

Distributed Hash Tables in Peer-to-Peer-Netzwerken

Alexander Wildschütz, Nikolai Wyderka

Göttingen, 20. März 2009

Felix Klein Gymnasium,
Böttinger Straße 17,
37073 Göttingen

Fachlehrer: Martin-Ernst Kraus, Andreas Flemming

Inhaltsverzeichnis

I	Einleitung	1
II	Distributed Hash Tables - die Theorie	2
1	Peer-to-Peer-Netzwerke (NW)	2
1.1	Einleitung	2
1.2	Definition	3
1.3	Vergleich mit Client-Server-Systemen	4
1.4	Geschichte der Peer-to-Peer-Netzwerke	5
1.4.1	Napster	5
1.4.2	Gnutella	6
1.5	Fazit	7
2	Distributed Hash Tables (AW)	7
2.1	Einleitung	7
2.2	Hashfunktionen	8
2.3	Das Netzwerk Chord	9
2.4	Das Netzwerk Kademia	13
3	Sicherheit (AW)	16
3.1	Angriffsmöglichkeiten	16
3.1.1	Incorrect Lookup Routing	16
3.1.2	Denial of Service	17
3.2	Bedeutung für Peer-to-Peer-Netzwerke	18
4	Fazit (AW)	19
III	Distributed Hash Tables - die Anwendung	20
1	Einleitung (NW)	20
2	Modell unseres Netzes XINC (NW)	20
2.1	Aufbau	20
2.2	Unterschiede zu Chord und Kademia	23

2.3	RSA	24
3	Umsetzung von XINC (AW, NW)	25
3.1	Verwendete Programmiersprache	25
3.2	Umsetzung der <i>k</i> -Buckets	26
3.3	Netzwerk-Protokoll	27
IV	Fazit/Ausblick	29
V	Appendix	I
A	Programmcode	I
B	Beispielanwendung	II
Literatur		II

Teil I

Einleitung

Technische Defekte, Internetkriminalität und Naturkatastrophen sind ein Problem jedes informationstechnischen Systems, das Daten verwaltet. Doch wie kann man Daten dezentral und sicher speichern und dennoch schnell darauf zugreifen?

Distributed Hash Tables (DHT), zu Deutsch „Verteilte Hashtabellen“, sind eine sehr junge Methode, dieses Problem mithilfe von sogenannten *Peer-to-Peer-Netzwerken* (P2P-Netzwerken) und Hashfunktionen zu lösen. Im Gegensatz zu reinen P2P-Netzwerken wie Napster oder Gnutella, die zentrale Index-Server und Tiefensuche nutzen, stellt der Ansatz der DHT eine zugleich effiziente, als auch dezentrale, und somit schwer angreifbare Methode dar, um P2P-Netze zu verwalten. Gerade die effiziente Suche ist eines der wichtigsten Probleme dieser Systeme und wird auf unterschiedliche Art gelöst.

Distributed Hash Tables werden immer wichtiger und kommen bereits in vielen verteilten Systemen zum Einsatz, allein schon deshalb lohnt sich ein Blick auf diese Technik. Diese Arbeit wird einen Überblick über die Geschichte dieser Technik, deren Funktionsweise und der praktischen Nutzung geben. Dabei wird geklärt, was Hashfunktionen sind und die Funktionsweisen der bestehenden DHT-Systeme Chord und Kademia erläutert. Schließlich wird ein auf den beschriebenen Techniken basierendes Modell für ein verteiltes Netzwerk zum sicheren Austausch von Textnachrichten entwickelt und in einem Programm umgesetzt, das auf dem Netzwerk Kademia aufbaut. Hier wird zudem ein Schwerpunkt auf die Sicherheit der Kommunikation gelegt. In Zeiten von Bespitzelungsaffären und erweiterter technischer Möglichkeiten unerwünschter Mithörer, ist die Abhörsicherheit der Kommunikation ein wichtiges Thema.

Zunächst soll jedoch eine Einführung in die Grundlagen und Geschichte der Peer-to-Peer-Netzwerke gegeben werden, denn sie stellen die Basis für jedes DHT-Netzwerk dar.

Teil II

Distributed Hash Tables - die Theorie

1 Peer-to-Peer-Netzwerke

1.1 Einleitung

„First we thought the PC was a calculator. Then we found out how to turn numbers into letters with ASCII — and we thought it was a typewriter. Then we discovered graphics, and we thought it was a television. With the World Wide Web, we’ve realized it’s a brochure.” (Douglas Adams)

Hört man das Wort *Peer-to-Peer-Netzwerk*, denkt man häufig an illegale Tauschbörsen, in denen kriminelle Raubkopierer ihre Errungenschaften kostenlos der Allgemeinheit anbieten.

Tatsächlich sind viele Peer-to-Peer-Netzwerke dafür ausgelegt, Dateien auszutauschen. Dabei kommt es auch immer wieder zum Transfer urheberrechtlich geschützter Werke, wie digitalisierter Filme, Musik und Computersoftware. Deshalb werden viele dieser Strukturen als der dunkle, kriminelle Hinterhof des Internets gehandelt - völlig zu Unrecht, denn die Idee hinter diesen Netzwerken ist keinesfalls der Gesetzesbruch, sondern ein schwer angreifbares, robustes und selbstorganisierendes Netzwerk zur Verwaltung von

Daten zu schaffen. In diesen Netzen können viele verschiedene Nutzer aus aller Welt zusammen kommen, um Daten zu tauschen. Gerade diese Zielsetzung ist ein noch in den Kinderschuhen steckendes Teilgebiet der Informatik, welches immer mehr an Bedeutung gewinnt. Seit Jahren nimmt der prozentuale Anteil des Datenverkehrs im Internet, der von P2P-Netzwerken erzeugt wird, zu und ist bereits für

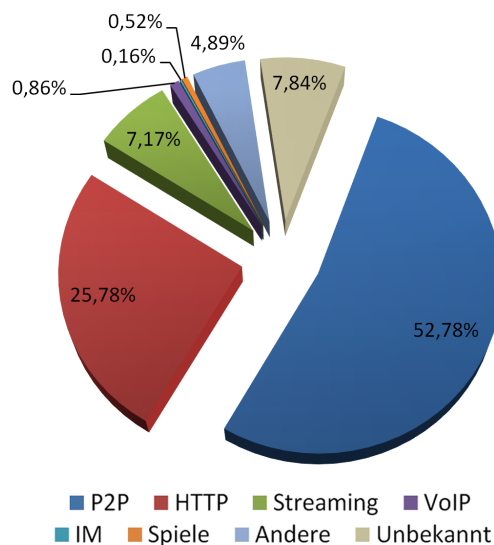


Abbildung 1: Verteilung des Internet-Traffics nach Protokollen in Deutschland 2009

den Großteil des Gesamt-Traffics verantwortlich (Vgl. [Sch09]).

