich bei diesem Lookup um einen Nachbarschaftslookup, so wird auf allen Kontakten aus F eine $PING$-RPC ausgeführt und der Pfadlookup nach Ablauf von $t_{success}$ beendet.

Während dieser Algorithmus parallel ausgeführt wird, werden die Antworten der RPCs genutzt, um eine sortierte Liste zu füllen. Sie beinhaltet k Kontakte, welche in aufsteigender Distanz zum Hashwert S sortiert sind. Bei einem Knotenlookup wird der Lookup abgebrochen, wenn der Zielknoten in diese Liste aufgenommen wird. Bei einem Nachbarschaftslookup müssen alle Pfadlookups terminieren. Die Liste enthält dann k Nachbarn von S.

Der Wert α gibt an, wie viele Knoten parallel in jedem Schritt befragt werden sollen. Dies beugt einer netz-immanenten Knotenfluktuation vor. Die Zeit $t_{success} = t_{fail}$ wird so gewählt, dass sie der doppelten durchschnittlichen *Round-Trip-Time* (RTT) entspricht. Die *Round-Trip-Time* kann wie bei TCP über alle RPCs berechnet werden.

Abbildung 4.6 zeigt einen *Pfadlookup* eines Knotens und macht die Funktionsweise des Algorithmus an einem Beispiel deutlich.

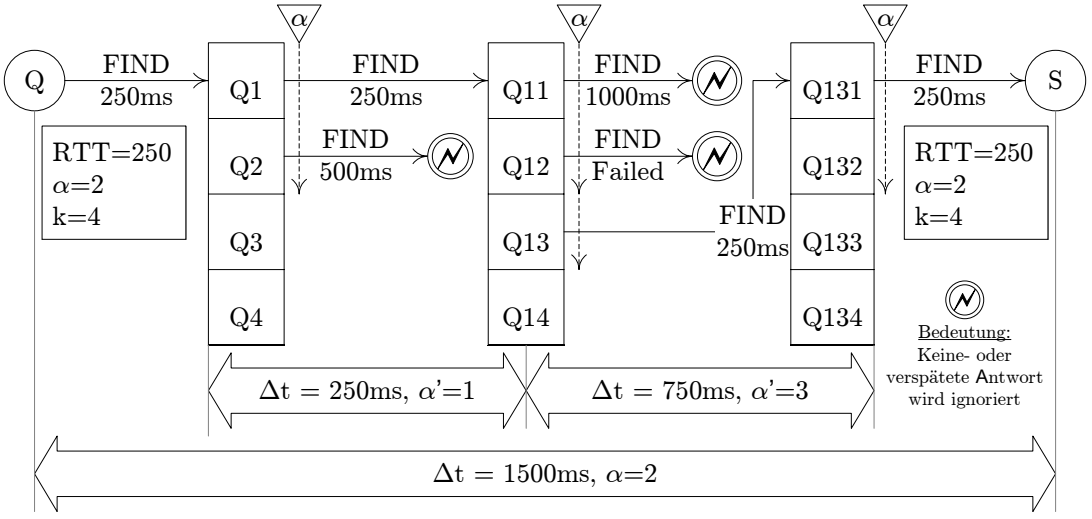


Abbildung 4.6: Pfadlookup eines Knotens

4.2.5 Abschätzen der Netzgröße

Um feststellen zu können, ob sich das Netz in seiner Aufbauphase befindet und damit, wie in 3.2.3 erwähnt, überwachte Signaturen benötigt werden, muss die ungefähre Netzgröße bekannt sein.

Es existieren für dieses Problem schon einige Lösungsansätze. So wird in [JMB04, JM04] ein Verfahren zum Zählen über Datenaggregation vorgestellt. Dieses Verfahren setzt voraus, dass der Zählvorgang von genau einem Knoten gestartet wird. Dies wirft Sicherheitsprobleme auf, denn eine Menge bösartiger Knoten könnte den Vorgang so oft starten, bis die Konvergenz der Netzgröße nicht mehr möglich ist. Es soll hier ein Verfahren vorgestellt werden, das im Prinzip eine Kombination aus Datenaggregation und Abschätzen darstellt.

Über die Nachbarschaft eines Knotens lässt sich die Netzgröße abschätzen: Sei n_0 die *NodeID* eines Knotens und $n_1, \ldots, n_\delta$ seien *NodeID*s der Nachbarn von n_0. Dann ist die durchschnittliche Distanz der Nachbarschaft gegeben durch:

$$L := \frac{1}{\delta} \cdot \sum_{i=1}^{\delta-1} (d_\oplus(n_0, n_{i+1}) - d_\oplus(n_0, n_i))$$

Damit ist die Netzgröße durch den Wertebereich der *NodeID* abschätzbar, denn es gilt:

$$N_l := \frac{2^n}{L}$$

dabei ist n die Anzahl der Bits der *NodeID*.

Diese Abschätzung ist aber nur sehr grob, denn die Wahrscheinlichkeit, dass ein neuer Knoten in einer Nachbarschaft zu einem vorhandenen Knoten liegt, ist $\frac{\delta}{N-1}$ (N ist die Netzgröße). Das heißt diese Abschätzung ändert sich erwartungsgemäß erst nach $\frac{N-1}{\delta}$ neuen Knoten. Dies hat zur Folge, dass die Abschätzung für große N immer ungenauer wird.

Um die Abschätzung zu verfeinern, wird zusätzlich eine Art Average-Algorithmus [CS04] benutzt. Dazu wird mit jeder Nachricht die eigene abgeschätze reelwertige Netzgröße $N_1 := N_l + N_\Delta$ übertragen. Erhält ein Knoten eine Nachricht mit der Netzgröße N_2, so errechnet dieser die neue Abweichung $N'_\Delta := \frac{1}{2} \cdot (N_2 + N_\Delta - N_1)$. Da Kademlia RPCs benutzt, wird diese Operation quasi parallel auf zwei Knoten ausgeführt. Dadurch besitzen beide Knoten die gleiche Abschätzung. Aus Sicherheitsgründen wird lediglich eine maximale Abweichung von $-\frac{N_1}{2} < N_\Delta < \frac{N_1}{2}$ akzeptiert. Daher lässt sich die Abschätzung durch bösartige Knoten nur sehr schwach beeinflussen.

Dass der Algorithmus nach wenigen Nachrichten pro Knoten konvergiert, wird im Abschnitt 6.2 der Evaluierung gezeigt.

4.3 Verteilte Hashtabelle

Die verteilte Hashtabelle ist für die Speicherung, Replizierung und Überprüfung von Daten zuständig. Im Gegensatz zu Kademlia muss sichergestellt sein, dass bösartige Knoten möglichst wenig Einfluss auf die Daten haben können. Wie in Abschnitt 3.4.1 analysiert, müssen die Angriffe der Replikation und des Insertion Denial-of-Service verhindert werden.

Jedes Schlüssel-Wert Paar enthält zusätzlich ein Attribut *Time-to-live* (`TTL`), welches angibt bis zu welchem Zeitpunkt dieses Paar gespeichert wird, sowie die *NodeID* des Autors.

In den folgenden Abschnitten wird zunächst die Ablage und Abfrage von Daten beschrieben. Darauf folgt die Wartung der verteilten Hashtabelle welche Mehrheitsentscheidungen fällt und die eigene Hashtabelle nach dem Netzbeitritt mit Replikaten füllt. Zusätzlich wird im Abschnitt „Optimierungen“ ein Verfahren vorgestellt, mit welchem sich Daten noch besser vor Angriffen schützen lassen.

4.3.1 Ablage von Daten

Für die Ablage der Daten kommt, wie bei Kademlia, der `STORE` RPC zum Einsatz. Er fordert einen Knoten auf, ein Schlüssel-Wert Paar (K, V) zu speichern. Wie in Abschnitt 3.4 beschrieben, werden lediglich die Hashwerte eines Schlüssels gespeichert. Weiterhin wird ein Schlüssel-Wert Paar nur auf einem Knoten gespeichert, wenn unter dem Schlüssel noch kein anderer Wert gespeichert wurde. Jedoch ist des dem Autor eines Schlüssel-Wert Paares gestattet, dieses zu ändern oder zu löschen.

Für jeden *Remote-Procedure-Call* der verteilten Hashtabelle wird eine starke Signatur der Sicherheitsschicht erwartet. Sie stellt die Integrität der Daten sicher und authentifiziert den Autor der Nachricht.

Für die Ablage von Daten muss ein Kryptopuzzle zu einem gegebenen Schlüssel gelöst werden, um den Insertion Denial-of-Service zu erschweren und den Replikationsangriff zu verhindern. Zusätzlich muss die Speicherung eines Schlüssel-Wert Paares von den Nachbarn des Autors autorisiert werden, um den Insertion DoS nicht nur zu erschweren, sondern zu vermeiden.

Diese beiden Schritte sind notwendig, denn die Authorisierung alleine würde eine Schwachstelle bieten: Es wird zwar das Einfügen beschränkt, aber der Aufwand der Prüfung, welcher nicht unerheblich ist, kann durch eine Vielzahl von Speicheraufforderungen so hoch werden, dass der Knoten daraufhin zusammmenbricht. Deshalb muss dieser Aufwand durch das Kryptopuzzle „entschädigt“ werden.

Das Kryptopuzzle funktioniert wie in [Bac]: Sei k der Hashwert des Schlüssels. Dann wähle ein b so, dass der Präfix über c-Bits von $H(k \oplus b)$ mit dem Präfix der eigenen *NodeID* übereinstimmt (also eine partielle Kollision). b ist dann der Beweis, dass dieses Kryptopuzzle gelöst wurde. Der Wert c wird so gewählt, dass er den Aufwand zur Berechnung von b des `STORE` RPCs entschädigt. Dieses Kryptopuzzle erschwert auch den Replikationsangriff, denn der Angreifer müsste in *sehr* kurzer Zeit eine eigene Lösung des Puzzles berechnen. Dies macht einen Session-Key überflüssig.

Für die Autorisierung des Schlüssels wird ein `GRANT` RPC auf allen Nachbarn ausgeführt. Diese speichern unter der *NodeID* des Autors die Hashwerte der Schlüssel, die bisher gespeichert wurden. Wurden bereits zu viele Kennungen registriert, so wird von der `GRANT` RPC ein Fehler zurückgegeben. Andernfalls wird der neue Schlüssel in die Liste eingetragen und ist damit autorisiert. Daraus ergibt sich ein Schlüssel-Wert-Paar welches als Schlüssel die *NodeID* des Autors und als Wert eine Liste der registrierten Schlüssel des Autors enthält. Diese wird wie jedes andere repliziert und kann von jedem Knoten abgefragt werden.

Wurde der `STORE` RPC aufgerufen, so prüft jeder Knoten, ob die Operation autorisiert wurde. Dies geschieht durch den `CHECK` RPC, welcher jeweils an alle Nachbarn des Autors versendet wird.

Das Verfahren lässt sich wie folgt zusammenfassen:

1. Lösen des Kryptopuzzles für einen gegebenen Schlüssel und *NodeID*
2. Sende `GRANT` RPCs an die eigenen Nachbarn
3. Sende `STORE` RPCs an die Nachbarn des designierten Knotens.
4. *Alle* Nachbarn überprüfen, ob die Speicherung autorisiert wurde, indem sie die `CHECK` RPC auf den Nachbarn des Autors ausführen.
5. Wurden alle Prüfungen bestanden, wird das Schlüssel-Wert Paar gespeichert.

Abbildung 4.7 stellt dieses Verfahren anschaulich dar.

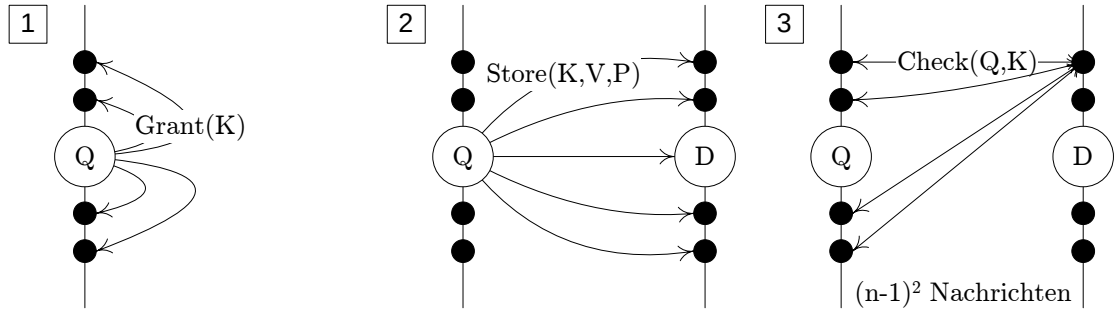


Abbildung 4.7: Das Grant-Check-Store (GCS) Verfahren. Q ist der Autor, und D der designierte Knoten der Daten

4.3.2 Abfrage von Daten

Das Abfrage von Daten wird über zwei RPCs realisiert: `FIND_HASHES` und `GET_VALUE`.

`FIND_HASHES` gibt eine gegebene Anzahl von Schlüssel-Wert Hashpaaren zurück, welche sich in der Nachbarschaft von einem gegebenen Hashwert befinden. Dies ist auch für die spätere Wartung nötig. `GET_VALUE` gibt den Wert zu einem gegebenen Schlüssel zurück, wenn dieser auf dem Knoten gespeichert wurde.

Um eine Mehrheitsentscheidung fällen zu können, werden von allen Nachbarschaftsknoten des gesuchten Schlüssels die Hashwerte des Wertes über `FIND_HASHES` angefordert. Ist eine Entscheidung möglich, so wird der Knoten der den richtigen Wert gespeichert hat, mit `GET_VALUE` nach dem Wert gefragt. Antwortet dieser Knoten nicht oder stimmt der Hashwert des zurückgegebenen Wertes nicht mit der Mehrheitsentscheidung überein, so wird ein weiterer Knoten gefragt, bis ein richtiger Wert zurückgegeben wird.

4.3.3 Wartung der Hashtabelle

Nachdem ein Knoten dem Netz beigetreten ist, wird direkt im Anschluss mit der Wartung der verteilten Hashtabelle begonnen. Dazu werden alle Nachbarn nach dem Schlüssel gefragt, welcher der eigenen Knoten *NodeID* am naheliegendsten ist. Die Nachbarn antworten mit Hashwerten des Schlüssels und dem Hashwert des dazugehörigen Wertes. Für jeden Schlüssel wird dann über die Nachbarschaftstabelle überprüft, ob sich dieser auch wirklich in der Nachbarschaft des eigenen Knotens

befindet. Ist dies der Fall werden die Schlüssel in eine Tabelle eingetragen. Diese Tabelle enthält zu jedem Schlüssel die verschiedenen Hashes der Werte sowie deren Anzahl.

Ist eine gewisse Anzahl an gleichen Hashwerten erreicht und ist dies die Mehrheit, so wird der reale Wert von einem der Knoten angefordert und gespeichert. Sollte der Hashwert des abgefragten Werts nicht mit dem Hashwert der Mehrheitsentscheidung übereinstimmen, so werden weitere Knoten befragt, bis dieser stimmt. Desweiteren wird überprüft, ob sich ein Schlüssel nicht mehr im Zuständigskeitsbereich des Knotens befindet. Der Zuständigkeitsbereich kann etwas größer gewählt werden als lediglich die k Nachbarn. Das verhindert, dass die Knotenfluktuation ein ständiges Löschen und erneutes Speichern auslöst.

Dieser Vorgang kann nach einem längeren Zeitraum wiederholt werden (> 2 Stunden). Abbildung 4.8 zeigt die inkrementelle Mehrheitsentscheidung. Desweiteren werden eigene Schlüssel-Wert Paare, welche sich auf anderen Knoten befinden, aktualisiert. Das heißt, das *Time-to-Live* Attribut wird auf den entsprechenden Knoten durch eine `STORE` RPC aufgefrischt.

Schlüssel	Hashwerte der gespeicherten Werte				Status
	1	2	3	4	
1673	**8751 (04)**	1742 (01)	0123 (03)	-	Gespeichert: 8751
3249	2394 (01)	9058 (02)	2898 (02)	3498 (01)	Unentschieden
1235	**3453 (08)**	2345 (02)	6753 (01)	5645 (01)	Gespeichert: 3453
5465	3453 (04)	2784 (04)	6778 (04)	-	Unentschieden

Abbildung 4.8: Inkrementelle Mehrheitsentscheidung. Die Anzahl der Werte mit gleichem Hashwert steht in den Klammern

4.3.4 Optimierung

Um die Integrität der Daten weiter abzusichern, werden in diesem Abschnitt optionale Optimierungsmöglichkeiten vorgestellt:

Optimierung der Datenablage

Das GCS (Grant-Check-Store) Verfahren zur Datenablage nutzt das Kryptopuzzle um Replikationsangriffe zu erschweren. Sollte dieses Kryptopuzzle nicht ausreichen oder wird ein Schlüssel überwiegend an vertrauenswürdige Teilnehmer weitergegeben, so kann die Datenspeicherung noch weiter optimiert werden.

Dazu wird der Hashwert des Schlüssels als privater Schlüssel in einem ECDSA Signaturverfahren gewählt. Aus diesem privaten Schlüssel kann nun der öffentliche Schlüssel generiert werden. Dies funktioniert mit asymmetrischen Verschlüsselungsverfahren, welche auf elliptischen Kurven basieren besonders gut, da hier zur Schlüsselgenerierung lediglich eine Multiplikation innerhalb der Kurvenparameter benötigt wird.

Dieses Schlüsselpaar wird nun verwendet, um den Wert zu signieren. Öffentlicher Schlüssel, Signatur und Wert bilden damit das neue Schlüsselpaar, welches unter dem Hashwert des öffentlichen Schlüssels abgelegt wird. Ein Angreifer kann dieses

Paar jetzt nur noch fälschen, wenn er den ursprünglichen Schlüssel kennt. Dies hat einerseits den Vorteil, dass Mehrheitsentscheidungen schwerer aus dem Gleichgewicht zu bringen sind, wenn der ursprüngliche Schlüssel nicht bekannt wird. Andererseits, wird der Replikationsangriff in jedem Fall vermieden.

Ein Nachteil ist aber die fehlende *Crypto-Agility*, denn dieses Verfahren muss stets gleich bleiben. Es besteht keine Möglichkeit zur Änderung des Signaturverfahrens, ohne die alten Schlüssel zu verlieren. Angesichts der Tatsache, dass die Signatur, nachdem der urprüngliche Schlüssel öffentlich bekannt ist, gefälscht werden kann, ist dieser Nachteil nur von geringer Bedeutung.

Abbildung 4.9 zeigt die Generierung sowie die Verifizierung einer solchen kurzlebigen Datensignatur. Die nötige Veränderung der Suche nach einem so generiertenn Schlüsselpaar ist in Abbildung 4.10 dargestellt.