Mit Daten sind nicht zwangsläufig Dateien gemeint, sondern jegliche Art von Informationen. In der Praxis werden Peer-to-Peer-Netzwerke unter anderem auch als Alternative zu Kommunikationsplattformen wie Jabber oder ICQ genutzt. Ein bekanntes Beispiel ist Skype, ein Programm, mit dessen Hilfe man über das Internet telefonieren kann (Vgl. [Mah07], S.248). Aber auch wissenschaftliche Anwendungsbereiche sind denkbar. Hier ist SETI@Home ein nennenswertes Beispiel, ein Netzwerk, in dem Rechenoperationen auf alle Teilnehmer verteilt werden. Mit über dreihunderttausend gleichzeitigen Teilnehmern ist es das weltweit größte Netzwerk für verteiltes Rechnen (Vgl. [Kon09]). Doch was genau ist eigentlich ein Peer-to-Peer-Netzwerk?

1.2 Definition

Peer-to-Peer heißt auf Deutsch soviel wie direkt oder unmittelbar. Peer alleine bedeutet allerdings gleichrangig. Tatsächlich zeichnet sich das Peer-to-Peer-Netzwerk dadurch aus, dass alle Teilnehmer¹ gleichberechtigt sind (Vgl. [Mah07], S.6-7).

Aufgrund der Struktur des Internet-Protokolls (IP) sind aber rein technisch alle Verbindungen ohnehin schon gleichrangig und direkt (Vgl. [Mah07], S.11), und somit streng genommen ein Peer-to-Peer-Netzwerk. Die Hierarchie entsteht erst durch darüberliegende Protokolle (z.B. das Hypertext Transfer Protocol, HTTP), die das Client-Server Prinzip einführen (Vgl. [Fie99]). Häufig nimmt man deshalb in die Definition noch die Suche nach bestimmten Daten oder auch Objekten in diesem Netzwerk auf. Eine mögliche Definition ist:

Definition 1 „*Ein Peer-to-Peer-Netzwerk ist ein auf dem Internet basierendes, verteiltes System zur effizienten und skalierenden Suche nach [...] Objekten ohne zentrale Autorität*“ ([Bej05])

Diese Definition ist für unsere Zwecke gut geeignet, denn sie grenzt den Begriff des Peer-to-Peer-Netzwerks gut von anderen Systemen, wie dem momentan vorherrschenden Client-Server-Systemen ab, bei dem eine zentrale Autorität, der Server, hierarchisch den Clients übergeordnet ist. Man spricht deshalb von hierar-

¹Der Begriff Teilnehmer wird hier, wie auch Client und Knoten, als Bezeichnung für ein elektronisches System verwendet, das Teil eines Rechnernetzes ist.

chischen Systemen. Skalierend bedeutet, dass die Suchzeit bei steigender Teilnehmerzahl nicht linear mit dieser steigt, sondern in einem vertretbaren Rahmen bleibt.

Bei Peer-to-Peer-Netzwerken ergeben sich somit drei zentrale Problemstellungen, die wir wie folgt benennen:

1. Beitritt (Bootstrapping): Da keine zentrale Verwaltungseinheit verfügbar ist, muss ein neuer Knoten durch andere Methoden in das Netzwerk integriert werden.
2. Suche (Routing): Die in dem Netzwerk verwalteten Daten müssen effizient und zuverlässig gefunden werden. Dabei muss auch bei steigender Teilnehmerzahl die Suchzeit in annehmbaren Dimensionen bleiben.
3. Robustheit: Fallen einzelne oder mehrere Teilnehmer aus, darf das restliche Netzwerk davon nicht oder nur in geringem Maße betroffen sein. Gerade weil ein Peer-to-Peer-Netzwerk häufig überwiegend aus privaten Computern besteht, ist die Ausfallrate sehr hoch, beispielsweise wenn der PC ausgeschaltet wird.

1.3 Vergleich mit Client-Server-Systemen

Die Motivation hinter Peer-to-Peer-Netzwerken ergibt sich aus der Problematik der Systeme, bei denen ein Teilnehmer von einem anderen abhängig ist, wie es bei vielen aktuellen Protokollen, wie zum Beispiel dem HTTP-Protokoll, üblich ist. Hier stellt ein zentraler Server mit meist vielen Ressourcen und guter Internetanbindung Dienste bereit. Der HTTP-Webserver bietet unter anderem HTML-Dokumente an, die auf Anfrage den Clients übermittelt werden. Dadurch sind aber alle Clients abhängig vom Server: Fällt dieser aus, ist dessen Dienst nicht mehr verfügbar. Greifen viele Clients gleichzeitig auf den Server zu, hat dieser dementsprechend mehr zu tun, wodurch die Antwortzeiten größer werden. Bei zu vielen Anfragen schafft er es möglicherweise gar nicht mehr zu antworten. Das ist auch für Angreifer interessant. Um einen Dienst anzugreifen, genügt es, dessen zentralen Server ausfindig zu machen, um ihn mit Techniken wie dem Denial of Service (DoS)-Angriff anzugreifen. Der DoS-Angriff ist eine Methode, bei der der Server mit (falschen) Anfragen überflutet wird, wodurch dessen Ressourcen überlastet werden und er nicht mehr in der Lage ist, korrekte Anfragen in annehmbarer Zeit zu bearbeiten (siehe Kapitel II.3:

Sicherheit). Dies geschieht auch immer wieder im World Wide Web. Besonders kritisch ist dies, wenn nicht nur einzelne Server, sondern die für die Funktionalität des Internet wichtigen Nameserver unter Beschuss geraten. Sie sind dafür zuständig, Anfragen mit Hostnamen, wie zum Beispiel *radonlabs.de*, in konkrete IP-Adressen zu übersetzen. Einen solchen Angriff gab es beispielsweise 2008, in dessen Folge viele in Deutschland bereitgestellten Seiten vorübergehend nicht mehr verfügbar waren (Vgl. [Bac08]).

1.4 Geschichte der Peer-to-Peer-Netzwerke

1.4.1 Napster

Funktionsweise: Mit den Problemen der Client-Server-Architektur im Bewusstsein schuf SHAWN „NAPSTER“ FANNING 1999 das erste populäre Peer-to-Peer-Netzwerk namens *Napster*, bei dem die Clients erstmals auch untereinander kommunizierten um Dateien zu tauschen, ohne einen zentralen Server dafür zu benötigen.

Allerdings mussten die Clients auch wissen, wo welche Dateien zu finden waren. Hierfür war wieder eine zentrale Einheit, der Index-Server notwendig, bei der sich Clients melden konnten, wenn sie Dateien anzubieten hatten oder diese suchten. Damit wurde zwar die Hauptlast vom Server genommen, das Problem der Angreifbarkeit wurde dadurch jedoch nicht gelöst. Fällt der zentrale Indexserver aus, können auch keine Dateien mehr bereitgestellt oder gesucht werden. Dies widerspricht ebenfalls der Definition (1). Dennoch erfreute sich dieses Prinzip schnell großer Beliebtheit und wurde zum Warenumschlagplatz für alle Art von Daten, darunter natürlich auch für urheberrechtlich geschütztes Material.