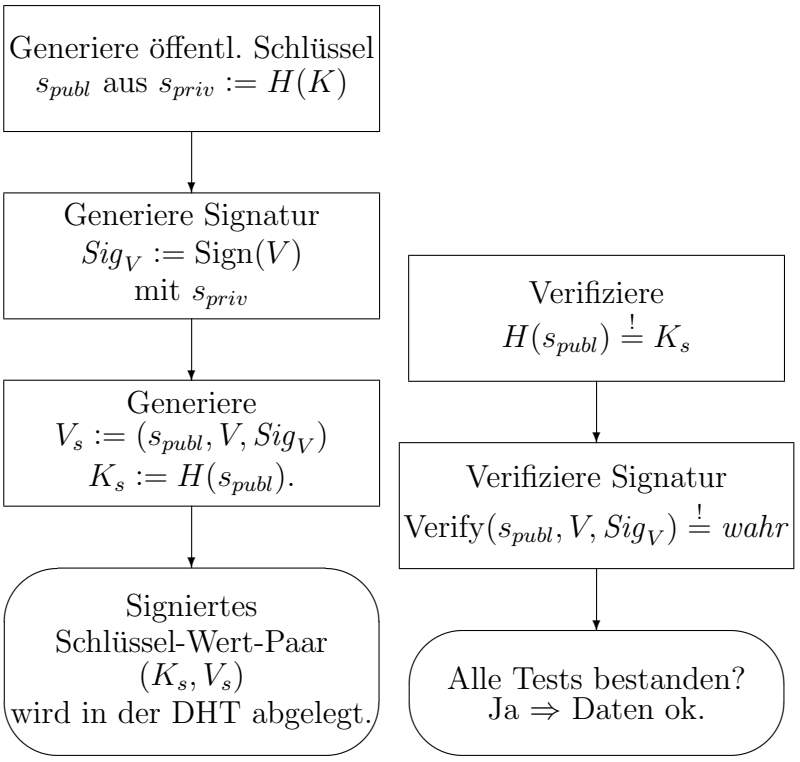


Abbildung 4.9: Generierung (links) und Verifizierung (rechts) einer kurzlebigen Datensignatur

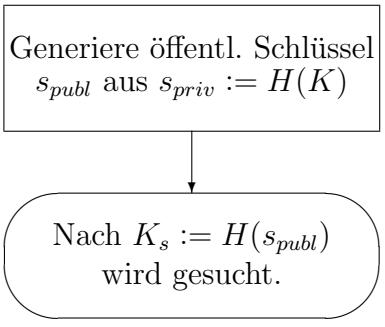


Abbildung 4.10: Suchen nach signierten Daten

Überprüfung der replizierten Daten

Jeder replizierende Knoten kann überprüfen, ob alle anderen replizierenden Knoten mehrheitlich die gleichen Daten speichern. Sollten Knoten über einen längeren Zeitraum der Mehrheitsentscheidung entsprechend falsche Daten speichern, können diese Knoten aus der Nachbarschaftsliste entfernt werden und in eine „schwarze Liste“ eingetragen werden. Um zu verhindern, dass bösartige Knoten erkennen können, dass der Knoten eine Überprüfung durchführt, kann die Anfrage über einen zufällig gewählten Knoten umgeleitet werden („*redirected probing*“).

Eine weitere Möglichkeit besteht darin, dass andere Knoten, welche nach einem Schlüssel-Wert Paar gesucht haben, die *stark* signierten Nachrichten der falschen Daten an alle Nachbarn des Schlüssels weiterleiten. Häufen sich diese Beweismittel, können die Knoten ebenfalls in die schwarze Liste eingetragen werden.

4.4 Namensdienst und P2P-SIP-Proxy

Der Namensdienst für einen Voice-over-IP Proxy kann aufgrund des vorherigen Entwurfs recht einfach realisiert werden. So wird unter einem Namen (Schlüssel) eine zertifikatähnliche Struktur (Wert) abgelegt. Diese enthält eine oder mehrere *NodeID*s des Teilnehmers. Weiterhin können auch Umleitungsinformationen in diesem Datensatz abgelegt werden.

Der P2P-SIP Proxy läuft mit der DHT lokal auf dem Rechner des Teilnehmers. Dies ist sinnvoll, denn so können SIP-Clients wie gewohnt weiterverwendet werden. Der SIP-Client registriert sich somit lokal und nicht mehr an einem administrierten SIP-Proxy. Es können sich auch mehrere SIP-Clients an einem P2P-SIP-Proxy registrieren, denn die DHT stellt lediglich eine Abbildung von einem Domainnamen auf eine *NodeID* her. Ein mögliches URI[3] Format könnte somit folgende Form haben:

```
p2psip://alice@host
```

Die DHT würde in diesem Fall den Namen „`host`“ suchen und die Anfrage an den in der DHT gespeicherten Knoten weiterleiten. Der lokale Proxy `host` kann dann prüfen, ob es eine „alice“ gibt und ggf. weiterreichen. Ist nur ein Domainname gegeben, so wird dieser natürlich mit leerem Namen weitergereicht. Abbildung 4.11 zeigt einen Sitzungsaufbau über ein P2P-SIP System.

Mit dem MSN Messenger ist es möglich seine Email-Adresse als Namen zu verwenden. Auch dies kann hier realisiert werden, indem das @-Zeichen einfach verdoppelt wird:

```
p2psip://alice@@tm.uka.de
```

Dies sind Vorschläge wie ein P2P-SIP-Proxy realisiert werden kann. An der entgültigen Standardisierung arbeitet mittlerweile eine P2P-SIP Arbeitsgruppe.

[3] Universal Ressource Identifier

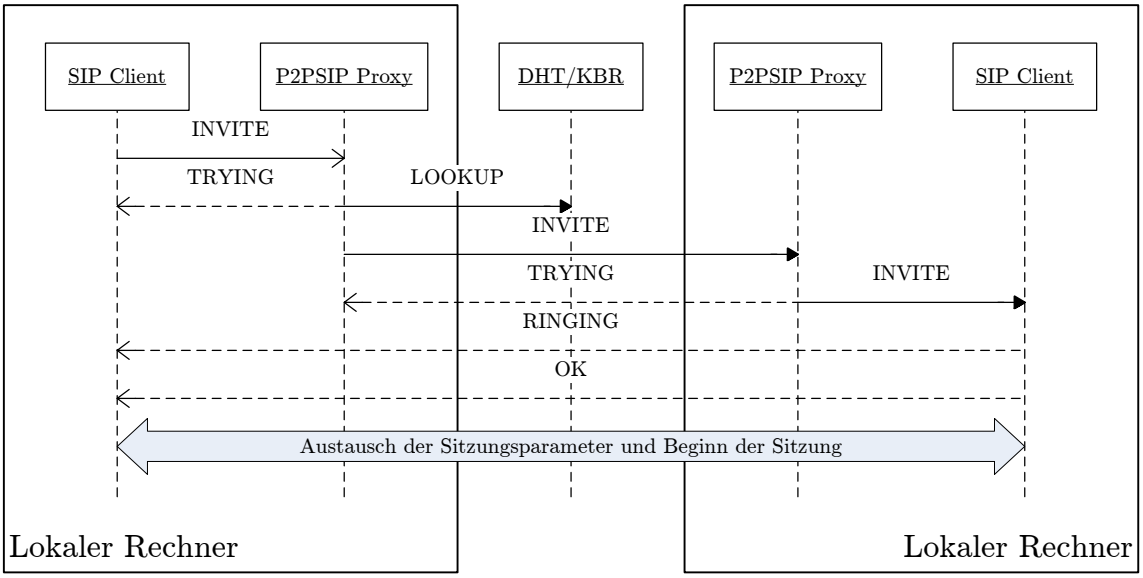


Abbildung 4.11: Aufbau einer Sitzung über P2P-SIP

5. Implementierung

Dieses Kapitel gibt zunächst einen Überblick über den am Institut für Telematik entwickelten Overlay-Simulator „OverSim“. Danach wird die Integration der Implementierung des entworfenen Peer-to-Peer Systems kurz vorgestellt.

5.1 Der Simulator OverSim

Zur Simulation der vorgestellten DHT wird OverSim verwendet. OverSim ist eine Erweiterung des OMNeT++ Simulators [Pon93]. OMNeT++ simuliert Umgebungen, welche auf diskreten Ereignissen basieren. Dazu gehören insbesondere Kommunikationsnetze. OMNeT++ stellt eine komponentenbasierte Architektur zur Verfügung. Simulationskomponenten werden in C++ geschrieben und können dann über sogenannte NED-Dateien kombiniert werden.

Zur Simulation von Kommunikationsnetzen werden bereits eine Vielzahl von Komponenten durch das INET Framework in OMNeT++ bereitgestellt. Diese erlauben die Simulation von IP, UDP/TCP, 802.11, Ethernet, PPP, IPv6, OSPF, RIP und MPLS sowie einiger weitere Protokolle. OverSim erweitert OMNeT++ um ein Framework, welches es ermöglicht Overlaynetze auf Basis des INET-Frameworks zu simulieren. Es bietet auch die Simulation eines einfachen Underlays (*SimpleNetwork*), welches UDP Nachrichten ohne Berücksichtigung der darunterliegenden Schichten überträgt. Der Vorteil ist, dass dadurch mehr Knoten innerhalb eines Zeitraums simuliert werden können. Im folgenden werden die einzelnen Komponenten vorgestellt:

Ein OverSim Knoten besteht aus zwei Komponenten: Einer Applikation- und Overlayimplementierung. In der Applikation werden Anfragen an das Overlay simuliert. Die Overlayimplementierung antwortet auf diese Anfragen. Die Formulierung der Anfragen folgt P. Druschel's Common-API [CDKR02]. Dieses Konzept versucht alle Funktionaliäten eines Overlays in einer API zu vereinen und besteht aus den 4 Basisdienstprimitiven: *route*, *put*, *get* und *remove*. Der Nachrichtentyp *route* wird verwendet, um Nachrichten im Overlay weiterzuleiten, *put*, *get* und *remove* dienen der Datenspeicherung in der verteilten Hashtabelle. Um die Entwicklung zu vereinfachen gibt es in OverSim für Applikation und Overlay Rahmenklassen:

BaseApplication: Die BaseApplication kapselt den Versand von Basisdienstprimitivnachrichten in gleichnamigen Methoden.

BaseOverlay: Das BaseOverlay nimmt Basisdienstprimitivnachrichten entgegen und ruft die gleichnamigen abstrakten Methoden auf. Diese Methoden können in abgeleiteten Klassen implementiert werden. Zusätzlich wird der anfallende Datenverkehr eines Knotens protokolliert.

Um ein Overlaynetz aufzubauen und Knotenfluktuation zu simulieren, werden zwei weitere Module benötigt: Das BootstrapOracle und der OverlayConfigurator:

BootstrapOracle: Das BootstrapOracle verwaltet alle Overlayknoten in einem Netz. Jedes Overlay registriert seine *NodeID* an diesem Modul und kann bereits registrierte Knoten abfragen.

OverlayConfigurator: Der OverlayConfigurator erzeugt und entfernt Overlayknoten im Overlaynetz. Hier können Knotenfluktuationsparameter sowie die initiale, minimale und maximale Anzahl der zu simulierenden Overlayknoten angegeben werden. Ausführlichere Informationen zu diesen Parametern sind in Tabelle 5.1 zu sehen.

Parameter	**Beschreibung**
simulateMobility	Gibt an, ob die Mobilität eines Knotens simuliert werden soll.
initialMobilityDelay	Die initiale Verzögerung einer Veränderung in Sekunden.
targetMobilityDelay	Die Zielverzögerung einer Veränderung in Sekunden.
initialOverlayTerminalNum	Anzahl der initialen Knoten im Netz.
targetOverlayTerminalNum	Anzahl der maximalen Knoten im Netz.
creationProbability	Wahrscheinlichkeit, dass ein neuer Knoten dem Netz beitritt.
migrationProbability	Wahrscheinlichkeit, dass ein Knoten eine neue IP-Adresse erhält.
removalProbability	Wahrscheinlichkeit, dass ein Knoten das Netz verlässt.

Tabelle 5.1: Parameter des OverlayConfigurators

Durch OMNeT++ besitzt OverSim eine Benutzeroberfläche, mit welcher sich jeder Parameter des Overlaynetzes manuell konfigurieren lässt. Abbildung 5.1 zeigt ein Übersichtsfenster einer Overlaysimulation. Auch das Starten einer Simulation über die Kommandozeile ist möglich. In diesem Fall müssen alle Parameter in Initialisierungsdateien (sogenannte INI Dateien) angegeben werden. Die API und die Module von OverSim befinden sich noch im Entwicklungsstadium. OverSim wird zum ersten Mal im Rahmen einer Diplomarbeit verwendet, was zu sehr vielen Verbesserungsvorschlägen führte. Diese konnten allerdings innerhalb des kurzen Zeitraums nicht vollständig eingebracht werden.

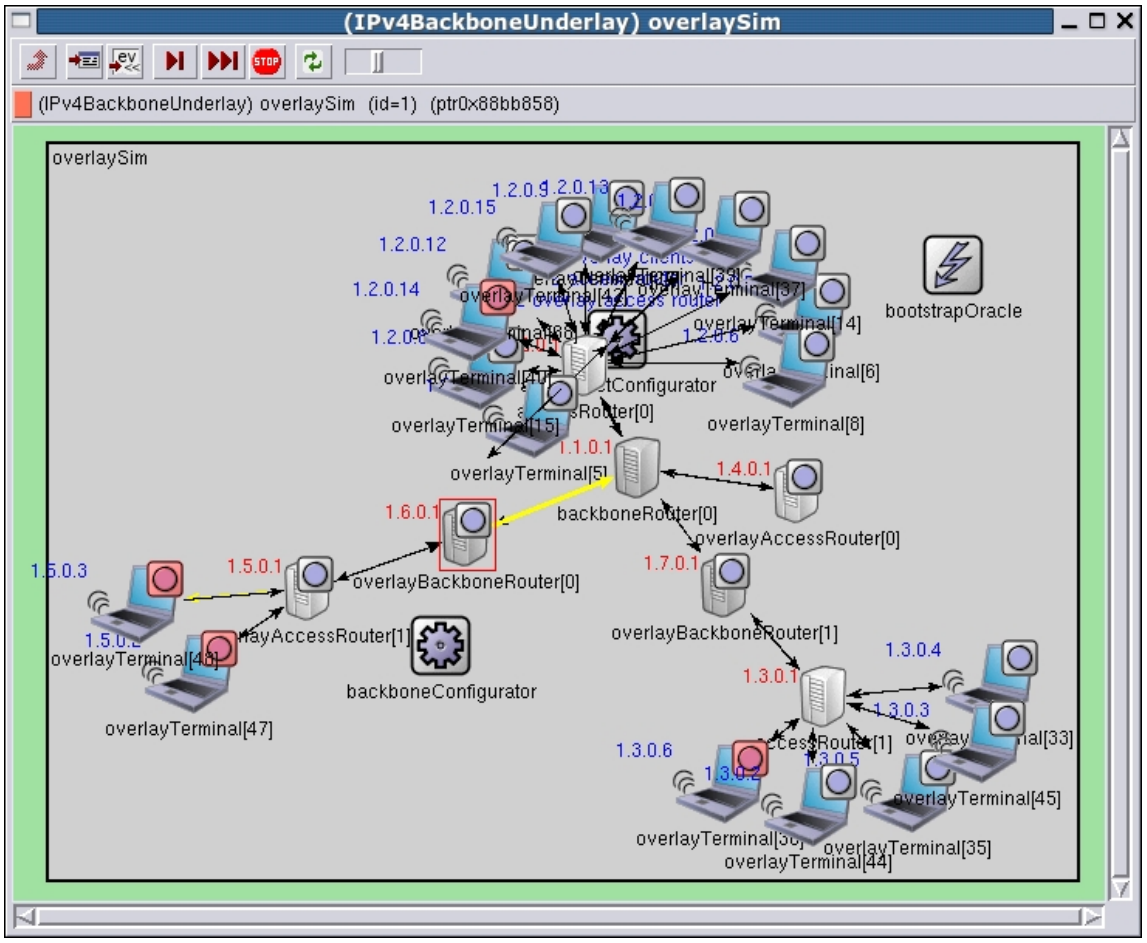


Abbildung 5.1: Eine Bildschirmaufnahme von OverSim

5.2 Integration in OverSim

Um die Simulation mit OverSim zu ermöglichen, wird zunächst das BaseOverlay um einen RPC-Dienst erweitert welcher die weitere Implementierung vereinfacht. Dann wird das ursprüngliche Kademlia-Protokoll implementiert und um die Nachbarschaftsliste sowie den parallelen Lookup über disjunkte Pfade erweitert. Im Folgenden werden die Klassen der Implementierung kurz vorgestellt:

Kademlia: Die Klasse `Kademlia` ist von der Klasse BaseOverlay abgeleitet und ist dafür zuständig eingehende RPCs abzuarbeiten.

RoutingTable: Die Klasse `RoutingTable` implementiert die Routingtabelle und die Nachbarschaftsliste. Sie wird je Knoten einmalig durch die Klasse Kademlia instanziiert.

KademliaLookup: Die Klasse `KademliaLookup` verwaltet die parallelen Pfadlookups und die bereits besuchten Knoten.

KademliaLookupPath: Die Klasse `KademliaLookupPath` wird durch die Klasse `KademliaLookup` je nach Anzahl der zu verfolgenden disjunkten Pfade mehrfach instanziiert und führt jeweils einen Lookup über genau einen disjunkten Pfad durch. Das Ergebnis dieses Lookups wird an `KademliaLookup` zur Bearbeitung übergeben.

Die Beziehungen dieser Klassen sind in Abbildung 5.2 zu sehen. Die Kontakte, welche für die Routingtabelle benötigt werden, wurden wie in Abschnitt 4.2.1 beschrieben in einer Klassenhierarchie realisiert (siehe Abbildung 5.3). Desweiteren implementiert

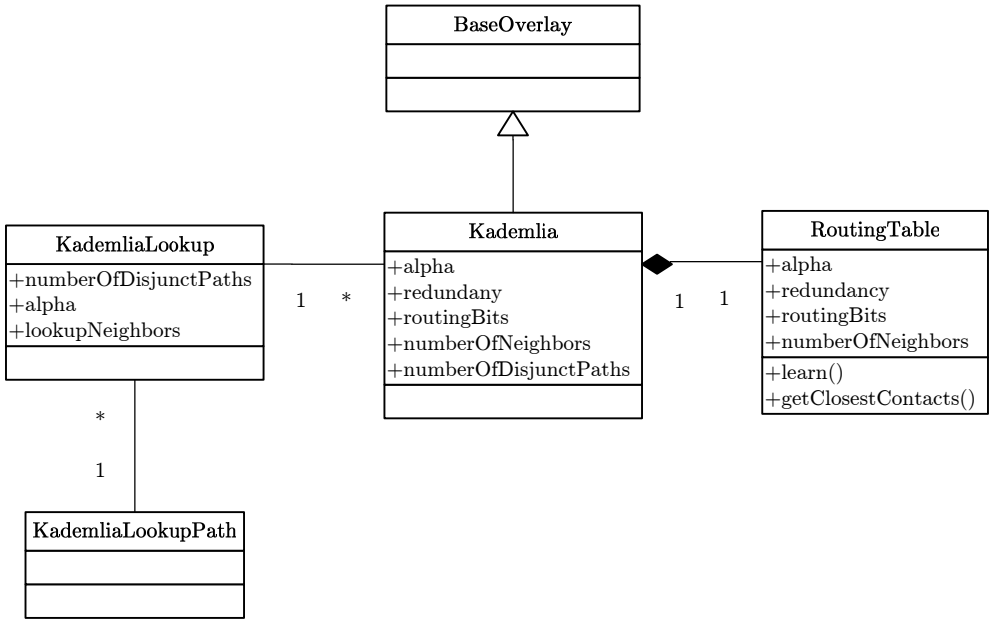


Abbildung 5.2: Das grobe Klassendiagramm der Implementierung

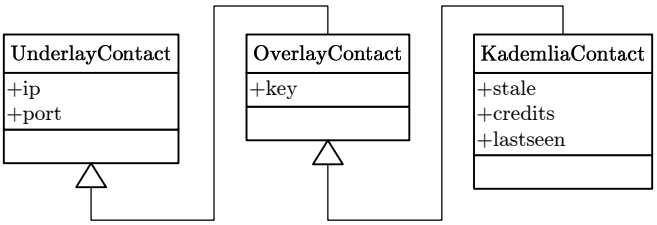


Abbildung 5.3: Klassendiagramm der Kontakte

die Klasse `Kademlia` eine OverSim-Komponente. Diese Komponente kann mit Hilfe einer Initialisierungsdatei parametrisiert werden. Die Parameter sowie ihre äquivalenten Bezeichner, welche in dieser Arbeit verwendet werden, sind in Tabelle 5.2 aufgelistet.

Parameter	**Sym**	**Beschreibung**
alpha	α	Anzahl der gleichzeitig zu befragenden Knoten bei jedem Lookupschritt
redundancy	k	Anzahl der verfügbaren redundanten Wege
routingBits	b	Anzahl der zu berücksichtigenden Bits in der Routingtabelle
numberOfDisjunctPaths	d	Anzahl der disjunkten Pfade beim Lookup
numberOfNeighbors	n, k'	Anzahl der Nachbarn

Die Spalte „Sym“ zeigt die, in dieser Arbeit verwendeten Bezeichnungen

Tabelle 5.2: Simulationsparameter der vorgestellten DHT

5.3 Besonderheiten der Implementierung

In diesem Abschnitt werden Besonderheiten wie Annahmen, Verfeinfachungen, die Simulation bösartiger Knoten sowie der Simulationsablauf beschrieben.

Annahmen und Vereinfachungen

Die Simulation wird aufgrund des limitierten Speicherplatzes und der zur Verfügung stehenden Rechenzeit möglichst effizient gehalten. So wird auf die Implementierung der Sicherheitsschicht verzichtet und lediglich die Länge der Signaturen in den Nachrichten berücksichtigt um eine Evaluierung des Kommunikationsaufwands zu ermöglichen. Da die Knotenfluktuation im Netz in dieser Arbeit eine untergeordnete Rolle einnimmt, werden alle Simulationen mit $\alpha = 1$ ausgeführt und es wird davon ausgegangen, dass sich das Netz nach dem Aufbau mit N Knoten nicht mehr verändert.

Desweiteren ist es für die Evaluierung ausreichend zu erkennen, ob Daten sicher abgelegt werden können. Dazu genügt es, wenn bekannt ist, ob ein Knoten bösartig ist oder nicht. Deshalb wird auf die reale Datenspeicherung auf den einzelnen Knoten verzichtet. Durch den eingesparten Speicherplatz kann eine größere Anzahl Knoten simuliert werden.

Simulation bösartiger Knoten

Eine Beurteilung der Sicherheit des hier vorgestellten dezentralen Peer-to-Peer Voice-over-IP Systems basiert auf dem Anteil der erfolgreichen Anrufe. Dazu muss ein Teilnehmer zunächst auffindbar und dann erreichbar sein. Für die Auffindbarkeit wird der Datenlookup benötigt, um eine *NodeID* zu einem gegebenen Namen zu finden. Der Knotenlookup ist notwendig um den Teilnehmer unter der *NodeID* erreichen zu können. Im Folgenden soll deshalb ein „worst-case“-Szenario vorgestellt werden, bei dem der Angreifer bezüglich Daten- und Knotenlookup maximalen Schaden anrichtet.

Einem Angreifer gelingt es ausschließlich mit sehr viel Aufwand eine komplette Nachbarschaft einer *ID* zu kontrollieren. Im schlimmsten Fall kann der Angreifer somit lediglich versuchen, durch die Rückgabe bösartiger Knoten einen Lookup in ein eigenes paralleles Netz umzuleiten. Dies führt dazu, dass Daten in dem parallelen Netz des Angreifers gespeichert werden oder ein gesuchter Knoten nicht gefunden werden kann. Gibt der Angreifer keine Knoten zurück, so bleibt er in der Suche unberücksichtigt.

Für die Simulation bedeutet dies, dass ein Lookup immer dann fehlschlägt, wenn ein bösartiger Knoten auf einem der benutzten Pfade liegt. Der Lookup wird in diesem Fall einfach abgebrochen, da davon ausgegangen werden kann, dass weitere Ergebnisse nicht mehr von Nutzen sein werden. Desweiteren kann bei einem Knotenlookup davon vorausgesetzt werden, dass lediglich nach gutartigen Knoten gesucht wird, da es der Praxis widerspricht ein Telefonat mit einem bösartigen Knoten zu führen.

Für den Datenlookup liegt der schlimmste Fall vor, wenn alle Daten in gleicher Weise verändert werden. Dies richtet maximalen Schaden an, denn die Mehrheitsentscheidung fällt in diesem Fall zu Gunsten der modifizierten Daten aus. Es wird davon

ausgegangen, dass ein Datenlookup fehlschlägt, wenn die Mehrzahl der Replikate auf bösartigen Knoten liegt.

Besonderheiten der Simulation

Kademlia verwendet eine verallgemeinerte Hyperkubus Topologie. Dadurch sind die Pfadlängen nicht wie bei DeBruijn Topologien weitestgehend konstant, sondern normalverteilt [LKRG03b]. Diese Normalverteilung gilt allerdings nur, wenn $k = 1$ und $b = 1$ sind. Diese Parameter verändern die Pfadlänge derart, dass eine Vorhersage der Pfadlängen bzw. deren Verteilung nahezu unmöglich ist. Um dennoch die theoretischen Werte berechnen zu können, muss die Hopverteilung protokolliert werden.

Durch die Eigenschaften von Kademlia ergibt sich bezüglich der Simulationsparameter eine Einschränkung: durch die Struktur der Routingtabelle von Kademlia, ein Netzaufbau mit $k < 2$ mit hoher Wahrscheinlichkeit nicht möglich. Dies liegt an der LRU-Strategie, welche von Kademlia verfolgt wird. Deshalb werden die Simulationen immer mit einem $k > 1$ durchgeführt.

Ablauf der Simulation

Um die Simulation effizient zu gestalten, wird zunächst ein statisches Netz mit N Knoten aufgebaut. Danach werden Daten- und Knotenlookups durchgeführt um den Anteil der auffindbaren und erreichbaren Knoten zu ermitteln. Damit das Netz nicht neu aufgebaut werden muss, werden sukzessive 5% zufällig gewählte Knoten mit der Eigenschaft „bösartig“ markiert und die Lookups wiederholt. Diese Markierungen beeinflussen die Lookups wie beschrieben. Im Folgenden wird der Simulationsablauf schrittweise dargestellt:

1. Netzaufbau mit N Knoten.
2. Stabilisierungsphase, in der bei jedem Knoten mindestens ein Wartungsintervall der Routingtabelle abgewartet wird.
3. Datenlookup mit Mehrheitsentscheidung von N zufällig gewählten Schlüsseln. Ist der Anteil der bösartigen Knoten Null, so werden weitere $15N$ Lookups durchgeführt. Dies soll später das Problem der Lücken im *ID* Raum zeigen.
4. Knotenlookup von N gutartigen Knoten.
5. Erhöhung des Anteils bösartiger Knoten um 5%
6. Wiederholung der Schritte 3-5 bis 90% der Knoten bösartig sind.
7. Beenden der Simulation und speichern der Simulationsergebnisse.

Bei jeder Wiederholung wird der Kommunikationsaufwand, der Anteil der jeweils erfolgreichen Daten- und Knotenlookups und die Hopverteilung protokolliert.

6. Evaluierung

In diesem Kapitel wird der Entwurf aus Kapitel 4 anhand der Implementierung im Simulator evaluiert. Insbesondere wird der Anteil der erfolgreichen Knoten- und Datenlookups im Fall eines Angriffs durch bösartige Knoten untersucht.

6.1 Simulation des Overlays

Im Folgenden werden die Ergebnisse der durchgeführten Simulationen dargestellt und erläutert. Es wurden mehrere Simulationen, wie in Abschnitt 5.3 beschrieben, mit $N = 1000$, 5000 und 10000 Knoten durchgeführt. Zusätzlich wurden auch ausgewählte Simulationen mit $N = 40000$ Knoten durchgeführt. Für $N = 10000$ werden die Ergebnisse im Folgenden ausführlich erläutert. Die weiteren Simulationsergebnisse sind im Anhang A.1 zu finden.

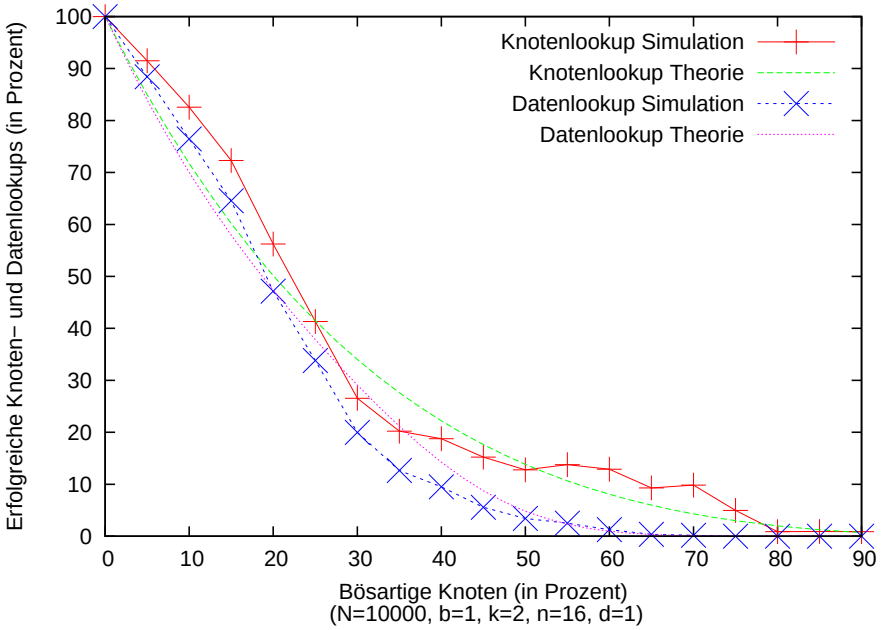
6.1.1 Sichere Daten- und Knotenlookups

Zunächst soll gezeigt werden, wie die Anzahl der verfolgten disjunkten Pfade bei Daten- und Knotenlookups den Erfolg (siehe Abschnitt 5.3) einer Lookupoperation beeinflusst. Dazu wird anfangs der einfache Fall mit nur einem Pfad betrachtet. Dieses Ergebnis spiegelt das Verhalten normaler DHTs mit Replikationsmaßnahmen und Mehrheitsentscheidungen wider.

Abbildung 6.1 zeigt dieses Simulationsergebnis. Im oberen Graphen werden die erfolgreichen Daten- und Knotenlookups in Prozent unter Einfluss von 0-90% bösartiger Knoten angegeben. Zum Vergleich werden die theoretischen Werte aus den Abschnitten 3.3.3 und 3.4.3 dargestellt. Der untere Graph zeigt das Histogramm der Hopverteilung.

Wird nun die Anzahl der parallel durchgeführten Lookups d erhöht (siehe Abbildung 6.2), so steigt der Anteil der erfolgreichen Lookups an. Der Parameter k wurde stets doppelt so groß gewählt wie die Anzahl der parallel durchgeführten Lookups. Dies stellt sicher, dass im Netz ausreichend disjunkte Pfade vorhanden sind. Es fällt auf, dass die durchschnittlich benötigte Anzahl Hops sinkt. Durch die höhere Anzahl Kontakte pro k-Bucket (größerer Parameter k) ist es wahrscheinlicher, dass

ein Knoten schon früher gefunden wird. Die Ergebnisse der Simulationen stimmen bis auf wenige Ausreißer sehr gut mit den theoretisch ermittelten Werte überein.



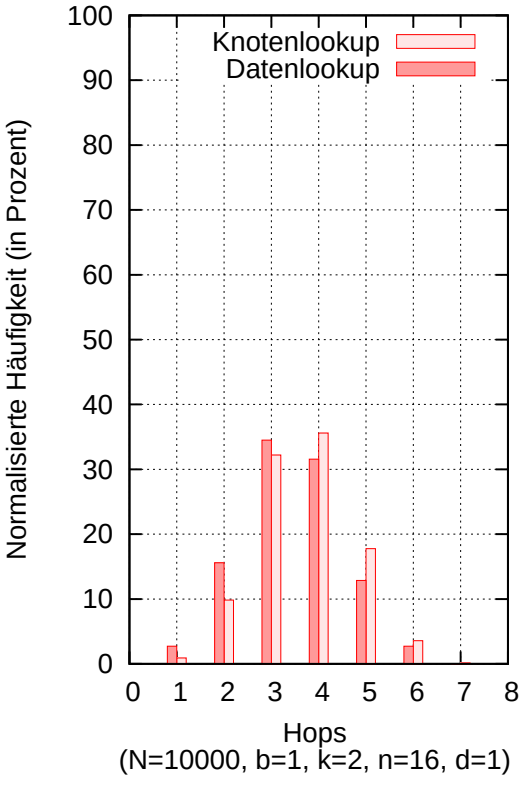


Abbildung 6.1: Simulation mit 10000 Knoten ohne Lookup über mehrere disjunkte Pfade mit $b = 1$, $k = 2$ und $d = 1$

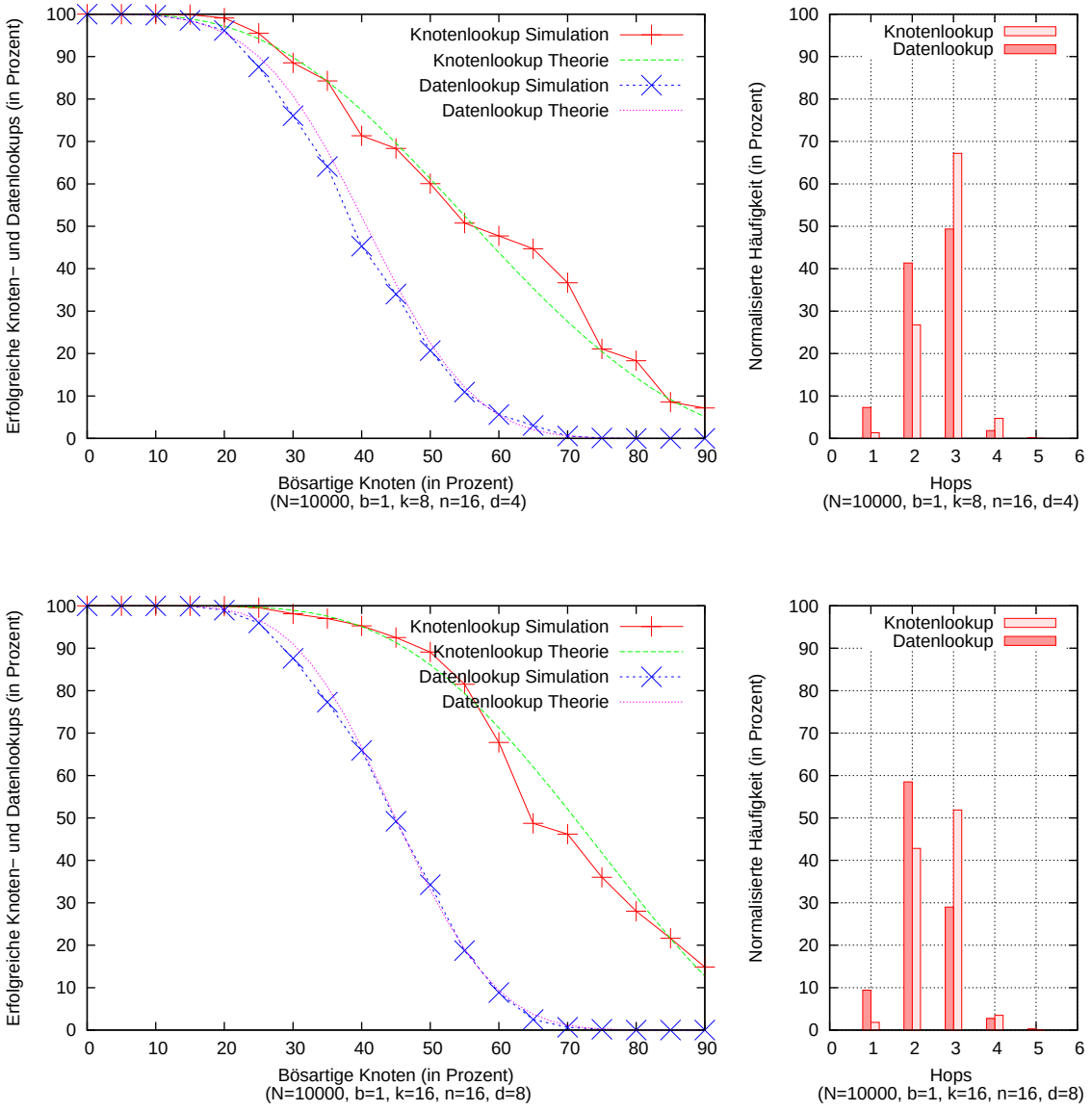


Abbildung 6.2: Simulation mit 10000 Knoten, $b = 1$, $k = 8$ und $d = 4, 8$

6.1.2 Kommunikationsaufwand der Lookups

In diesem Abschnitt soll gezeigt werden, wieviel Kommunikationsaufwand durch Daten- und Knotenlookups über mehrere Pfade entsteht. Die Simulationen berücksichtigen dabei auch den Kommunikationsaufwand der Signaturen, welche durch die Sicherheitsschicht (siehe Abschnitt 4.1) erstellt werden. Abbildung 6.3 zeigt den anfallenden Datenverkehr und die Anzahl *aller* Nachrichten bei mehreren disjunkten Lookups mit $N = 10000$ Knoten, $k = 16$ und $n = 16$. Wie erwartet steigt der Aufwand für den Knotenlookup linear mit der Zahl der verwendeten Pfade d. Dies impliziert auch, dass die Pfade ungefähr gleich lang sein müssen, denn ein frühzeitiges Auffinden eines Knotens führt zum Abbruch des Suchvorgangs. Beim Datenlookup fällt deutlich mehr Kommunikationsaufwand an als beim Knotenlookup, denn nachdem die Nachbarschaft einer *ID* gefunden wurde, wird zusätzlich jeder noch unbekannte Knoten angepingt. Da durch andere parallel durchgeführte Lookups bereits Knoten in die Nachbarschaftsliste des Lookups aufgenommen werden, wird die

Wahrscheinlichkeit, dass sich bereits alle Knoten in der Nachbarschaftsliste befinden, immer größer. Deshalb müssen mit steigender Anzahl der disjunkten Lookups am Ende eines Pfads immer weniger Knoten angepingt werden. Dies erklärt den nichtlinearen Anstieg der Nachrichten. Da die Ping-Nachrichten sehr klein sind, haben diese kaum Einfluss auf den Kommunikationsaufwand in Bytes.