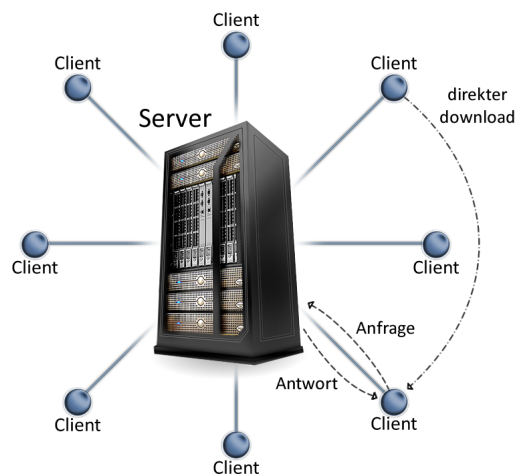


Abbildung 2: Schematische Darstellung des Napster-Netzwerks

Dem Druck der Lizenzinhaber gab Fanning jedoch schon im Jahr 2000 nach und

kommerzialisierte seine Plattform, die heute zwar noch immer existiert, allerdings inzwischen gänzlich auf dem Client-Server-Modell aufbaut (Vgl. [Mah07], S.55-57).

Probleme: Durch den Index-Server ist Napster kein Peer-to-Peer-Netzwerk im heutigen Sinne. Napster löst zwar das Routing- und Bootstrapping-Problem, da zum Beitritt des Netzwerkes und zur Suche darin lediglich der zentrale Server zu kontaktieren ist, das Problem der Robustheit bleibt jedoch.

1.4.2 Gnutella

Funktionsweise: Das Netzwerk Gnutella stammt aus dem Jahr 2000 und sollte die Schwachstellen von Napster ausbessern. Im Gnutella-Netzwerk ist jeder Knoten mit mehreren anderen, gleichberechtigten Knoten verbunden.

Um dem Netzwerk beizutreten, muss der neue Knoten lediglich die Adresse eines anderen aktiven Teilnehmers kennen. An diesen wird eine sogenannte Ping-Nachricht² geschickt, die dieser einfach an alle zu ihm verbundenen Knoten weiter leitet. Die Empfänger leiten sie wiederum weiter, bis das Paket eine maximale Strecke, also Anzahl von Weiterleitungen, zurück gelegt hat. Jeder der erreichten Knoten antwortet darauf mit einer Pong-Nachricht, in der seine Adresse steht. Dieses Paket wird über den gleichen Weg zurück zum Absender geleitet. Dadurch erhält der neue Netzwerkteilnehmer eine Liste von Adressen, zu denen er sich nun verbinden kann. Er ist somit in das Netzwerk integriert.

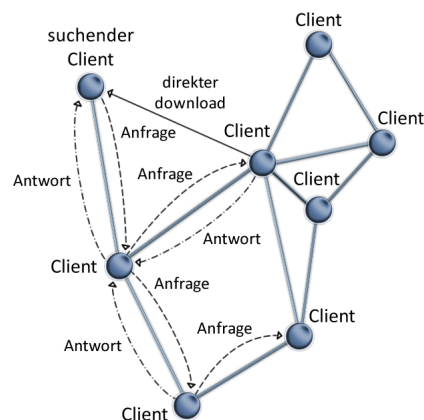


Abbildung 3: Eine Anfrage im Gnutella-Netzwerk

Eine Suche nach Daten läuft ähnlich ab. Eine Anfrage mit bestimmten Suchkriterien wird an alle verbundenen Knoten geleitet, die diese bis zu einer maximalen Suchtiefe weiterleiten. Stellt ein solcher Knoten passende Daten bereit, wird wie bei

²Eine Ping-Nachricht wird benutzt, um festzustellen, ob der Empfänger der Nachricht erreichbar ist. Erhält er die Anfrage, antwortet er mit einer Pong-Nachricht.

der Pong-Nachricht die Antwort auf gleichem Weg zurück geleitet. Der Suchende kann nun mit dem Anbieter der Daten Kontakt aufnehmen (Vgl. [Mah07], S.55-57).

Probleme: Im Gnutella-Netzwerk stellt das Bootstrapping kein Problem dar, da man lediglich einen Teilnehmer kennen muss. Auch die Robustheit ist groß: Da jeder Teilnehmer mehrere Verbindungen aufrecht erhält, hat er noch genug alternative Pfade für den Fall, dass ein Knoten ausfällt. Das Problem in diesem Netzwerk ist jedoch das Routing. Durch die begrenzte Anzahl der Weiterleitungen werden mit großer Wahrscheinlichkeit nur ein Bruchteil der Teilnehmer erreicht. Dadurch werden nur häufig vorkommende Dateien einigermaßen zuverlässig gefunden. Andererseits ist eine Begrenzung notwendig, da sonst die Zeit, die Pakete brauchen um Knoten zu erreichen, die weit entfernt³ liegen, unerträglich groß wird. Zudem könnte das Netzwerk durch die Vielzahl der Suchanfragen überflutet werden. Dennoch erfreut sich das Netzwerk großer Beliebtheit und ist noch heute in unveränderter Form im Einsatz (Vgl. [Mah07], S.60-62).

1.5 Fazit

Während des technischen Einsatzes der ersten Peer-to-Peer-Netzwerke wurde schnell deutlich, dass das Finden von Teilnehmern und Daten in Peer-to-Peer-Netzwerken ein zentrales Problem war und nicht zuverlässig und schnell zugleich gelöst werden konnte. Zwar konnte es durch zentrale Server umgangen werden, wie zum Beispiel in Napster, allerdings widersprach dies der Auffassung von Peer-to-Peer, womit ein dezentrales Netzwerk gemeint ist. Dies führte zur Entwicklung von Distributed Hash Tables, einem Ansatz, um dieses Problem zu lösen.

2 Distributed Hash Tables

2.1 Einleitung

Distributed Hash Tables sind eine sehr junge Methode, um Daten in einem Peer-to-Peer-Netzwerk mithilfe von Hashfunktionen zu finden und zu verwalten (siehe

³Hiermit ist nicht etwa die geographische Distanz gemeint, sie ergibt sich vielmehr aus der Anzahl der zwischen dem Start- und Zielknoten liegenden Knoten und deren Antwortzeiten. So kann es durchaus sein, dass eine Route über Knoten in Australien, Kanada und Saudi Arabien schneller ist, als über den Knoten des (geographischen) Nachbarn.

Kapitel II.2.2: Hashfunktionen). Dabei wird im klassischen Fall jedem Knoten und jedem verwalteten Datum eine eindeutige Zahl, Bitfolge oder Zeichenkette fester Länge zugeordnet (ID). Der Knoten mit der ID n verwaltet nun alle Daten der ID n . Diese ID darf natürlich nicht doppelt vergeben werden. Ein bekanntes Beispiel für eine Hashfunktion, die diese letzte Anforderungen erfüllt, ist der *Secure Hash Algorithmus* (SHA). Dieser arbeitet mit einer Zielmenge von bis zu 512 Bit, was eine Anzahl von 2^{512} Möglichkeiten bedeutet.