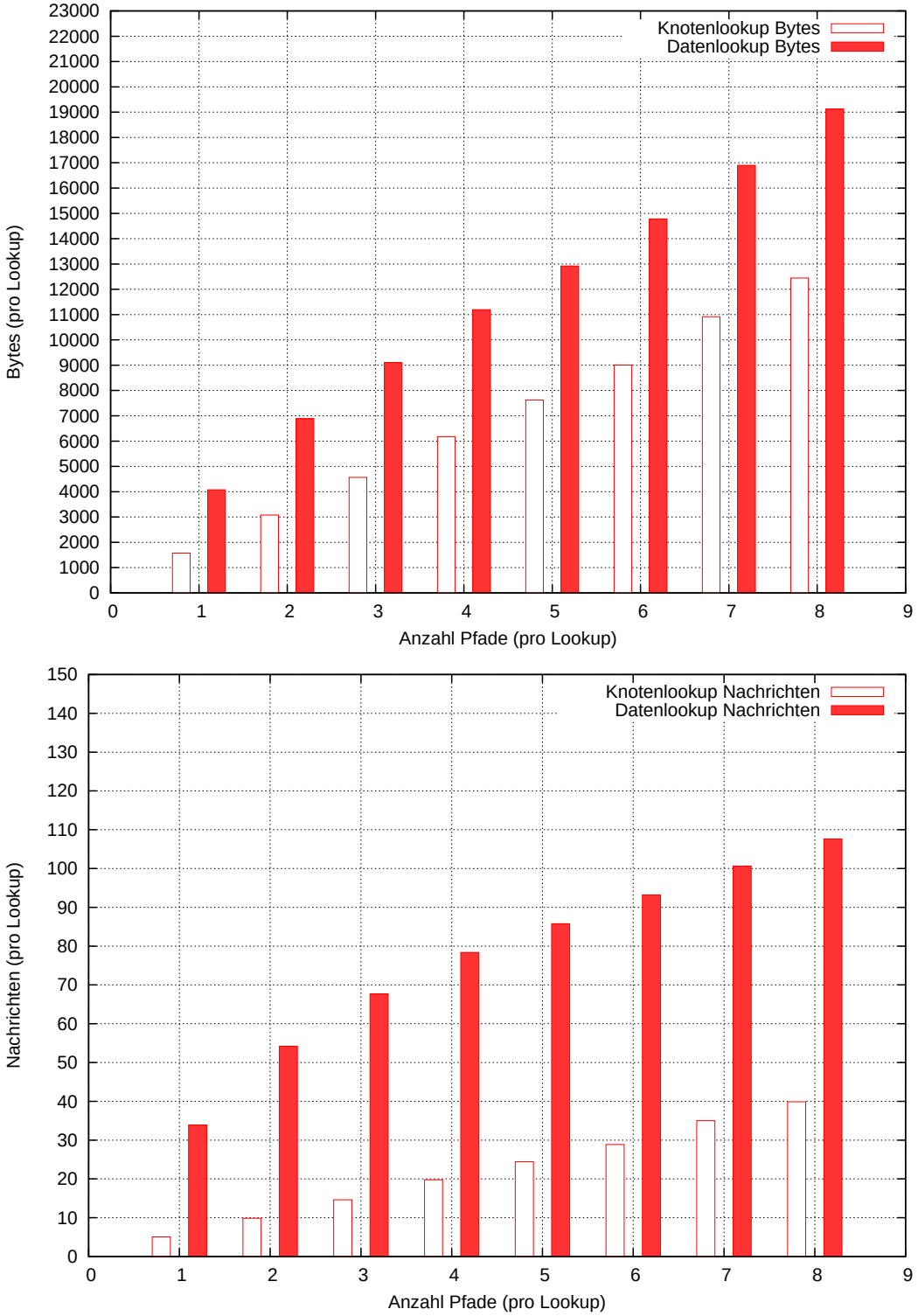


Abbildung 6.3: Anfallender Datenverkehr und Anzahl der Nachrichten bei mehreren disjunkten Pfaden mit $N = 10000$ Knoten, $k = 16$ und $n = 16$

6.1.3 Verringerung des Netzdurchmessers

Im Folgenden soll der Einfluss des Parameters b des Key-based Routings untersucht werden. Die Erhöhung von b verkleinert den Durchmesser des Graphen und damit die benötigte Hopanzahl. Aufgrund dessen, sollte der Erfolg von Lookupoperationen zunehmen. Für die Simulation wird die Anzahl der disjunkten Lookups $d = 8$, die Redundanz $k = 16$ und die Anzahl der Nachbarn $n = 16$ verwendet. Abbildung 6.4 zeigt die Verbesserung des Ergebnisses von $b = 1$ nach $b = 2$. Diese fällt erwartungsgemäß sehr gering aus, denn durch den Parameter $k = 16$ und die geringe Knotenanzahl von $N = 10000$ ist die Pfadlänge für $b = 1$ bereits sehr klein.

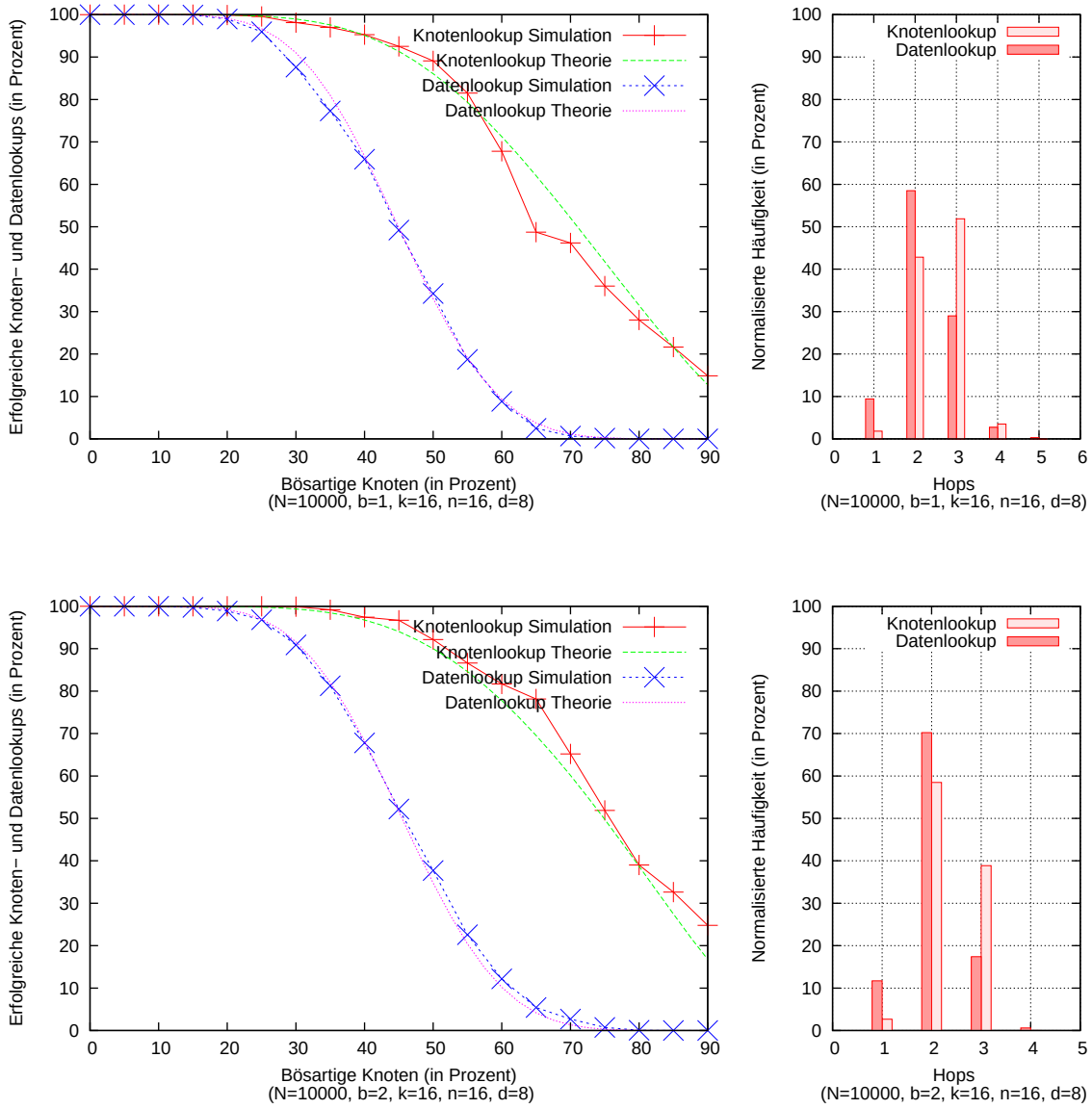


Abbildung 6.4: Einfluss des Parameters b für $b = 1$ und $b = 2$

Erwartungsgemäß werden die Ergebnisse für $b = 4$ auch weiterhin geringfügig besser. Eine Ausnahme ergibt sich allerdings für $b = 5$ in Abbildung 6.5. In diesem Ergebnis weichen die theoretischen von den simulierten Ergebnissen des Knotenlookups deutlich ab.

Dieses Phänomen lässt sich wie folgt erklären: Wie in dem Histogramm zu sehen ist gehen Knotenlookups hier nur noch über 2 Hops. Abzüglich des zweiten Hops, welcher in jedem Fall zu einem gutartigen Knoten führt, bleibt nur die Wahl des ersten Hops aus der eigenen Routingtabelle. Hier sind, im Gegensatz zu den theoretischen Überlegungen, nur die eigenen 8 naheliegendsten Knoten zu einer *NodeID* aus der Routingtabelle entscheidend. Es fehlt die „Mittelung“ über mehrere Hops. Die Wahrscheinlichkeit, dass einer der 8 Knoten gutartig ist, kann jedoch einfach durch $1 - m^8$ nachgerechnet werden. Exemplarisch ist die Wahrscheinlichkeit für $m = 0.90$ gerade 0.57 und für $m = 0.80$ mit 0.83 gegeben. Dies entspricht wiederum ziemlich genau den Simulationsergebnissen. In der Praxis dürfte dieser Fall jedoch selten auftreten.

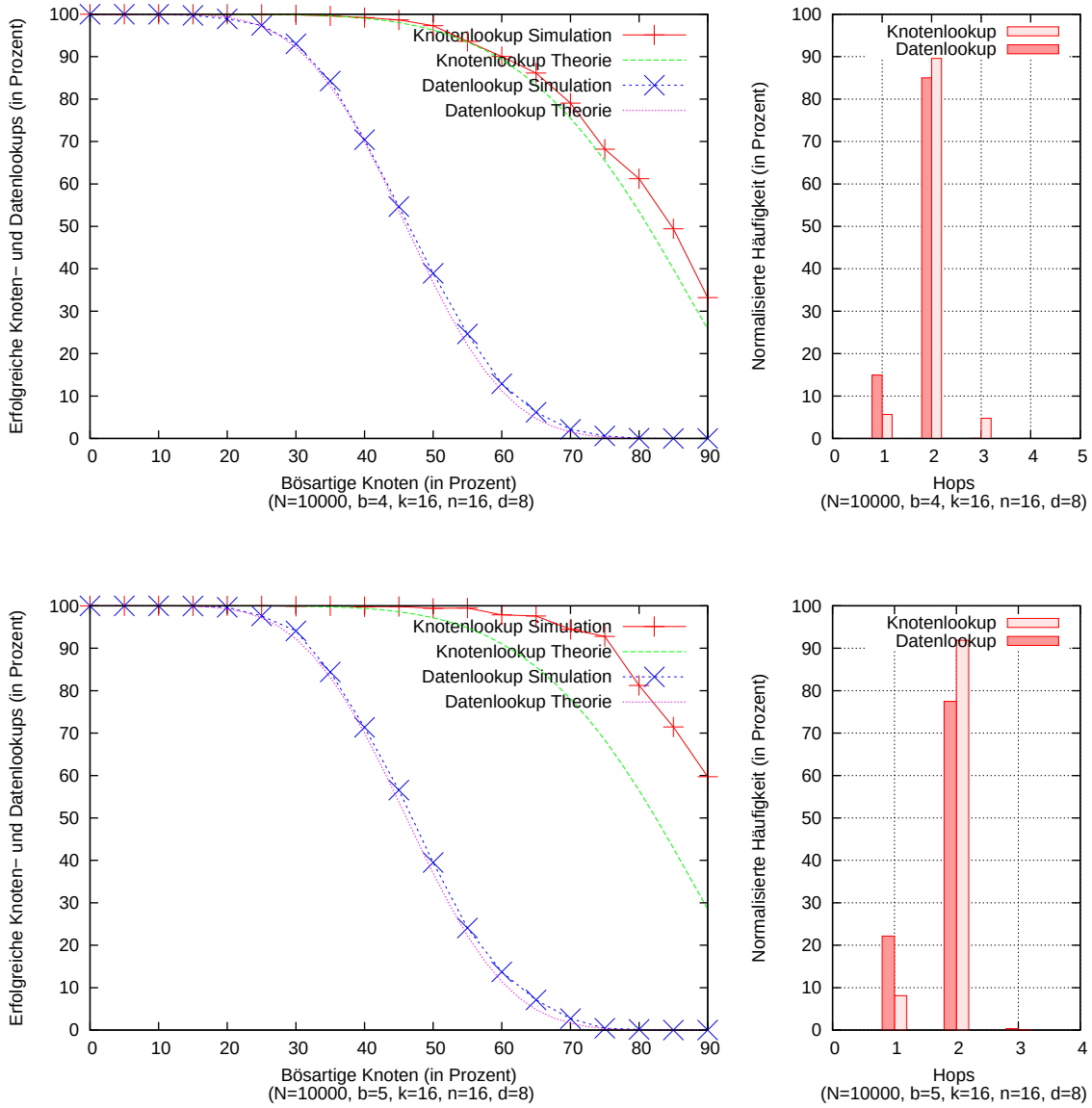


Abbildung 6.5: Einfluss des Parameters b für $b = 4$ und $b = 5$

6.1.4 Verringerung der Nachbarschaftsknoten

Bei zu geringer Überlappung der Nachbarschaften kommt es zu Lücken im *ID*-Raum (siehe Abschnitt 4.2.1). Dies führt dazu, dass Datenlookups fehlschlagen, falls die zufällig gewählten *ID*s eine oder mehrere dieser Lücken treffen. Das sich dies auch in der Simulation widerspiegelt, zeigt Abbildung 6.6 für zwei verschiedene Netzgrößen ($N = 5000$ und 10000) und Nachbarschaftsgrößen ($n = 2$ und $n = 4$).

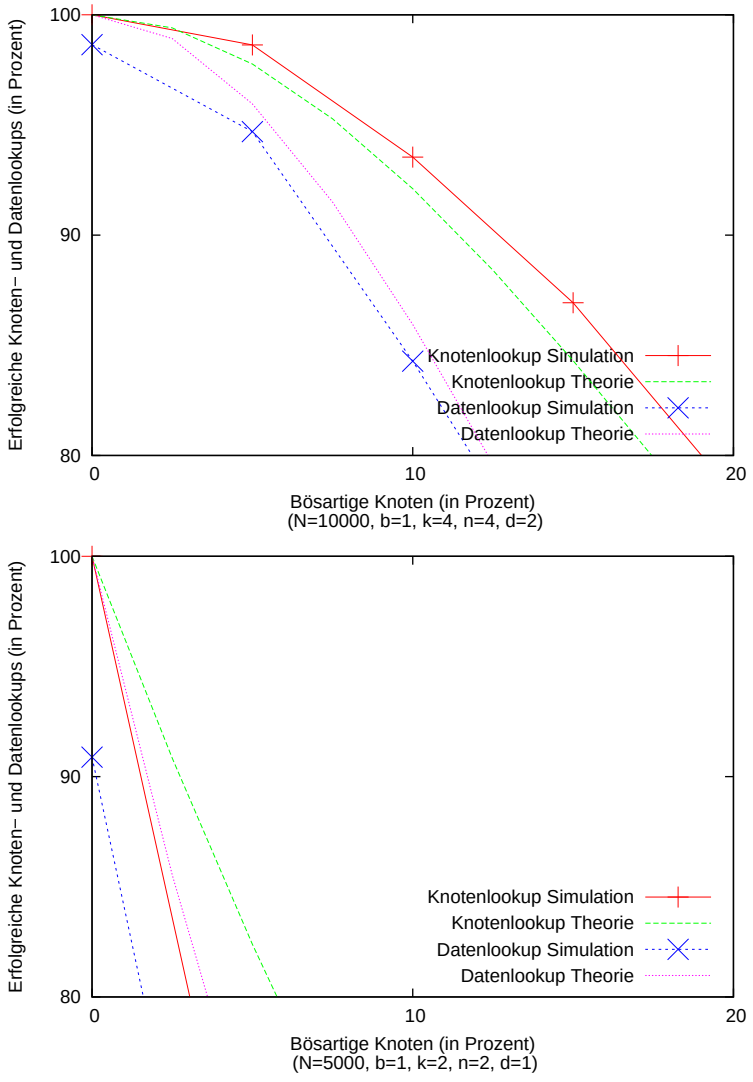


Abbildung 6.6: Lücken des *ID*-Raums für kleine n

6.1.5 Lookups bei geringer Redundanz

In diesem Abschnitt soll gezeigt werden, wie sehr die Anzahl der möglichen redundanten Pfade k in die Erfolgsrate der Lookups eingeht. Dazu wird in den folgenden Simulationsergebnissen $k = 2$ gewählt. Dadurch, dass die Anzahl der möglichen redundanten Pfade für $d > k$ parallel durchgeführte Lookups nicht ausreicht, wird erwartet, dass Anteil erfolgreicher Lookups nicht sehr stark ansteigt. Desweiteren bleibt die durchschnittliche Hopanzahl konstant. Dieses Verhalten ist auch in Abbildung 6.7 zu sehen.