Ein Problem, das sich daraus ergibt, ist die Anzahl der Knoten. In der Regel wird es nicht für jedes mögliche Datum der ID n einen zugehörigen Knoten der gleichen ID geben. Deshalb muss die gesamte Zielmenge unter den bestehenden Knoten aufgeteilt werden, so dass für jede ID n mindestens ein Knoten zuständig ist. Für eine skalierende Suche reicht das alleine jedoch nicht. Auch das entsprechende Netzwerk muss so aufgebaut sein, dass zuständige Knoten möglichst schnell gefunden werden. In der Regel wird für ein von einem Teilnehmer bereitgestelltes Datum mittels der Hashfunktion eine ID berechnet. Anschließend wird dem für diese ID zuständigen Teilnehmer mitgeteilt, dass dieses Datum angeboten wird und wie man es erreichen kann. Sucht nun ein anderer Teilnehmer nach diesem Datum, wird er bei der Suche den für dieses Datum zuständigen Knoten fragen. Dieser wird ihm die Adresse des bereitstellenden Knotens mitteilen, so dass eine direkte Verbindung zwischen dem Suchenden und dem Anbieter aufgebaut werden kann. Ein zentrales Problem ist hier die Suche nach dem für eine ID zuständigen Knoten, welches in der Praxis unterschiedlich gelöst wird (Vgl. [Rat02]).

2.2 Hashfunktionen

Hashfunktionen spielen eine zentrale Rolle in DHT, weshalb es sich lohnt, einen kurzen Blick darauf zu werfen. Hashfunktionen berechnen aus gegebenen Eingabedaten einen sogenannten Hash aus einer Zielmenge fester Größe. Diesen Vorgang nennt man auch hashen. Ein einfaches Beispiel für eine Hashfunktion ist der Modulo-Operator. Er gibt den ganzzahligen Divisionsrest zweier Zahlen an. Bei der Eingabe 42 würde die Funktion $42 \bmod 17 = 8$ ergeben, eine Eingabe von 314159265358979 ergibt 5. Die Zielmenge ist hier $\{0 \dots 16\}$, da es bei der Division durch 17 keinen Rest größer 16 geben kann. In der Praxis wird Modulo jedoch nicht als Hashfunktion verwendet, da es noch weitere Anforderungen gibt: Kleine

Änderungen an den Eingangsdaten sollen einen möglichst komplett anderen Hash ergeben, und die verwendete Funktion sollte schwer umkehrbar sein, damit von dem Hash nicht auf die Eingabedaten geschlossen werden kann. Eine solche Funktion ist beispielsweise SHA. Erhöht man jedoch bei Modulo den Eingabewert um eins, so wird auch das Resultat um eins größer sein. Außerdem ist der Modulo-Operator leicht umkehrbar, und es lassen sich in kurzer Zeit mit $n = k * 17 + 8$ für beliebige ganze Zahlen k beliebig viele Zahlen n berechnen, die das gleiche Ergebnis liefern würden.

In der Praxis werden Hashfunktionen unter anderem genutzt, um fehlgeschlagene Dateiübertragungen festzustellen. Dazu bietet die Quelle den mittels der MD5-Hashfunktion erstellten Hash der Datei an, der aus 128 Bit besteht. Nach der Übertragung der Datei erstellt das Ziel der Dateiübertragung seinerseits mit dem gleichen Algorithmus einen Hash der Datei und vergleicht diesen mit dem der Quelle. Wurde auch nur ein Bit fehlerhaft übertragen, sieht der Hash mit großer Wahrscheinlichkeit ganz anders aus. Deshalb werden Hashs auch Prüfsummen genannt.

Auch in der Kryptografie finden Hashfunktionen Einsatz. Sie stellen jedoch selbst keinerlei Verschlüsselung dar, obwohl häufig Gegenteiliges behauptet wird. In vielen Systemen, bei denen sich der Nutzer authentifizieren muss, wird vor der Übertragung an den Server sein Passwort mittels einer Hashfunktion gehasht und nur dieser Hash übertragen. Der Server kennt nur den Hash des Passworts, nicht das Passwort selbst und kann diesen dann mit dem übertragenen Hash vergleichen. So kann Datenmissbrauch seitens des Anbieters erschwert werden.

Ein Problem bei Hashfunktionen ist die Kollision. Durch die begrenzte Größe der Zielmenge des Hashs können verschiedene Eingaben zur Berechnung des gleichen Hashs führen. Diese Gefahr gilt es möglichst gering zu halten, indem die Länge des Hashs ausreichend groß gewählt wird und die Hashfunktion selbst die erstellten Hashs gleichmäßig über den Raum aller möglichen Hashs verteilt (Vgl. [Mah07], S.63-66).

2.3 Das Netzwerk Chord

Chord ist ein klassisches Beispiel für DHT nutzende Netzwerke. Es wurde 2001 von ION STOICA et al. vorgestellt und bietet die Basis für viele weitere Netze. Chord selbst stellt keine Funktionen zum Austausch von Daten bereit, sondern lediglich

zur Suche nach Knoten. Dies ist aber kein Nachteil, denn jeder Teilnehmer kann mithilfe dieser Suche dem für seine Daten zuständigen Knoten seine Kontaktadresse mitteilen. Sucht ein anderer Knoten nach genau diesen Daten, fragt er den für dieses Datum zuständigen Knoten und erhält von diesem die Adresse des tatsächlichen Anbieters der Daten, wodurch eine direkte Kommunikation möglich wird. Chord bietet zudem eine gute Skalierung (Vgl. [Mah07], S.81).

Das Netzwerk ist ringförmig aufgebaut. Jedem Knoten wird nun mithilfe einer Hashfunktion eine ID aus dem Adressraum 2^k zugeordnet, wobei k der Anzahl der Bits der von der Hashfunktion vergebenen IDs entspricht. Der Ring wird in 2^k Abschnitte, von 0 bis $2^k - 1$, unterteilt und jeder Knoten entsprechend seiner ID auf diesem Ring angeordnet. Jeder Knoten verwaltet nun alle Daten, deren ID kleiner oder gleich seiner eigenen und größer der des auf dem Ring vor ihm liegenden Knoten sind. Außerdem kennt jeder Knoten c

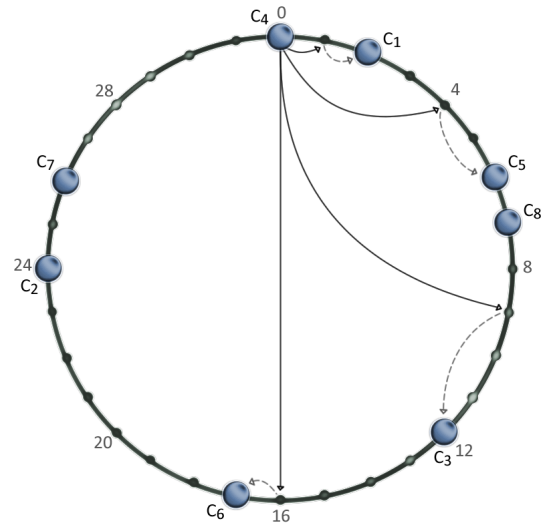


Abbildung 4: Das Chord-Netzwerk und die Finger des Knotens c_4

den ihm auf dem Ring folgenden c' (auch engl. successor genannt). Allein dadurch würde eine Suche bereits funktionieren: Sucht der Knoten c mit der ID $ID(c)$ das Datum respektive den für dieses Datum zuständigen Teilnehmer, prüft er zunächst, ob er selbst dafür zuständig ist. Ist das nicht der Fall, leitet er die Anfrage an seinen Nachfolger c' weiter, der wiederum prüft ob er zuständig ist um gegebenenfalls, sollte er ein entsprechendes Datum verwalten, die Kontaktdaten des bereitstellenden Knotens an den Suchenden zurückzugeben. Das setzt allerdings voraus, dass jeder, der ein solches Datum anbietet, seine Kontaktdaten über ähnliche Suchanfragen dem für die ID seines Datums zuständigen Knoten mitteilt.

Der Nachteil dieser einfachen Suchmethode ist die bereits angesprochene Skalierbarkeit. Im schlechtesten Fall (worst case) muss die Anfrage bei n Teilnehmern $n - 1$ mal weitergeleitet werden, um den passenden Knoten zu erreichen. Man spricht hier von der Komplexitätsklasse $\mathcal{O}(n)$. Sie beschreibt das asymptotische Laufzeitverhalten, also das Verhalten der Laufzeit bei steigender Teilnehmerzahl.

$\mathcal{O}(n)$ bedeutet, dass sich die Laufzeit an die Funktion $f(n) = a * n + b$ annähert. Betrachtet wird also nicht die tatsächliche Laufzeit, sondern die Art der Änderung bei steigendem n . Vereinfacht kann man sagen, dass bei doppelter Teilnehmerzahl sich auch die Anzahl der nötigen Suchanfragen verdoppelt.

Das ist bei großen Netzwerken, die aus vielen Teilnehmern bestehen, nicht praktikabel. Deshalb verwendet Chord noch eine weitere Technik, die sogenannte Finger-Tabelle. In ihr werden Informationen, sogenannte Finger, über bestimmte nachfolgende Knoten gespeichert. Dabei wird quadratisch vorgegangen, das heißt, es werden Verweise auf Knoten gespeichert, die für die Adresse $ID(c) + 1, ID(c) + 2, ID(c) + 4, \text{ usw.}$ zuständig sind. Allgemein gesagt: $ID(c) + 2^i$ für alle natürlichen

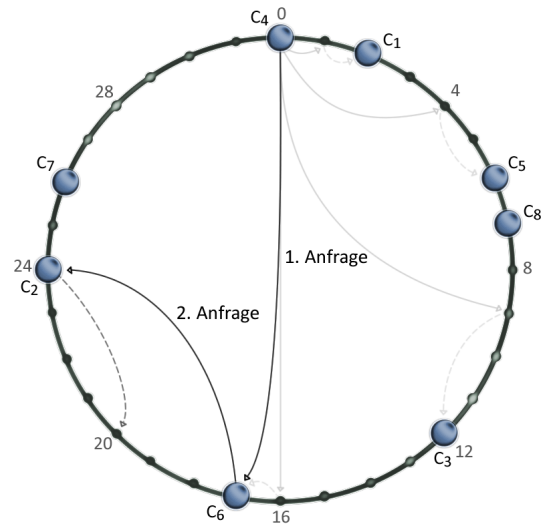


Abbildung 5: Schematische Suche nach ID 20 durch Knoten c_4

Zahlen $0 \leq i < k$. Somit erhält jeder Knoten eine Liste von k Fingern. Dabei spielt es keine Rolle, ob während dieses Vorgangs der Adressbereich von 2^k überschritten wird. Durch die Ringform des Netzwerks werden die Adressen $\geq 2^k$ durch den Modulo-Operator zurück in den validen Bereich geschoben. Die so korrigierte Formel lautet also $ID(c) + 2^i \bmod 2^k$. Die Abbildung 4 zeigt ein Netzwerk mit $k = 5$ und den Verweisen, die dem Knoten c_4 mit der ID 0 zur Verfügung stehen. Liegt an der Stelle $ID(c) + 2^i \bmod 2^k$ kein Knoten, wird stattdessen der für diese Adresse zuständige Knoten in die Liste eingetragen. Dadurch kann es vorkommen, dass der gleiche Knoten mehrmals in der Liste vorkommt.

Um ein Datum der ID n zu finden, schickt der suchende Knoten eine Anfrage an den nächsten Knoten den er kennt, der möglichst dicht an n , aber noch davor liegt. Dieser ist näher an der gesuchten ID und kennt somit auch mehr Knoten in der unmittelbaren Nachbarschaft. Auch er fragt nun den nächsten vor n liegenden Knoten den er kennt. Das wird solange wiederholt, bis es keinen Nachfolger mehr gibt, der vor n liegt. Der direkte Nachfolger dieses Knotens ist der für das Datum n zuständige Knoten. Der Vorteil dieses Aufbaus ist, dass hierdurch Suchanfragen wesentlich effizienter werden, da mit einer einzelnen Suchanfrage mindestens die

Hälfte des Chordrings umrundet werden kann. Mit jeder Suchanfrage halbiert sich im Mittel der Abstand zum Zielknoten. Dadurch braucht man in einem Chordring mit n Knoten im schlechtesten Fall $\mathcal{O}(\log_2(n))$ Anfragen. Das ist eine deutliche Steigerung zu der anderen Suchtechnik. In einem Netzwerk mit 65536 Knoten (2^{16}) würden mit der nicht skalierenden Suche im schlimmsten Fall 65535 Nachrichten verschickt werden müssen. Mit der skalierenden Suche sind es lediglich etwa $\log_2(65535) \approx 16$ Anfragen, also ein deutlicher Unterschied. Abbildung 5 zeigt ein mögliches Szenario einer solchen Suchanfrage. Der Knoten der c_4 mit der ID 0 sucht nach der ID 20 und benötigt dabei lediglich drei Suchanfragen (Vgl. [Mah07], S. 85-91).