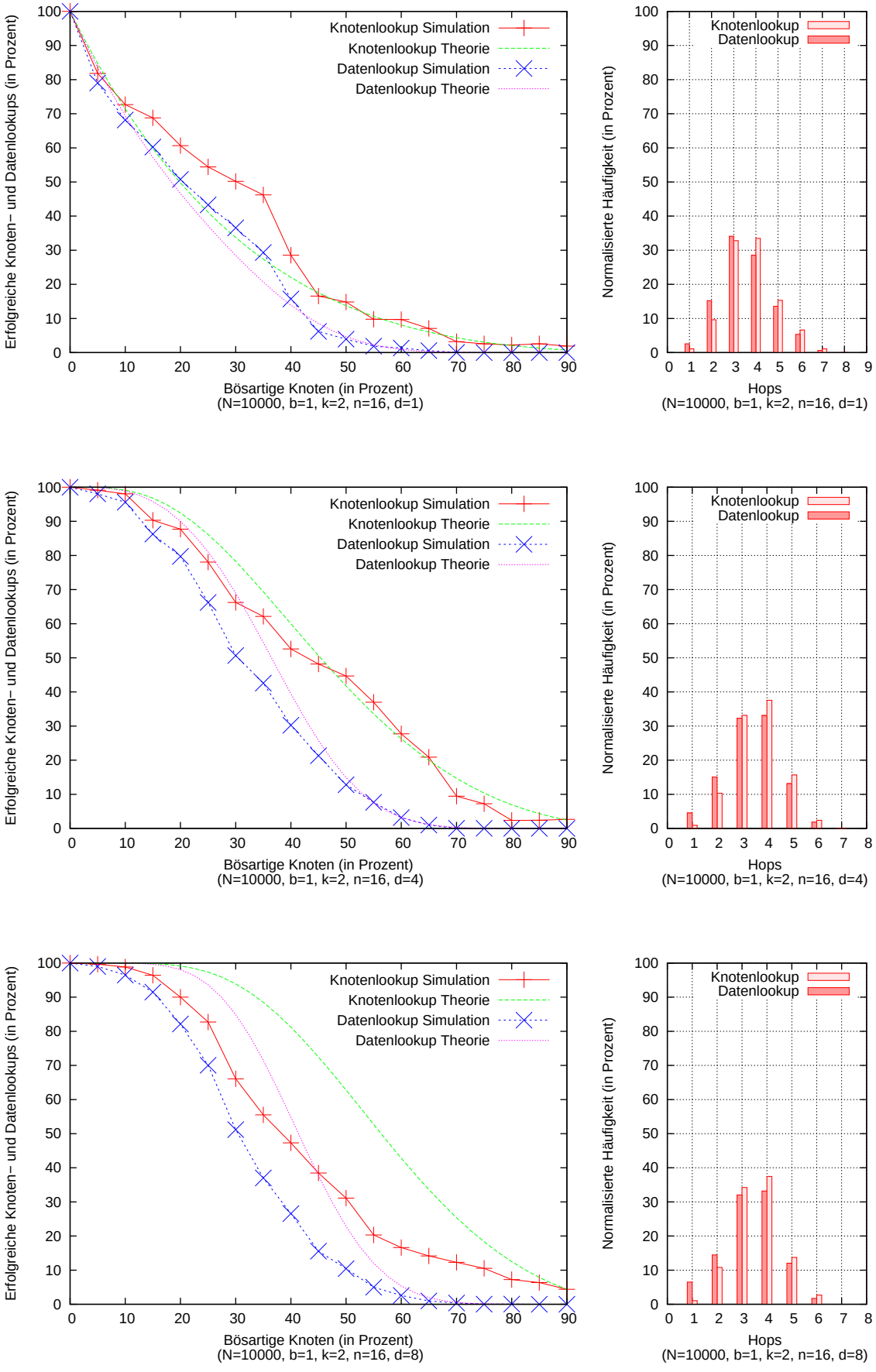


Abbildung 6.7: Mehrfache Pfade unter der Verringerung der Redundanz für $N = 10000$, $b = 1$, $k = 2$, $n = 16$ und $d = 1, 4, 8$

6.1.6 Simulationen mit 40000 Knoten

Wie schon zuvor erwähnt, wurden auch zwei Durchläufe mit $N = 40000$ Knoten, $b = 1$, $k = 16$, $d = 2$ und $d = 8$ durchgeführt. Diese Simulationen benötigen viel Rechenzeit und verbrauchen ca. $5,5$ GB Speicher. Abbildung 6.8 zeigt die Simulationsergebnisse. Da aufgrund des großen Rechenaufwands nur jeweils eine Simulation mit einem Seed durchgeführt werden konnte, weichen die Ergebnisse teilweise von den theoretischen Berechnungen ab.

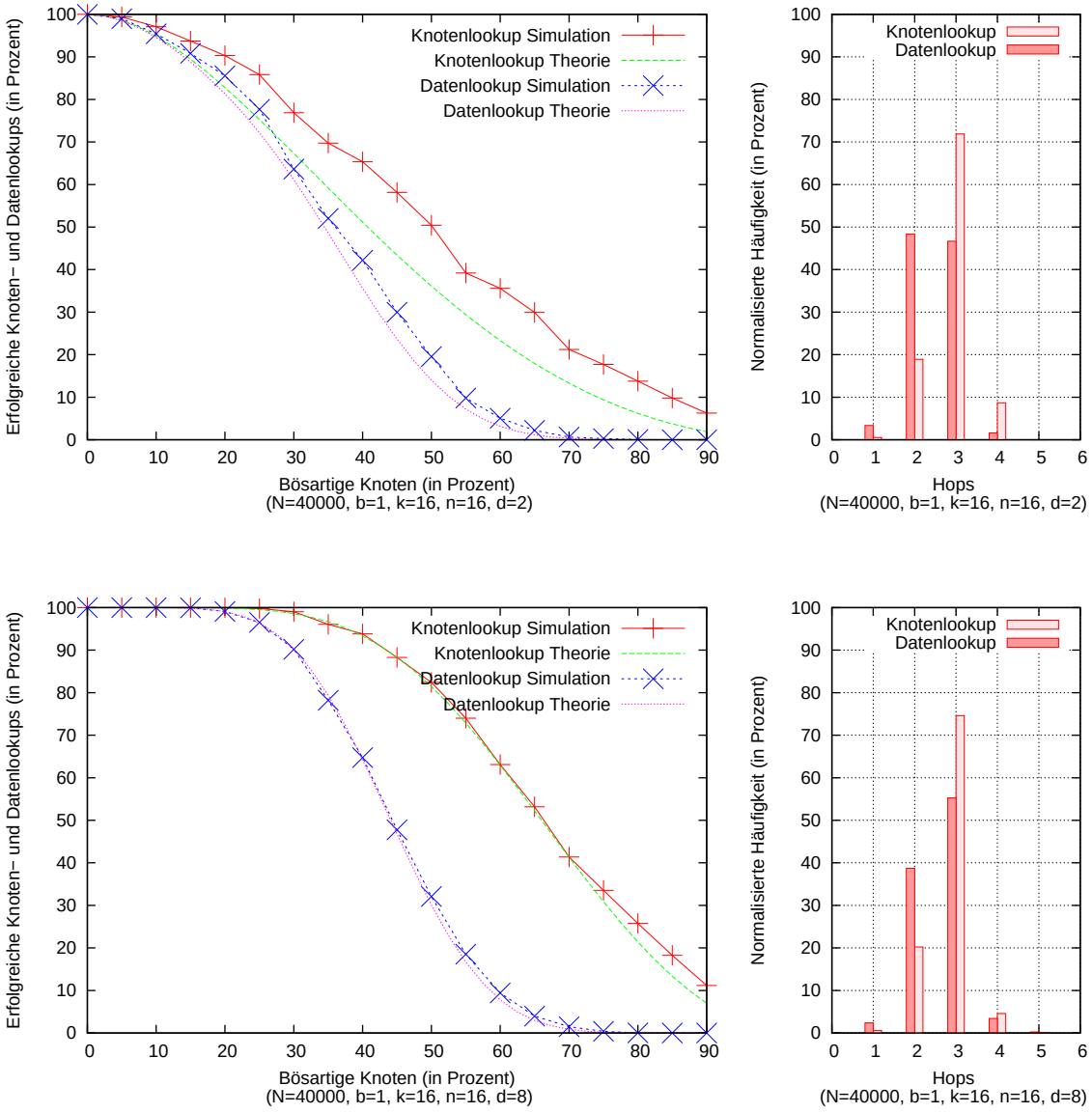


Abbildung 6.8: Simulationen mit $N = 40000$, $b = 1$, $k = 16$, $n = 16$ und $d = 2, 8$

6.2 Simulation der Netzgrößenabschätzung

Die Netzgrößenabschätzung wird für die Sicherheitsschicht (siehe Abschnitt 4.1) benötigt und wurde gemäß dem Entwurf in Abschnitt 4.2.5 implementiert. Für die Simulation wurden dem Netz sukzessive bis zu $N = 30000$ Knoten hinzugefügt. Zwischen jedem Einfügevorgang werden pro Knoten r RPCs aufgerufen, welche jeweils die eigene geschätzte Netzgröße mit der des anderen Knotens mitteln. Die Abbildungen 6.9 und 6.10 zeigen, dass so die geschätzte Netzgröße gegen die reale Netzgröße konvergiert. Mit steigender Rate r steigt die Konvergenzgeschwindigkeit.

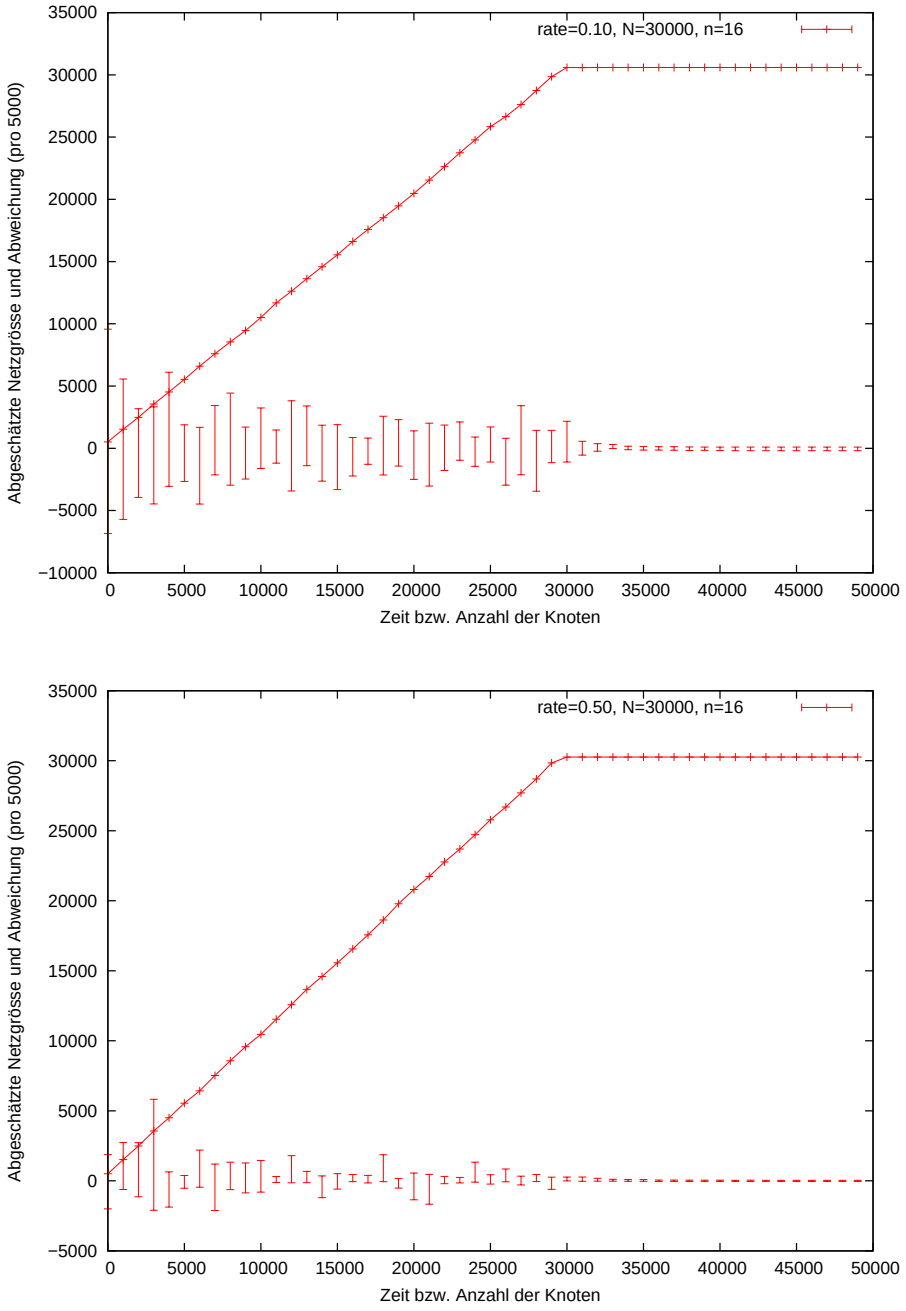


Abbildung 6.9: Abschätzung der Netzgröße mit $N = 30000$ und $r = 0.1, 0.5$

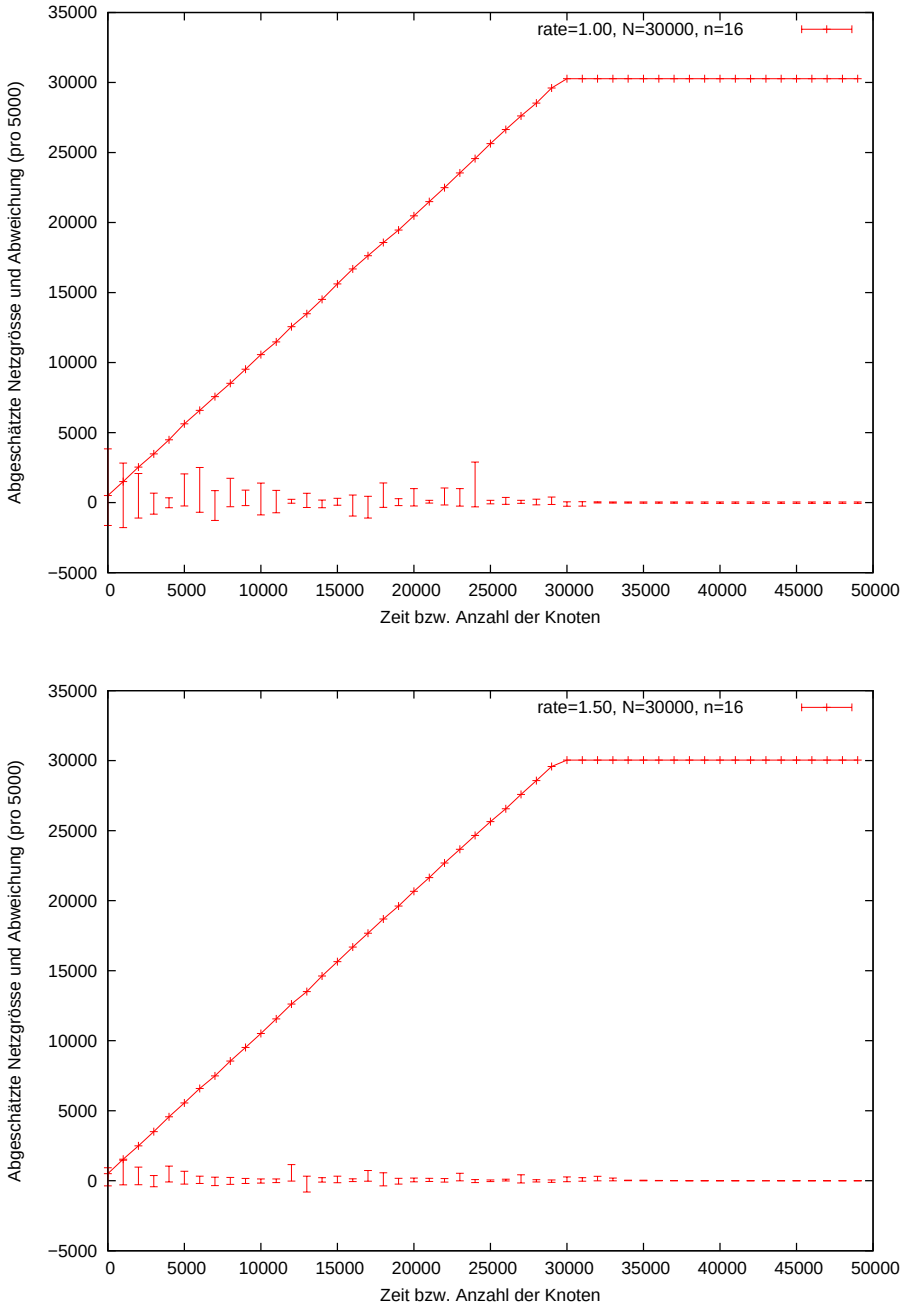


Abbildung 6.10: Abschätzung der Netzgröße mit $N = 30000$ und $r = 1.0, 1.5$

6.3 Bewertung der Ergebnisse

Die Simulationsergebnisse stimmen bis auf kleinere Ausreißer mit der Theorie überein. Auch Anomalien, welche durch die Verringerung des Redundanzparameters k oder der Größe der Nachbarschaftliste n auftauchen, konnten in der Simulation bestätigt werden.

Wie das Simulationsergebnis aus Abbildung 6.8 zeigt, gelingen bei einem Anteil von 20% bösartiger Knoten im Netz noch nahezu 100% der Datenlookups. Ein Knotenlookup ist sogar noch mit 30% bösartigen Knoten erfolgreich. Dieses Ergebnis ist durch Erhöhung des Parameters b ohne weiteres auch für Netze mit mehreren Millionen Knoten zu erreichen. Damit ein Angreifer 20% der Knoten in einem Netz der Größe $N = 100000$ kontrollieren kann, muss dieser $N_A = 20000$ Kryptopuzzles lösen. Wird eine durchschnittliche Bearbeitungszeit von $t = 3$ Stunden pro Kryptopuzzle angenommen, so muss der Angreifer $N_A \cdot t = 7$ Jahre aufwenden um die Sicherheit des Netzes gefährden. In der Praxis bietet dieses Verfahren somit einen ausreichenden Schutz vor Angreifern.

Der Kommunikationsaufwand, welcher durch die Verfolgung mehrerer Pfade entsteht, ist nicht zu verachten. Zwar ist der Anstieg weitestgehend linear, aber für eine ausreichende Erfolgswahrscheinlichkeit der Lookups (für $d > 4$) dennoch relativ hoch. Durch eine dynamische Anpassung dieses Parameters könnte der Aufwand jedoch der jeweiligen Netzsituation angepasst werden. Außerdem könnten bereits durch den Namensdienst aufgelöste Namen auf dem P2P-SIP-Proxy zwischengespeichert werden, um erneute Lookups zu vermeiden. Weiterhin wurde ein Netz ohne Knotenfluktuation betrachtet (siehe Abschnitt 5.3), was der Parameter $\alpha = 1$ impliziert. Durch die Erhöhung dieses Parameters (um die Knotenfluktuation auszugleichen), wird der Kommunikationsaufwand pro Lookup nocheinmal steigen. Hier können aber gegebenenfalls die Überlegungen zur Accordion DHT [JLK] helfen. In der Arbeit wird dieser Parameter adaptiv an die reale Knotenfluktuation angepasst.

Die durchgeführte Simulation der Netzgrößenabschätzung aus Abschnitt 6.2 konvergiert wie erwartet nach wenigen Schritten gegen die reale Netzgröße und stellt somit ein gutes Verfahren zur Netzgrößenabschätzung dar.

7. Zusammenfassung

Peer-to-Peer Systeme erfreuen sich wachsender Beliebtheit. Ihre Anwendung in Filesharingbörsen ist mittlerweile weit verbreitet. Mit Skype wurde erstmals ein Peer-to-Peer System vorgestellt, welches auch Voice-over-IP Gespräche ermöglicht. Allerdings handelt es sich bei Skype um ein hybrides Peer-to-Peer Netz. Bei Skype wird das Auffinden eines Teilnehmers weiterhin mittels zentraler Komponenten realisiert, was weiterhin einen *Single-Point-of-Failure* darstellt.

In der IETF wird momentan an dem Entwurf eines dezentralen Peer-to-Peer Voice-over-IP-Systems gearbeitet. Die dort verfolgten Ansätze basieren auf dem weit verbreiteten Session Initiation Protocol (SIP). Durch den Verzicht auf zentrale Komponenten, ist der Schutz eines solchen Netzes vor Angriffen eine große Herausforderung.