Der Nachteil dieser Methode ist jedoch, dass der Aufwand, das Netzwerk zu organisieren und stabil zu halten, stark steigt. Beim Eintritt eines Knotens in das Netzwerk sind außerdem zusätzliche Schritte nötig, um die Fingertabelle aufzubauen. In der Praxis nutzt Chord alleine drei Funktionen, die jedem Knoten zur Verfügung stehen, um diese Probleme zu lösen: Die *fix_fingers*-Funktion startet eine Suche für jeden Eintrag seiner Fingertabelle, so dass er die zuständigen Knoten für die Adressen $ID(c) + 2^i \bmod 2^k$ bekommt und diese in die Tabelle eintragen kann. Diese Funktion wird regelmäßig aufgerufen, so dass auch bei Änderungen im Netzwerk durch ausfallende und neu hinzukommende Knoten dafür gesorgt wird, dass die Tabelle aktuell bleibt. In der Praxis wird aber nicht jedesmal die komplette Tabelle neu aufgebaut, da das eine große Netzwerklast bedeutet, sondern häufig nur jeweils ein zufälliger Eintrag neu gesetzt. Die Funktion *stabilize* fragt den Nachfolgeknoten nach seinem Vorgänger. Dieser sollte der fragende Knoten selbst sein. Ist das nicht der Fall, wenn etwa ein neuer Knoten dazukommt, dessen ID zwischen der des Aufrufers und dessen Nachfolgers liegt, wird der neue Knoten als neuer Nachfolger genommen. Für diesen Fall wird außerdem die Funktion *notify* benötigt, die dann den neuen Nachfolger benachrichtigt, dass er einen neuen Vorgänger hat. Fällt ein Knoten aus, ist das nicht weiter schlimm, denn die Korrektheit der eigenen Fingertabelle ist nur für die Skalierbarkeit der Suche wichtig. Ist die Fingertabelle fehlerhaft, etwa weil die darin enthaltenen Knoten nicht mehr im Netzwerk sind, kann zur Not noch auf die nicht skalierende Suche zurückgegriffen werden, solange zumindest noch der direkte Nachfolger antwortet. Ist seine Fingertabelle noch korrekt, kann die Suche von dort aus wieder skaliert werden.

Möchte ein neuer Knoten c dem Netzwerk beitreten, muss er auch bei Chord

mindestens einen aktiven Teilnehmer des Chord-Netzes kennen, zudem wird mithilfe der Hashfunktion seine eigene ID berechnet. Über den bekannten Knoten wird nun eine Suchanfrage nach der eigenen ID durchgeführt, die den direkten Nachfolger von c liefert. Diesem wird mittels *notify* mitgeteilt, dass c dessen neuer direkter Vorgänger ist. Um seinen eigenen Vorgänger zu erfahren, muss er warten, bis dieser wiederum *notify* aufruft und erfährt, dass c existiert. Durch diese Techniken ist sowohl die Anforderung nach Ausfallsicherheit, dem Beitritt sowie der skalierbaren Suche erfüllt. Wir haben hier also erstmals ein wirkliches Peer-to-Peer-Netzwerk, das auch einer großen Anzahl von Teilnehmern die Möglichkeit bietet, zuverlässig Knoten zu finden und dabei komplett dezentral ist (Vgl. [Mah07], S. 81-91).

Ein Problem von Chord ist der verhältnismäßig große Aufwand, der nötig ist, um das Netzwerk stabil zu halten. Außerdem ist bereits bei kleineren Änderungen im Netzwerk die Funktion des Systems nicht mehr gegeben. Fallen beispielsweise mehrere Knoten, die hintereinander im Chordring liegen, gleichzeitig aus, kann der Vorgänger nicht mehr auf seinen Nachfolger zugreifen und die von diesem Knoten verwalteten Daten sind für längere Zeit nicht mehr auffindbar.

2.4 Das Netzwerk Kademlia

Kademlia wurde 2002 von PETAR MAYMOUNKOV und DAVID MAZIÈRES vorgestellt. Es wurde mit dem Ziel entwickelt, die Vorteile bestehender Systeme wie Chord und Gnutella aufzugreifen und dessen Schwachstellen zu verbessern.

Kademlia verwendet dabei, wie auch Chord, DHT, um sowohl Nutzdaten als auch Teilnehmer zu verwalten. Jedem Teilnehmer wird dabei eine eindeutige ID aus 160 Bits zugeordnet. Anders als bei Chord wird jedoch keine Fingertabelle verwendet, sondern sogenannte k -Buckets (engl. für Eimer). Jeder Teilnehmer hat 160 solcher Buckets, die jeweils k Verweise auf andere Knoten fassen. In der Praxis beträgt k 20. Außerdem werden, anders als bei Chord, nicht regelmäßig die Verweise aktualisiert. Bei Kademlia geschieht dies passiv als Nebeneffekt anderer Nachrichten, die verschickt werden. Jedesmal, wenn eine Nachricht eintrifft oder als Resultat einer eigenen Anfrage Informationen über Knoten erhalten werden, werden diese Informationen in den entsprechenden Bucket aus den k -Buckets hinzugefügt. Ist der Knoten bereits bekannt, wird er an das Ende des entsprechenden Buckets geschoben. Dadurch sind die Listen so sortiert, dass die zuletzt gesehenen Knoten am

Ende stehen. Ist der Knoten noch nicht in der Liste, der entsprechende Bucket aber voll, wird der erste Knoten, also der, der am längsten nicht gesehen wurde, angepingt. Antwortet er, wird er an das Ende geschoben und der neue Knoten verworfen. Antwortet er nicht, wird er aus den Buckets entfernt und der neue Knoten an das Ende gesetzt. Dadurch werden Knoten, die das Netzwerk verlassen haben automatisch nach einiger Zeit aus den k -Buckets entfernt indem sie durch neue ersetzt werden.

Um herauszufinden, in welchen Bucket ein Knoten gehört, wird zunächst der Abstand zu diesem berechnet, indem die eigene ID über die XOR-Funktion⁴ bitweise mit der ID des Anderen verknüpft und als Ganzzahl interpretiert wird. Es gilt: $Abstand(ID_1, ID_2) = ID_1 \oplus ID_2$. Der Knoten wird nun in Abhängigkeit dieses Abstandes in einem der 160 Buckets gespeichert. Dabei gilt: Der i -te Bucket ist für alle Knoten zuständig, deren Abstand zu dem eigenen Knoten im Bereich 2^{i-1} bis $2^i - 1$ liegt. Der erste Bucket ist also für den Abstand 1 zuständig, der zweite für die Abstände 2 und 3, der dritte für 4 bis 7 und so weiter (Vgl. [May02], S.56-57).

Um das zu veranschaulichen gehen wir zunächst davon aus, dass die Länge der IDs nicht 160, sondern lediglich 4 Bit beträgt. Damit reduziert sich auch die Anzahl der Buckets auf 4. Lernt ein Knoten der ID $(5)_{10} = (0101)_2$ den Knoten der ID $(9)_{10} = (1001)_2$ kennen, so ist der berechnete Abstand zwischen diesen Knoten $(0101)_2 \oplus (1001)_2 = (1100)_2 = (12)_{10}$. Mithilfe des Logarithmus finden wir nun den zuständigen Bucket. Aus $\log_2(12) \approx 3.58$ folgt: $i = 4$. Der Knoten würde also im vierten Bucket gespeichert.

Es fällt auf, dass besonders viel Platz für Knoten ist, die nahe am Eigenen liegen und wenig Platz für weit entfernte. Das bedeutet, dass ein Knoten im Laufe der Zeit viele Informationen über die unmittelbare Nachbarschaft sammelt, jedoch wenige Kenntnisse über weit entfernte hat. Diese Eigenschaft wird ausgenutzt, um effizient nach anderen IDs suchen zu können. Dazu wird zunächst aus den eigenen Buckets eine Liste der k , im Normalfall also 20, der gesuchten ID nächsten Knoten ermittelt. An einer festen Anzahl α von Knoten wird dazu die *find_node*-Funktion aufgerufen, die dem Aufrufenden die k Kontaktdaten zu Knoten zurückliefert, die der aufgerufene Knoten kennt, die der gesuchten ID am nächsten sind. In der Praxis wird für α meistens 3 genutzt. Aus den so gesammelten Daten werden nun die k

⁴Das binäre XOR (exklusives Oder) vergleicht die gegebenen Operanden bitweise. Ein Bit im Ergebniswort ist genau dann 1, wenn die entsprechenden Bits der Operanden unterschiedlich sind. Der Operator für XOR ist \oplus . Bsp.: $(0111)_2 \oplus (0101)_2 = (0010)_2$

besten, also dem Ziel am nächsten liegenden Knoten ausgewählt und der Vorgang wiederholt, bis sich an der Liste nichts mehr ändert oder bereits alle Knoten in der Liste mittels *find_node* befragt wurden. Bei jedem Schritt tastet man sich so näher an das Ziel heran. Durch die logarithmische Struktur der k -Buckets erhält man außerdem von den dem Ziel näheren Teilnehmern mit hoher Wahrscheinlichkeit eine Liste mit Teilnehmern, die noch näher an dem Ziel liegen, da für diese mehr Platz ist.

Wie auch bei Chord ergibt sich so eine skalierende Suche mit der Komplexitätsklasse $\mathcal{O}(\log_2(n))$. Als Ergebnis erhält man nun eine Liste der k der Ziel-ID am nächsten liegenden Knoten, die nun wahlweise nach gesuchten Daten befragt (*find_value*-Funktion) oder angewiesen werden können, Informationen über eigene Daten zu speichern (*store*-Funktion). Dazu wird mittels des verwendeten SHA-1-Hash-Algorithmus eine 160-Bit ID für die angebotenen Daten erstellt und die oben beschriebene Suche nach dieser ID durchgeführt. Den ermittelten k Knoten, die der ID der Daten am nächsten liegen, wird die ID und die eigenen Kontaktmöglichkeiten gegeben, so dass diese den Knoten übermittelt werden können, die nach genau dieser ID suchen. Diese landen ebenfalls bei den k diese ID verwaltenden Knoten, es reicht aber auch nur einen der k Knoten zu finden. Für den Fall, dass zwischenzeitlich Knoten hinzukommen, die noch näher an der Ziel-ID liegen, veröffentlicht außerdem jeder Knoten alle 60 Minuten alle von ihm verwalteten Daten erneut im Netzwerk (Vgl. [May02], S.57,60). Praxistests haben ergeben, dass sich ein größeres α positiv auf die durchschnittliche Suchzeit auswirkt. Wie sehr sich die Änderung des Intervalls von 60 Minuten auf die Suchzeit auswirkt, hängt im Wesentlichen davon ab, wie lange ein Knoten durchschnittlich im Netzwerk bleibt (Vgl. [Li04]).

Um dem Netzwerk beizutreten, muss der neue Knoten, wie bei bisher allen Peer-to-Peer-Netzwerken, einen Teilnehmer kennen. Diesen fügt er zu seinen k -Buckets hinzu und führt anschließend eine Suche nach seiner eigenen ID durch. Das hat gleich zwei Vorteile: Erstens bekommt er durch die automatische Aktualisierung der k -Buckets so Informationen über seine direkte Nachbarschaft, zweitens wird seine eigene ID durch den Vorgang des Suchens im Netzwerk bekannt.

Die Vorteile gegenüber Chord liegen in der Vielzahl der Knoten, die zum Speichern genutzt werden, sowie in der gleichzeitigen Abfrage mehrerer Knoten. Dadurch sind Daten auch nach dem Ausfall mehrerer Knoten mit hoher Wahrschein-

lichkeit noch verfügbar, außerdem werden verschiedene Knoten gleichzeitig benutzt, wodurch Sackgassen, die durch Ausfälle einzelner Teilnehmer entstehen können, umgangen werden können. Ein weiterer Vorteil liegt in der Datenstruktur der Buckets: bei jedem Suchvorgang und sonstigen Nachrichten erhält ein Knoten Informationen über viele verschiedene Teilnehmer, so dass entstehende Löcher schnell umwachsen werden. Somit ist Kademia vor allem für sehr dynamische Netzwerke geeignet, das heißt für Netzwerke, in denen häufig Knoten ausfallen und neue hinzukommen (Vgl. [May02], S.59).

Kademia löst auf elegante Weise die Probleme des Bootstrappings, des Routings und durch die redundante Suche auch der Zuverlässigkeit der Suche. Kritikpunkte an Kademia sind die komplizierte Listenverwaltung und der hohe Netzwerktraffic, der zwar durch die automatische Verteilung der IDs leicht gesenkt, durch die vielen gleichzeitigen Anfragen und das regelmäßige neue Speichern der Daten jedoch vergleichsweise hoch ist (Vgl. [Li04]). In Netzwerken, in denen es wichtig ist, dass von der ID der verwalteten Daten nicht auf die Daten selbst geschlossen werden kann, ist außerdem die verwendete und inzwischen als kryptografisch unsicher eingestufte Hashfunktion ein Sicherheitsrisiko (Vgl. [Wan05]).

3 Sicherheit

3.1 Angriffsmöglichkeiten

3.1.1 Incorrect Lookup Routing

Ein möglicher Angriff auf ein P2P-Netzwerk ist das sogenannte *Incorrect Lookup Routing*. Dabei kann ein bössartiger Knoten einen Lookup (also das Suchen eines Knotens) an einen falschen oder nicht existenten Knoten weiterleiten, um so einen weiteren Lookup zu provozieren. Eine erhöhte Netzwerklast und ggf. Nichtauffindbarkeit von Knoten sind die Folge.

Jedoch kann dieser Angriff abgewehrt werden, wenn jede Weiterleitung die Suche ein Stück näher an den Endknoten bringen muss. Dabei muss der Initiator der Suche den Vorgang beobachten können, um festzustellen, ob sich die Entfernung zum Ziel wirklich verringert (Vgl. [Gow06]).