7.1 Ergebnisse dieser Arbeit

In dieser Arbeit wurde eine DHT entworfen, die als Grundlage für ein sicheres dezentrales Voice-over-IP-System dient. Dazu wurden zunächst allgemeine Angriffe auf Peer-to-Peer Systeme analysiert und mögliche Gegenmaßnahmen vorgestellt. Hierzu zählt ein Verfahren zur Wahl der *NodeID*, welche Sybil- und Eclipseangriffe durch Kryptopuzzles erschwert, eine Authentifizierung der Knoten ermöglicht sowie die Integrität von Nachrichten sicherstellt (siehe Abschnitt 3.2 und 4.1).

Auf Basis dieser *NodeID*s wurde das Kademlia-Routing in Abschnitt 3.3.2 so modifiziert, dass es bösartigen Knoten nicht mehr gelingt gezielt Knoten in die Routingtabelle eines anderen Knotens unautorisiert einzufügen. Zusätzlich wurde Kademlia um einen parallelen Knotenlookupmechanismus über mehrere disjunkte Pfade erweitert, welcher einen hohen Schutz vor bösartigen Knoten aufweist (siehe Abschnitt 3.3.3 und 4.2.4). Um dieses Verfahren auf einen Datenlookup übertragen zu können, wurde Kademlia in Abschnitt 4.2.1 um eine Nachbarschaftliste erweitert, welche sicherstellt, dass jeder Knoten alle weiteren replizierenden Knoten zu einem Datum kennt.

Um die Integrität der Daten innerhalb der DHT zu schützen, werden Daten auf n Knoten repliziert und mittels einer Mehrheitsentscheidung sichergestellt, dass bösartige Knoten, welche weniger als die Hälfte der Replikate auf gleiche Art und Weise

verändern, keinen Einfluss auf das Ergebnis eines Datenlookups nehmen können (siehe Abschnitt 3.4.3). Desweiteren werden Replikations- und Insertion-DoS Angriffe beim Speichern eines Schlüssel-Wert Paares verhindert. Dies wird mit Hilfe eines Kryptopuzzles und dem *Grant-Check-Store* (GCS) Verfahren aus Abschnitt 4.3.1 erreicht. Im Gegensatz zu einem reinen Kryptopuzzle-Ansatz, welcher den Angriff lediglich erschweren würde, verhindert dieses Verfahren den Insertion-DoS Angriff vollständig.

Durch die in Abschnitt 4.3.4 vorgestellten Optimierungen können Replikationsangriffe sowie die Modifizierung von Daten durch Angreifer noch weiter erschwert werden. Zusätzlich wurde für das Key-based Routing in Abschnitt 4.2.2 eine Optimierung über Vertrauensbeziehungen vorgestellt, welche geeignet ist einen Sybilangriff zu verhindern. Dazu wurde das soziale Netz eines Voice-over-IP Teilnehmers im Routing berücksichtigt.

Diese Sicherheitsmaßnahmen sind die Grundbausteine, um Voice-over-IP Nutzerdaten sicher abzulegen und Kommunikationsteilnehmer im Netz zu finden. Der Entwurf eines SIP-Proxies auf Basis eines sicheren, dezentralen Namensdienstes, welcher in Abschnitt 4.4 vorgestellt wird, ermöglicht damit die Konstruktion eines dezentralen und autonomen Voice-over-IP Systems.

Um zu zeigen, dass der hier vorgestellte Entwurf die geforderten Sicherheitsmerkmale aufweist, wurden Knoten- und Datenlookups durch Simulationen evaluiert. Sie Simulationsergebnisse bestätigen die Richtigkeit der theoretischen Analysen. Desweiteren zeigen sie, dass die Lokalisation und die Erreichbarkeit eines Teilnehmers trotz eines beachtlichen Anteils bösartiger Knoten gewährleistet ist.

7.2 Ausblick

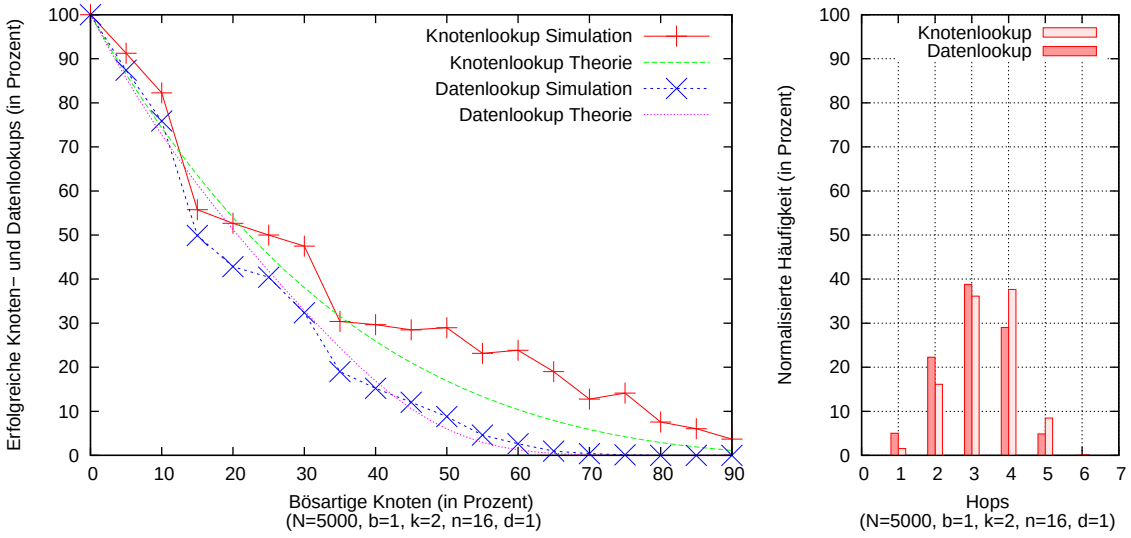
In dieser Arbeit wurde eine sichere DHT für ein dezentrales Voice-over-IP System auf Basis von Kademlia vorgestellt. Durch das reaktive Protokoll von Kademlia treten einige Sicherheitsprobleme erst gar nicht auf. In Zukunft wird es von Interesse sein die hier vorgestellten Sicherheitsmaßnahmen auch auf proaktive DHTs zu übertragen. Da der Fokus dieser Diplomarbeit auf den Sicherheitsaspekten lag, wurde auch nicht weiter untersucht, welchen Einfluss eine hohe Knotenfluktuation auf die replizierten Daten hat.

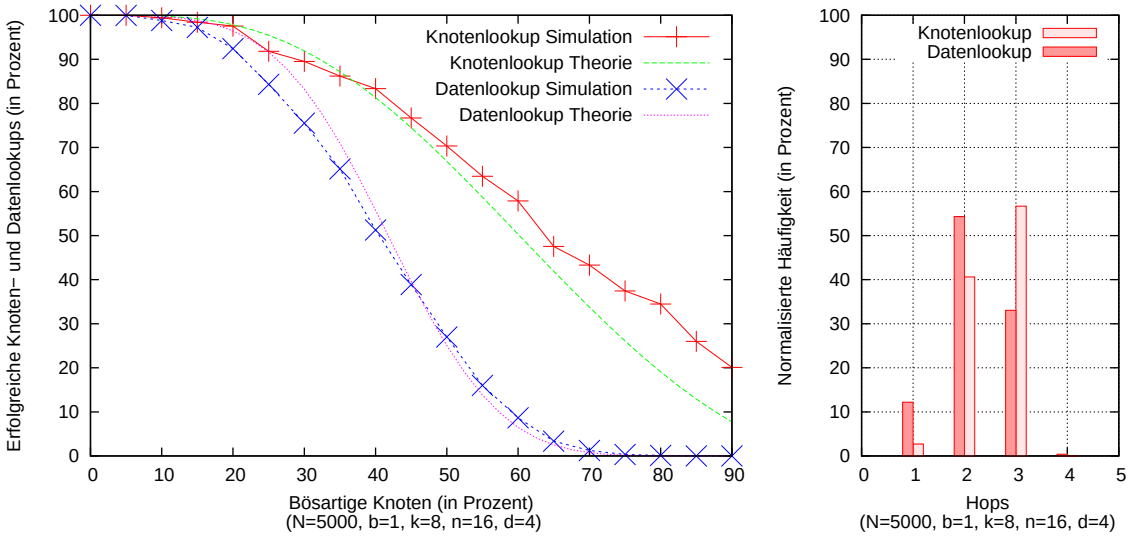
Als problematisch hat sich die Verwendung von zu kleinen Nachbarschaftlisten erwiesen. Durch diese können bei der entworfenen DHT Lücken im *ID*-Raum entstehen, was die Ablage eines Schlüssel-Wert Paares verhindert. Auch wenn eine solche Lücke für ausreichend große Nachbarschaftlisten höchst unwahrscheinlich ist, wäre eine generelle Vermeidung dieser Lücken wünschenswert.

Schließlich ist Kademlia durch die Verallgemeinerung der Hyperkubus-Topologie, ein Peer-to-Peer System der 2. Generation. Neuere Peer-to-Peer Systeme der 3. Generation verwenden eine Topologie auf der Basis von DeBruijn-Graphen. Diese Graphen zeichnen sich durch einen konstanten Ein- und Ausgangsgrad pro Knoten aus. Insbesondere die Verwendung einer auf Broose [GV04] basierenden Topologie wäre somit höchst interessant.

A. Anhang

A.1 Weitere Simulationsergebnisse





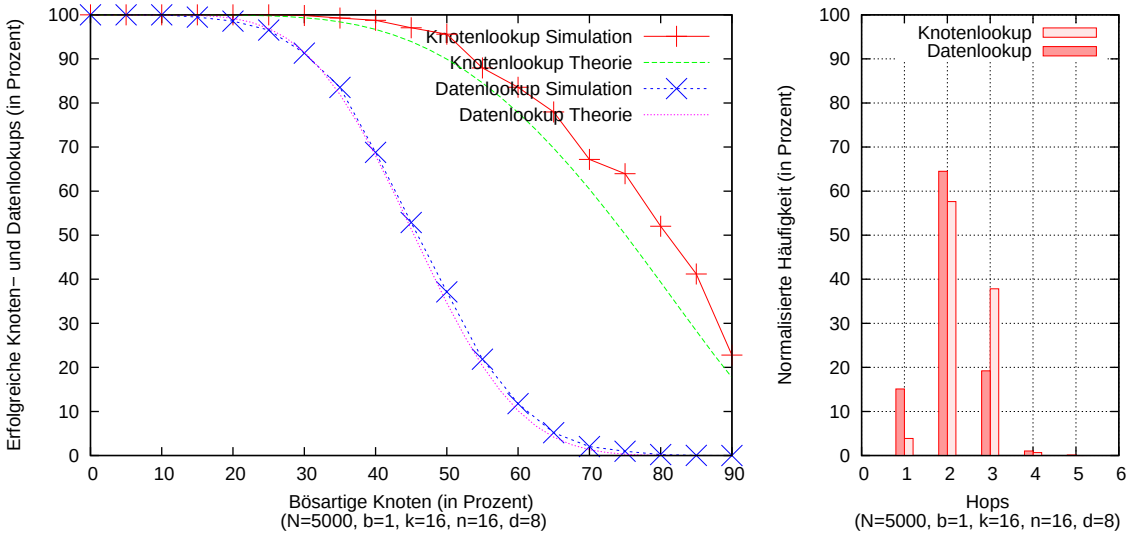
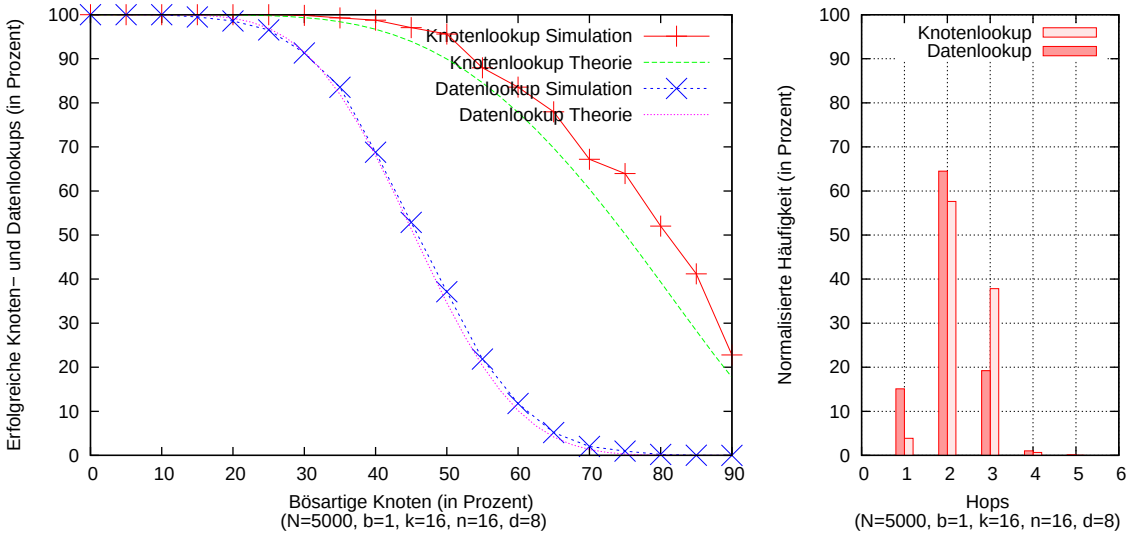
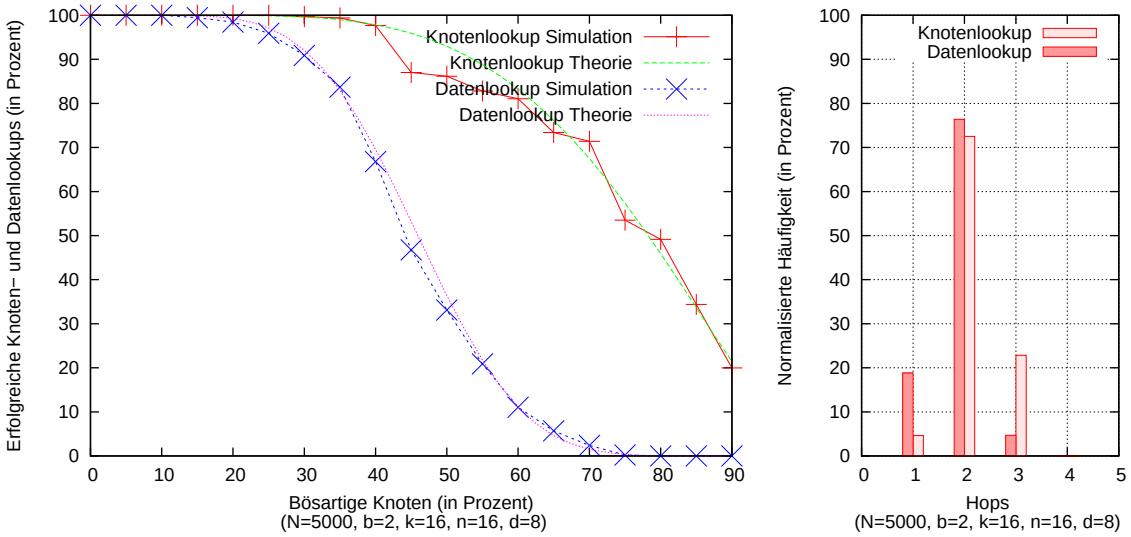


Abbildung A.1: Simulation mit 5000 Knoten, $b = 1$, $n = 16$, $k = 2, 8, 16$ und $d = 1, 4, 8$





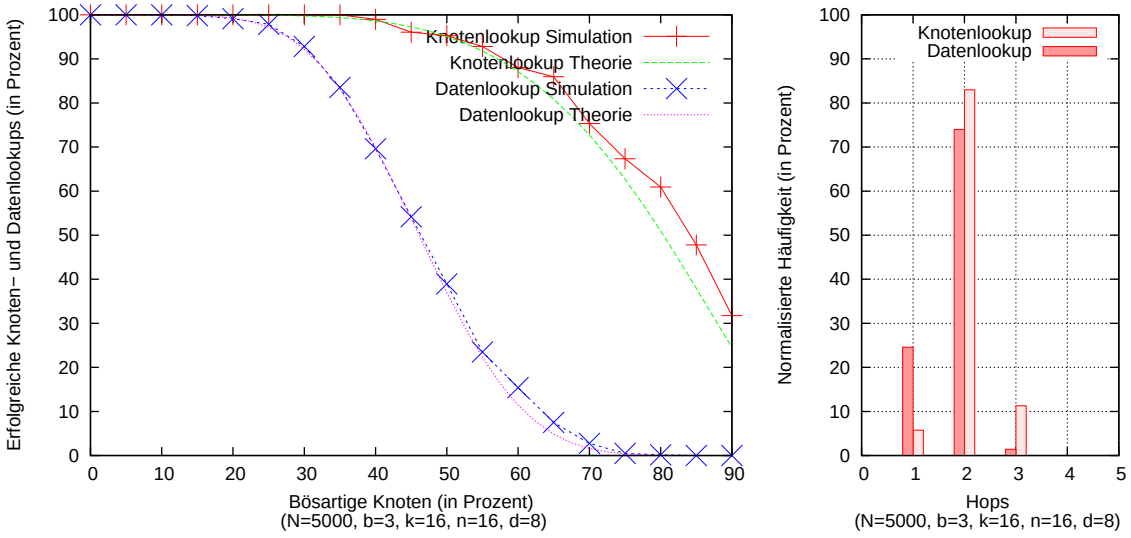
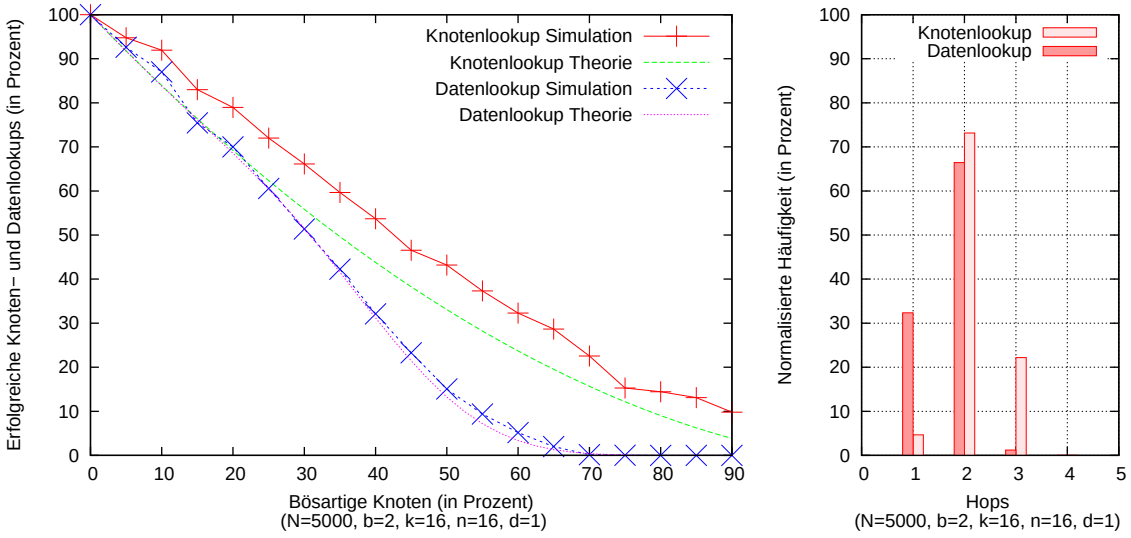
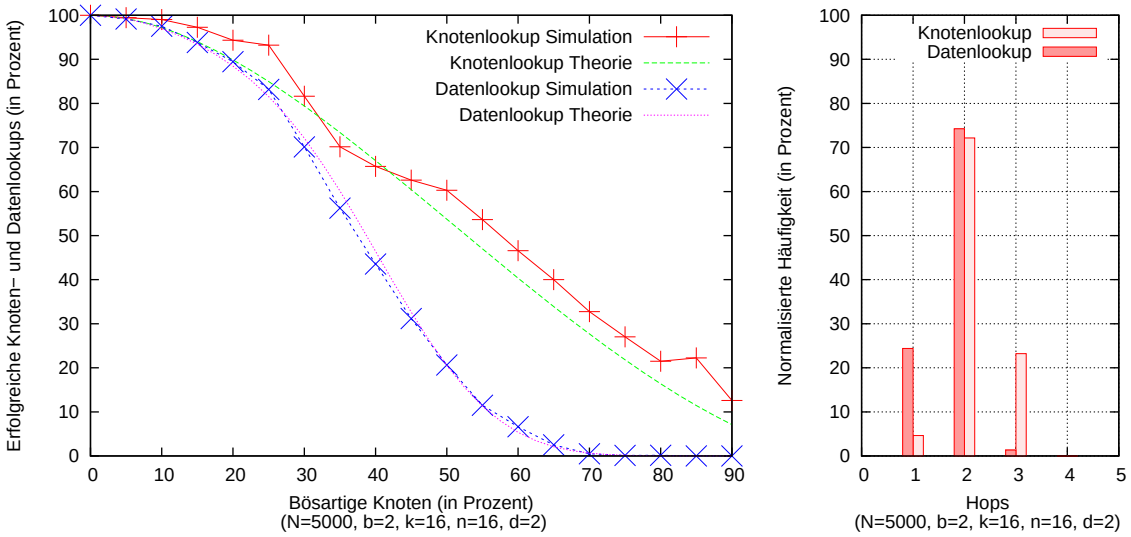


Abbildung A.2: Simulation mit 5000 Knoten, $b = 1, 2, 3$, $n = 16$, $k = 16$ und $d = 8$





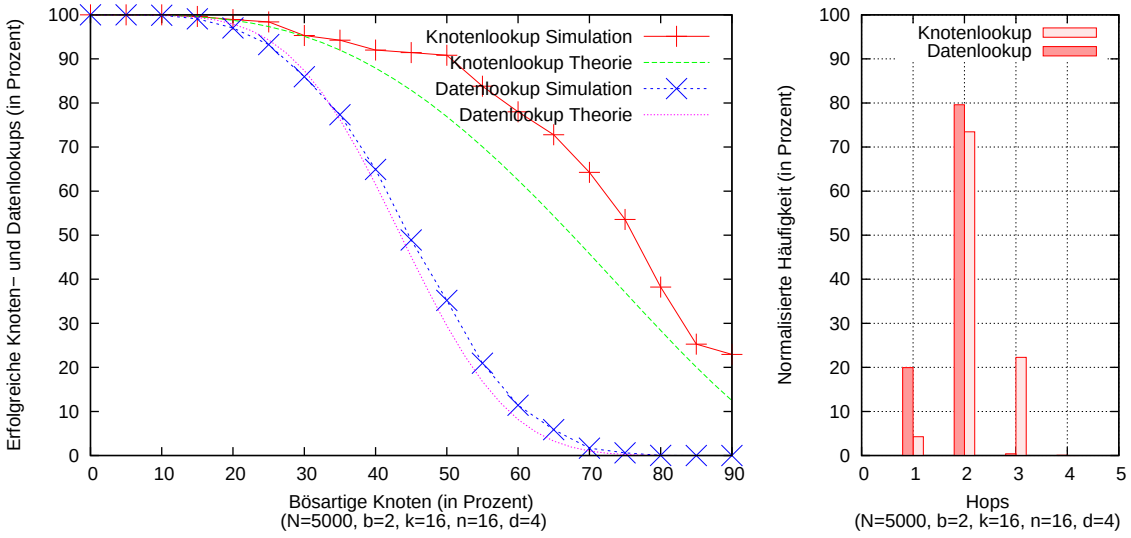


Abbildung A.3: Simulation mit 5000 Knoten, $b = 2$, $n = 16$, $k = 16$ und $d = 1, 2, 4$

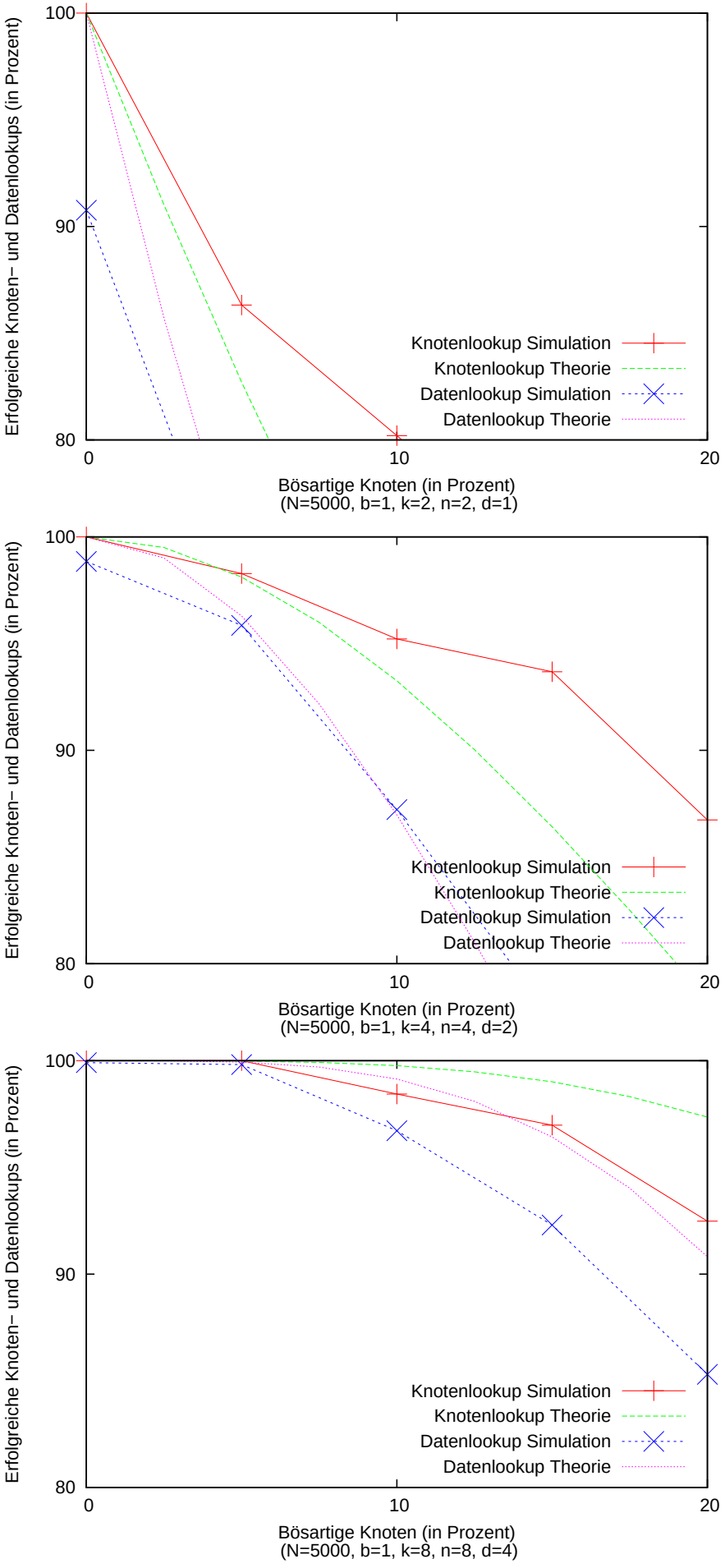
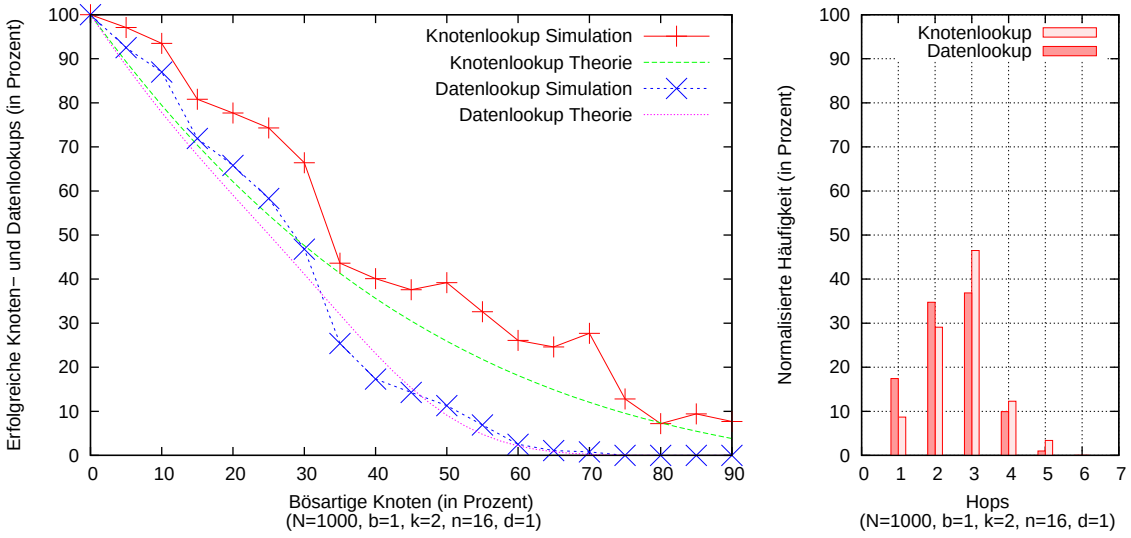
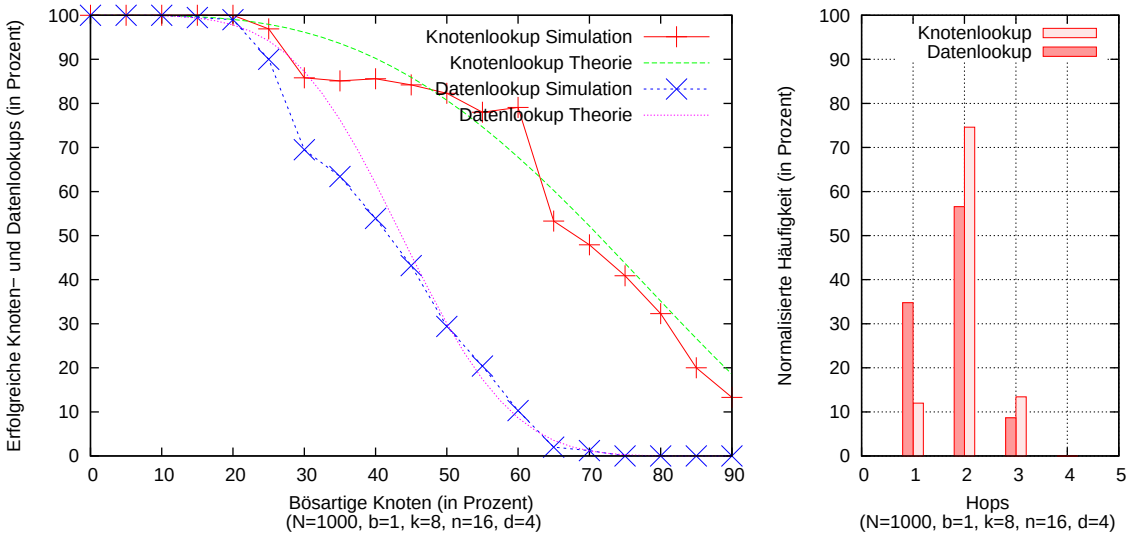


Abbildung A.4: Simulation mit 5000 Knoten, $b = 1$, $n = 2, 4, 8$, $k = 2, 4, 8$ und $d = 1, 2, 4$





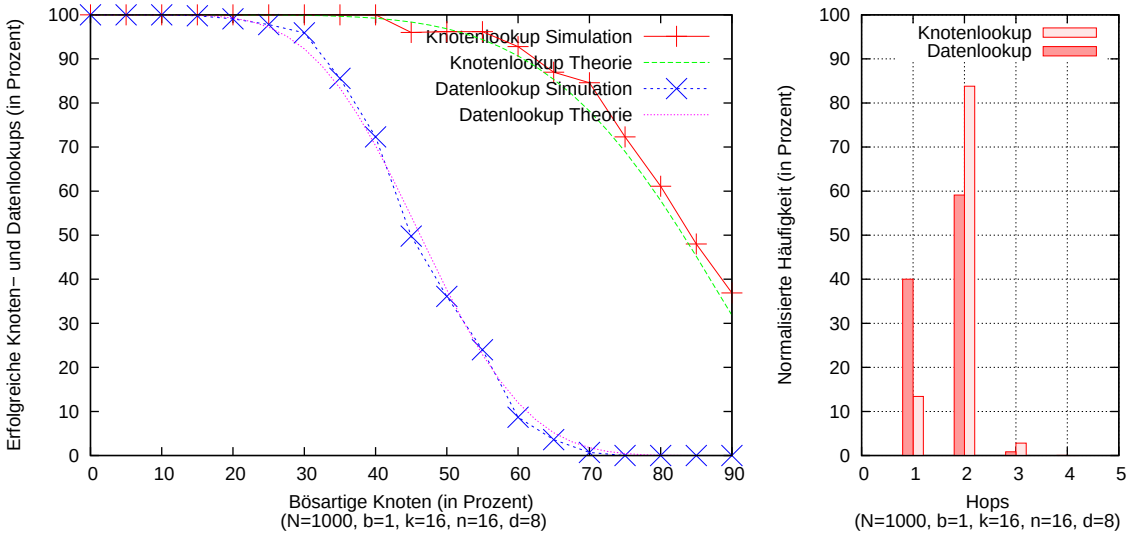
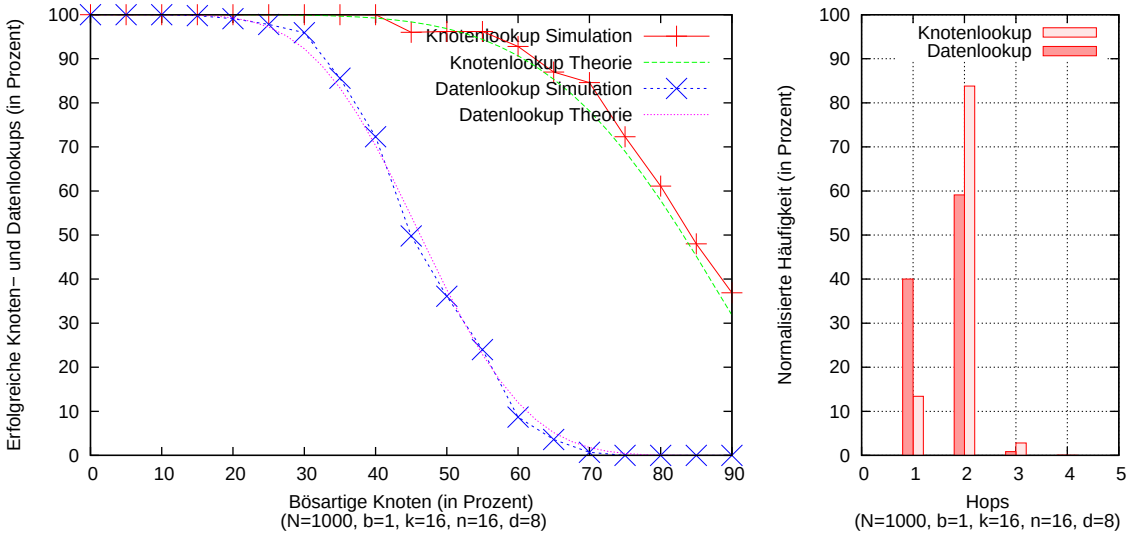
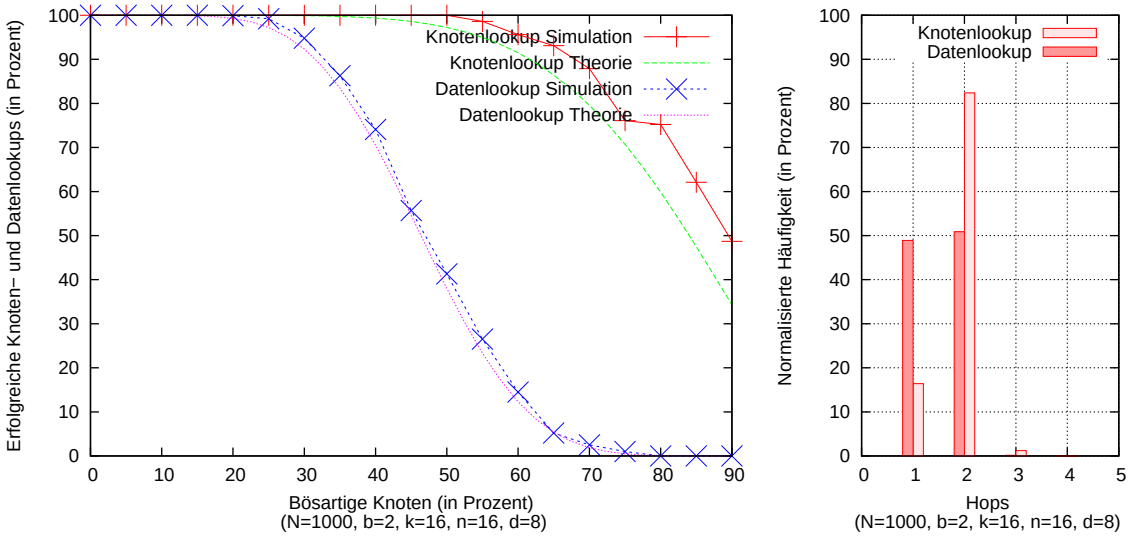


Abbildung A.5: Simulation mit 1000 Knoten, $b = 1$, $n = 16$, $k = 2, 8, 16$ und $d = 1, 4, 8$





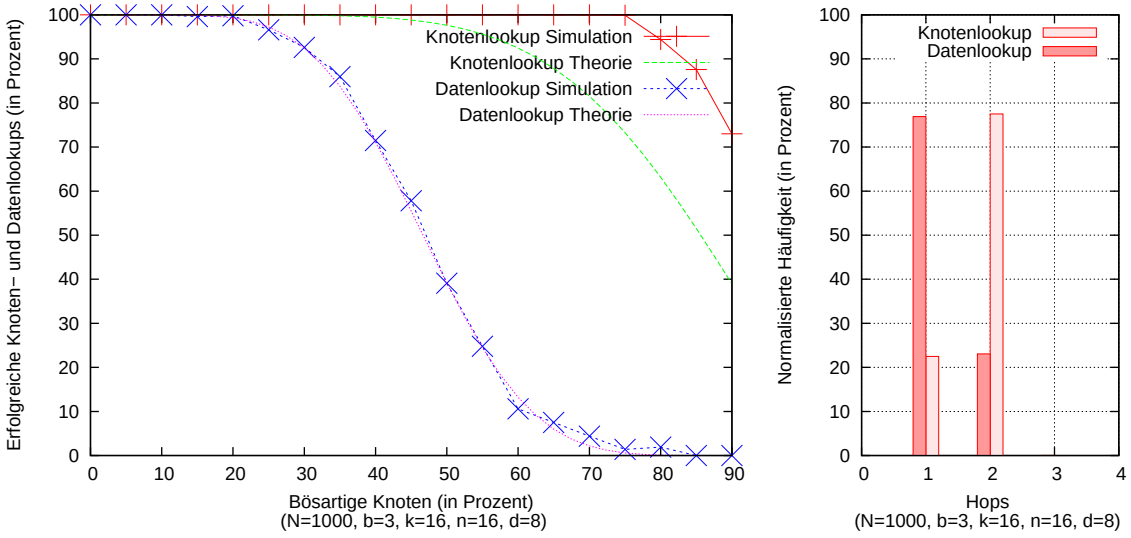


Abbildung A.6: Simulation mit 1000 Knoten, $b = 1, 2, 3$, $n = 16$, $k = 16$ und $d = 8$

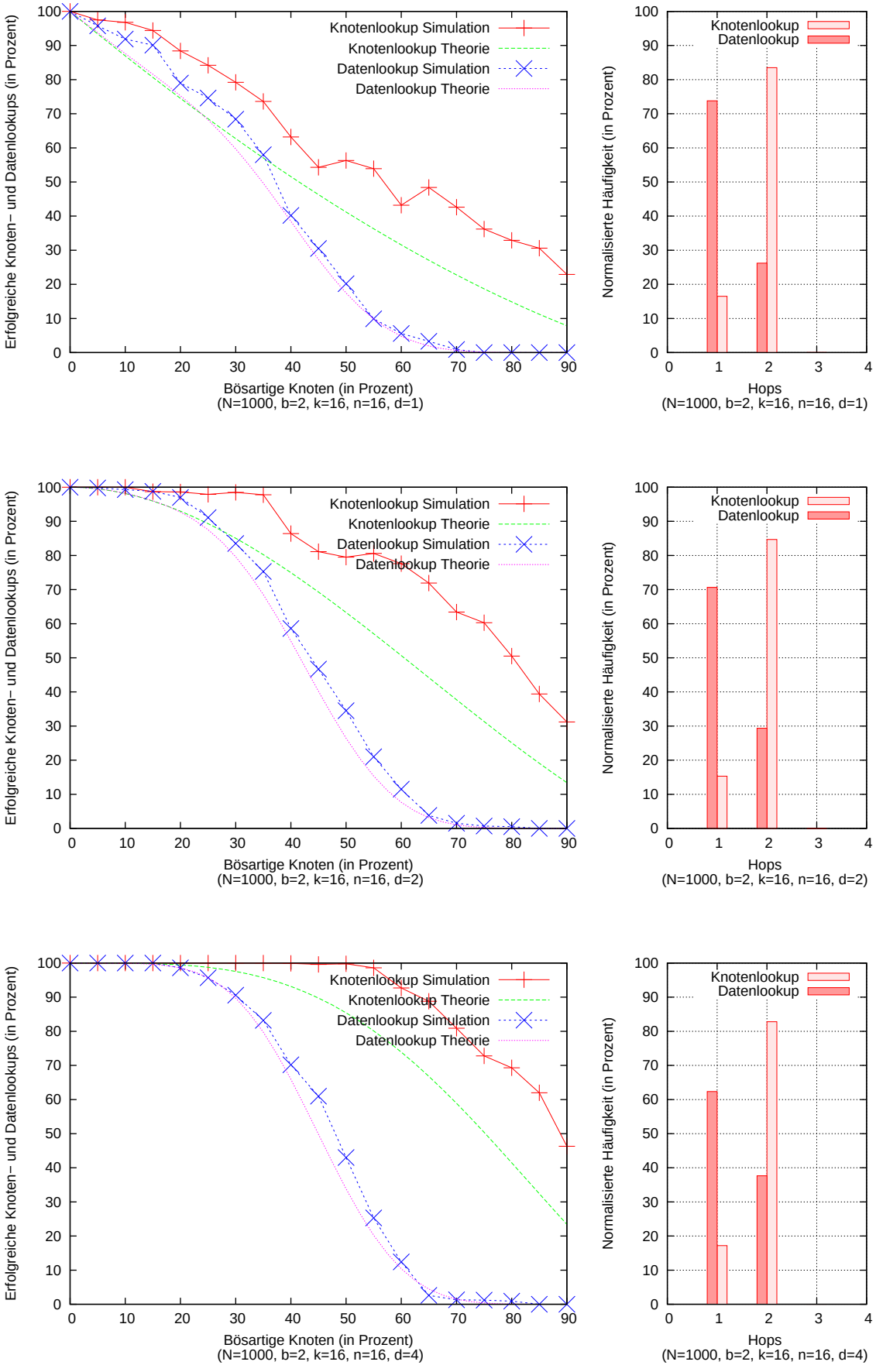


Abbildung A.7: Simulation mit 1000 Knoten, $b = 2$, $n = 16$, $k = 16$ und $d = 1, 2, 4$

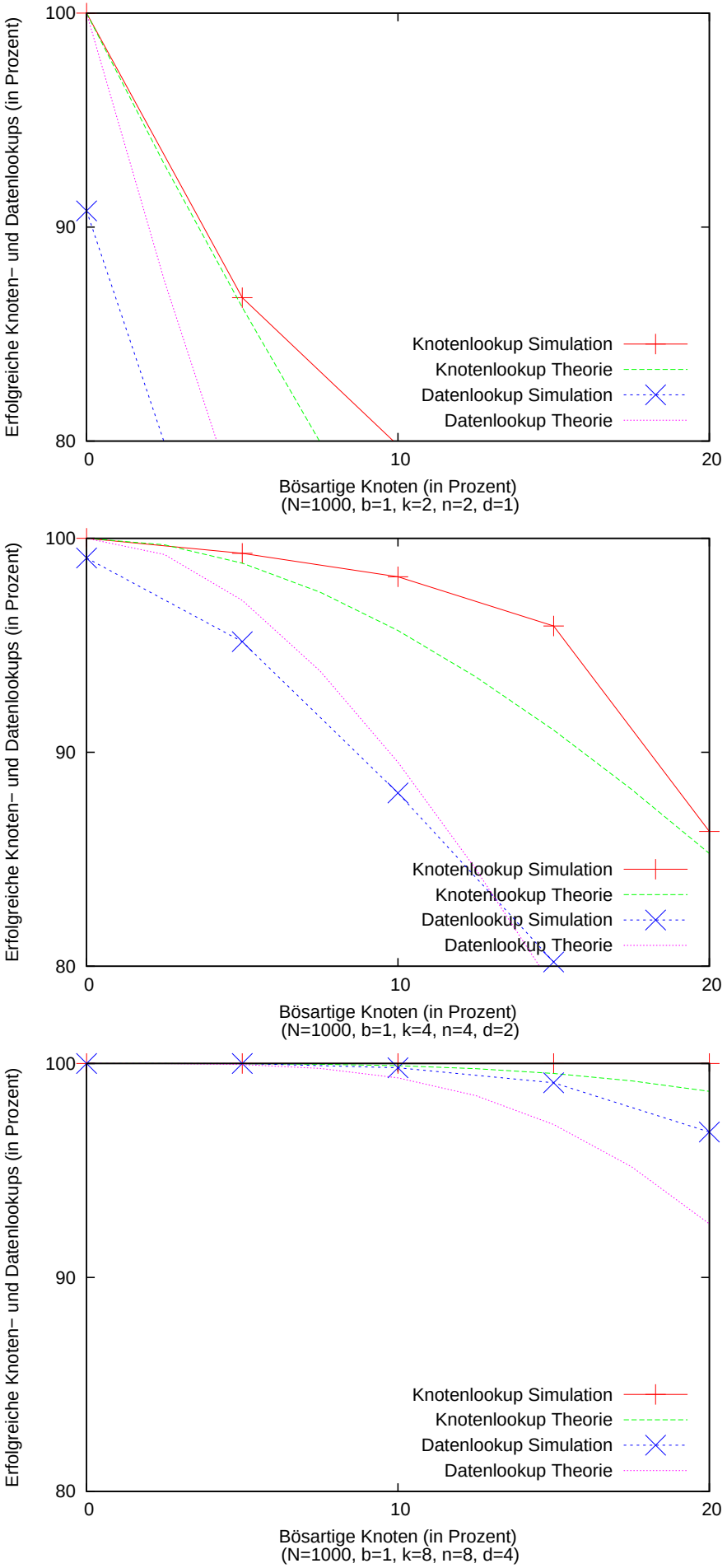


Abbildung A.8: Simulation mit 1000 Knoten, $b = 1$, $n = 2, 4, 8$, $k = 2, 4, 8$ und $d = 1, 2, 4$

A.2 Eigenschaften der XOR-Metrik

Die XOR-Metrik $d_\oplus := (a, b) \to a \oplus b$ ist die Grundlage des Kademlia Routings. Um diese neuartige Metrik zu verstehen, sollen hier die Eigenschaften genauer erläutert werden indem sie mit der Metrik der Norm des absoluten Betrags $d_s := (a, b) \to |a - b|$ verglichen werden. Diese Metrik wird im folgenden als *Standardmetrik* bezeichnet.

Sei $X := \{0, \ldots, 2^n - 1\}$ eine Menge natürlicher Zahlen mit $n \in \mathbb{N}$, dann gilt bezüglich der Standardmetrik:

$$\forall\, x \in X, x + 1 \in X : d_s(x, x + 1) = 1,$$

und damit

$$\sum_{i=0}^{2^n-2} d_s(i, i + 1) = 2^n - 1$$

Dieser Sachverhalt ist klar und bekannt. Bei der XOR-Metrik verhält sich diese Regel aber völlig anders, betrachtet man folgende Beispiele mit $n \in \{2, 3\}$:

x	$d_\oplus(x, x+1)$
00	1
01	3
10	1
11	-
$\sum$	5

x	$d_\oplus(x, x+1)$	x	$d_\oplus(x, x+1)$
000	1	100	1
001	3	101	3
010	1	110	1
011	7	111	-
		$\sum$	17

So lässt sich folgende Gesetzmässigkeit folgern:

$$\sum_{i=0}^{2^n-2} d_\oplus(i, i + 1) = \sum_{i=1}^{n} 2^{n-i} \left(2^i - 1\right) = (n - 1) \cdot 2^n + 1$$

Dieser Sachverhalt wird in folgendem Satz festgehalten:

Satz A.1 *Sei $X := \{0, \ldots, 2^n - 1\}$ eine Menge natürlicher Zahlen mit $n \in \mathbb{N}$ und $d_\oplus : (a, b) \to a \oplus b$ die XOR-Metrik so gilt für den metrischen Raum $(X, d_\oplus)$:*

$$\sum_{i=0}^{2^n-2} d_\oplus(i, i + 1) = (n - 1) \cdot 2^n + 1$$

Weiterhin ist dies der längste Polygonzug in $(X, d_\oplus)$.

Interessant ist auch welcher Abstand durchschnittlich zwischen Zufallszahlen zu erwarten ist. Es wird weiterhin der metrische Raum (X, d_s) und $(X, d_\oplus)$ betrachtet:

Sei $M_d(i) := \{(a, b) \in X \times X | d(a, b) = i\}$ die Menge der Wertepaare (a, b) welche den Abstand i besitzen, der Metrik d, dann ist der durchschnittliche Abstand definiert als:

$$\Delta_d := \frac{\sum_{i=0}^{2^n-1}(i \cdot |M_d(i)|)}{\sum_{i=0}^{2^n-1}(|M_d(i)|)}$$

x	$\frac{1}{2}\lvert M_{d_s}(x)\rvert$	x	$\frac{1}{2}\lvert M_{d_s}(x)\rvert$
0	-	4	4
1	7	5	3
2	6	6	2
3	5	7	1

x	$\frac{1}{2}\lvert M_{d_\oplus}(x)\rvert$	x	$\frac{1}{2}\lvert M_{d_\oplus}(x)\rvert$
0	-	4	4
1	4	5	4
2	4	6	4
3	4	7	4

Tabelle A.1: Anzahl Wertepaare in Abhängigkeit des Abstands für $n = 3$

Aufgrund der Eigenschaften einer Metrik können die Mengen der Null $(a, a) = 0$, sowie die reflexiven Paare $d(a, b) = d(b, a)$ können bei diesen Überlegungen ohne Beschränkung der Allgemeinheit weggelassen werden. Damit läßt sich die durchschnittliche Abstand wie folgt für die Standardnorm berechnen:

$$\Delta_{d_s} := \sum_{i=1}^{2^n-1} 2\,\frac{i\,(2^n - i)}{(2^n - 1)\,2^n} = \frac{1}{3}\,\frac{4^n - 1}{2^n - 1}$$

Verdeutlichen kann man sich diesen Sachverhalt auch an der Tabelle A.1. Der durchschnittliche Abstand für die XOR-Metrik ist dann:

$$\Delta_{d_\oplus} := \sum_{i=1}^{2^n-1} \frac{i}{2^n - 1} = \frac{1}{2}\,\frac{4^n - 2^n}{2^n - 1}$$

Werden $\Delta_{d_\oplus}$ und $\Delta_{d_\oplus}$ im Verhältnis betrachtet, kann festgestellt werden, dass gilt:

$$\frac{\Delta_{d_\oplus}}{\Delta_{d_s}} := \frac{3}{2}\,\frac{4^n - 2^n}{4^n - 1}$$

und somit:

$$\lim_{n\to\infty} \frac{\Delta_{d_\oplus}}{\Delta_{d_s}} = \frac{3}{2}$$

Dieses Verhalten wird im folgenden Satz festgehalten:

Satz A.2 *Sei* $X := \{0, \ldots, 2^n - 1\}$ *eine Menge natürlicher Zahlen mit* $n \in \mathbb{N}$, $d_\oplus : (a, b) \to a \oplus b$ *die XOR-Metrik und* $d_s : (a, b) \to |a - b|$ *so gilt für den metrischen Raum* $(X, d_\oplus)$*:*

$$\lim_{n\to\infty} \frac{\Delta_{d_\oplus}}{\Delta_{d_s}} = \frac{3}{2}$$

sowie für zufällig gewählte $a_i, b_i \in X, i = 1 \ldots 2^n$ *und ausreichend große* n*:*

$$\sum_{i=1}^{2^n} d_\oplus(a_i, b_i) \approx \frac{3}{2} \sum_{i=1}^{2^n} d_s(a_i, b_i)$$

Literatur

[ANL01] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant authentication with client puzzles. *Lecture Notes in Computer Science*, 2133:170+, 2001.

[Bac] A. Back. Hash cash - a denial of service counter-measure. http://www.hashcash.org.

[Bor06] Nikita Borisov. Computational puzzles as sybil defenses. In *P2P '06: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 171–176, Washington, DC, USA, 2006. IEEE Computer Society.

[BWS06] Tanya Y. Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528, New York, NY, USA, 2006. ACM Press.

[CDG+02] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[CDKR02] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*, pages 140–145, New York, NY, USA, 2002. ACM Press.

[CF05] Alice Cheng and Eric Friedman. Sybilproof reputation mechanisms. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 128–132, New York, NY, USA, 2005. ACM Press.

[CS04] Peter Mahlmann Christian Schindelhauer. Algorithmen für peer-to-peer netzwerke, 2004.

[DC05] Stefano Paraboschi Simone Tiraboschi Davide Cerri, Alessandro Ghioni. Id mapping attacks in p2p networks. IEEE GLOBECOM 2005 proceedings, 2005.

[DH06] Jochen Dinger and Hannes Hartenstein. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, pages 756–763, Washington, DC, USA, 2006. IEEE Computer Society.

[Dou02] John R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[FC05] Michal Feldman and John Chuang. Overcoming free-riding behavior in peer-to-peer systems. *SIGecom Exch.*, 5(4):41–50, 2005.

[Fra] Justin Frankel. Das Gnutella Protokoll. `http://rfc-gnutella.sourceforge.net/`.

[Gol95] O. Goldreich. Foundations of cryptography (fragments of a book), 1995.

[GV04] Anh-Tuan Gai and Laurent Viennot. Broose: A practical distributed hashtable based on the de-bruijn topology. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P'04)*, pages 167–164, Washington, DC, USA, 2004. IEEE Computer Society.

[HJ97] M. Handley and V. Jacobson. Sdp: Session description protocol, 1997.

[HR90] Torben Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[HSS97] M. Handley, H. Schulzrinne, and E. Schooler. Sip: Session initiation protocol, 1997.

[JLK] Robert Morris Jinyang Li, Jeremy Stribling and M. Frans Kaashoek. Bandwidth-efficient management of dht routing tables.

[JM99] D. Johnson and A. Menezes. The elliptic curve digital signature algorithm (ecdsa), 1999.

[JM04] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of The 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, pages 102–109, Tokyo, Japan, 2004. IEEE Computer Society.

[JMB04] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, editors, *Engineering Self-Organising Systems*, number 2977 in Lecture Notes in Artificial Intelligence, pages 265–282. Springer, 2004.

[KK03] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[Kle00] Jon Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[Kle01] J. Kleinberg. Small-world phenomena and the dynamics of information, 2001.

[KSGM03] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM Press.

[LKRG03a] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems; routing distances and fault resilience, 2003.

[LKRG03b] Dmitri Loguinov, Anuj Kumar, Vivek Rai, and Sai Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 395–406, New York, NY, USA, 2003. ACM Press.

[LSP82] Lamport, Shostak, and Pease. The byzantine generals problem. In *Advances in Ultra-Dependable Distributed Systems, N. Suri, C. J. Walter, and M. M. Hugue (Eds.), IEEE Computer Society Press.* 1982.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[MN05] James W. Mickens and Brian D. Noble. Predicting node availability in peer-to-peer networks. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 378–379, New York, NY, USA, 2005. ACM Press.

[NR06] Naoum Naoumov and Keith Ross. Exploiting p2p systems for ddos attacks. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 47, New York, NY, USA, 2006. ACM Press.

[P2P] P2P-SIP. P2P-SIP Homepage. http://www.p2psip.org. September 2006.

[PHA+04] Jeffrey Pang, James Hendricks, Aditya Akella, Roberto De Prisco, Bruce Maggs, and Srinivasan Seshan. Availability, usage, and deployment characteristics of the domain name system. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 1–14, New York, NY, USA, 2004. ACM Press.

[Pon93] György Pongor. Omnet: Objective modular network testbed. In *MASCOTS '93: Proceedings of the International Workshop on Modeling, Analysis, and Simulation On Computer and Telecommunication Systems*, pages 323–326. Society for Computer Simulation, 1993.

[PRR97] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, 1997.

[RFH+00] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[RLS02] Rodrigo Rodrigues, Barbara Liskov, and Liuba Shrira. The design of a robust peer-to-peer system. In *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC*, pages 117–124, New York, NY, USA, 2002. ACM Press.

[SCDR04] Atul Singh, Miguel Castro, Peter Druschel, and Antony Rowstron. Defending against eclipse attacks on overlay networks. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop: beyond the PC*, page 21, New York, NY, USA, 2004. ACM Press.

[SCFJ98] Schulzrinne, Casner, Frederick, and Jacobson. RTP: A transport protocol for real-time applications. *Internet-Draft ietf-avt-rtp-new-01.txt (work inprogress)*, 1998.

[See06] Jan Seedorf. Using cryptographically generated sip-uris to protect the integrity of content in p2p-sip, 2006.

[SM02] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 261–269, London, UK, 2002. Springer-Verlag.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.