3.1.2 Denial of Service

Eine weitere Gefahr stellen Denial of Service, kurz DoS-Attacken dar. Dabei sendet ein Angreifer eine Flut aus ggf. manipulierten Anfragen an einen Knoten, mit dem Ziel, diesen durch Überlastung lahmzulegen. Werden eine größere Anzahl von Systemen, wie z.B. in Botnetzwerken⁵, zum Durchführen einer DoS-Attake benutzt, spricht man von einem DDoS (Distributed Denial of Service) Angriff. (D)DoS-Angriffe können auf verschiedene Weisen durchgeführt werden:

SYN-Flood Bei einem SYN Flood sendet der Angreifer manipulierte IP-Pakete an den Zielrechner. Dabei wird eine falsche Absenderadresse (IP Spoofing) angegeben und ein Verbindungsaufbau eingeleitet, der jedoch nie zustande kommt. Dabei schreibt der Zielrechner jedesmal einen Eintrag in eine spezielle Tabelle. Ist diese voll, verweigert der Knoten fortan neue Verbindungen bis wieder ein Eintrag frei wird, was normalerweise erst nach etwa zwei bis drei Minuten der Fall ist.

Smurf-Attacke Bei einem Smurf-Angriff sendet ein Angreifer fortlaufend ICMP-Echo-Requests (auch als Ping bekannt) an die Broadcast-Adresse eines Netzwerks. Die Broadcast-Adresse (zu Deutsch: Rundrufadresse) ist eine Adresse, auf der alle Teilnehmer lauschen und, falls an sie eine Nachricht geschickt wird, sie verarbeiten. Trägt der Angreifer als Absender-Adresse die IP-Adresse des Opfers ein, so antworten alle im Netzwerk befindlichen Rechner auf die vermeintliche Anfrage des Opfers. Diese Attacke lässt sich jedoch verhindern, indem man den Rechner so konfiguriert, dass er nicht auf ICMP-Echo-Requests antwortet, die an die Broadcast-Adresse gerichtet sind.

Ein Sonderfall stellt die sogenannte DRDoS (Distributed Reflected Denial of Service)-Attacke dar. Bei ihr adressiert der Angreifer seine Pakete nicht an das Opfer, sondern an andere Hosts, gibt jedoch als Absenderadresse die IP des Opfers an. Anschließend wird das Opfer mit Antworten auf diese Anfragen überhäuft. So ist es für das Opfer praktisch unmöglich den Angriff zurückzuverfolgen und die Identität des Angreifers zu ermitteln (Vgl. [Ens02]).

⁵Botnetzwerke sind ein Zusammenschluss von vielen Knoten, die Befehle eines einzelnen, wie zum Beispiel den Angriffsbefehl auf einen bestimmten Knoten, befolgen.

3.2 Bedeutung für Peer-to-Peer-Netzwerke

Incorrect Lookup Routing ist in den vorgestellten DHT-Netzen Chord und Kademia ein geringes Problem. Durch die Verwaltung in den k -Buckets und die rekursive Suche in Kademia wird dieser Art des Angriffs entgegengewirkt, da nur Knoten verwendet werden, deren Distanz zum Ziel geringer ist, als die der bisher zum Auffinden des Zielknotens genutzten.

Auch der Distributed Denial of Service ist für moderne Peer-to-Peer-Netzwerke von keiner großen Bedeutung, unter anderem weil sich die Forschung viel mit diesem Thema beschäftigt hat. Kademia verschickt mehrere Anfragen gleichzeitig und nutzt so verschiedene Wege zum Ziel, so dass auch eine Großzahl von angegriffenen Knoten wenig Einfluss auf Suchanfragen hat. Durch zusätzliche Techniken wie das Result Caching, bei dem Teilnehmer, die vor dem Angriff die Daten des Opfers abgefragt haben, diese automatisch auf weitere Knoten verteilen, wird das Szenario eines erfolgreichen Angriffs noch unwahrscheinlicher.

Chord ist theoretisch anfälliger gegen DDoS-Attacken. Durch die binäre Struktur und das große Wissen einzelner Knoten über ihre Nachbarschaft und nicht zuletzt auch dadurch, dass Daten jeweils nur von einem Teilnehmer verwaltet werden, ist es denkbar, dass eine Reihe von hintereinander liegenden Knoten angegriffen und so Teile des Netzwerks gestört werden.

Eine weitere Angriffsmöglichkeit besteht dann, wenn ein Netzwerk zur Lösung des Bootstrapping-Problems eine Liste von ständig verfügbaren Knoten bereitstellt, um neuen Teilnehmern zu ermöglichen, sich auf eine möglichst einfache Art in das Netz zu integrieren. Diese können gezielt unter Beschuss genommen werden, so dass neue Teilnehmer nur noch eingeschränkt oder gar nicht mehr beitreten können, ohne einen aktiven, alternativen Knoten zu kennen.

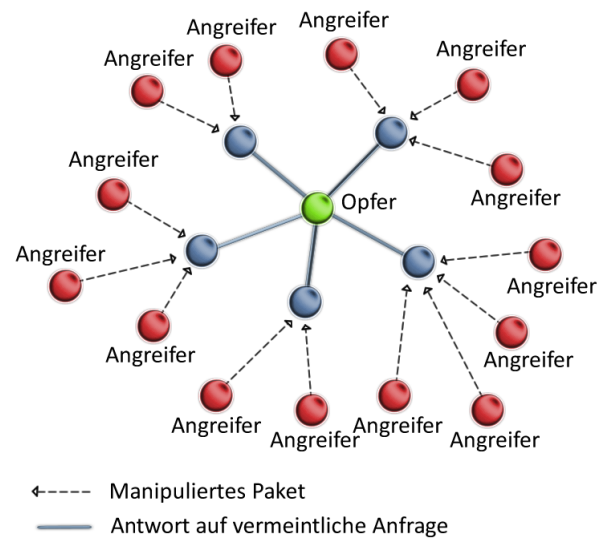


Abbildung 6: Ein DRDoS-Angriff

4 Fazit

Nach der Vorstellung der ersten DHT-basierenden Netzwerke brach eine regelrechte DHT-Hysterie aus. Diese Netzwerke ermöglichten mit relativ wenig Aufwand eine effiziente Suche. Sogar etablierte P2P-Systeme wie *eMule* und *BitTorrent* haben ihr Netzwerk inzwischen mit DHT-Unterstützung ausgestattet (Vgl. [Fal08]). Durch steigende Robustheit finden sie inzwischen sogar in verteilten Dateisystemen Verwendung, in denen ein großer Wert auf ständige Verfügbarkeit der Daten gelegt wird.

Ein allgemeines Problem ist zudem die Suche nach bestimmten Suchkriterien. Alle vorgestellten Netzwerke setzen voraus, dass die genaue ID eines gesuchten Datums bekannt ist. Eine Suche nach Teilen von Namen, Änderungsdatum oder bestimmtem Inhalt, wie man es von Suchmaschinen gewöhnt ist, ist so nicht möglich. Chord# stellt eine Modifizierung von Chord vor, die dieses Problem löst, allerdings auf Kosten der DHT (Vgl. [Sch05]). Zur Lösung dieses Problems unter Erhaltung der DHT kann eine zentrale Verwaltung verwendet werden, die Daten mit ID, Titel und anderen gewünschten Informationen katalogisiert. Diese Verwaltung würde jedoch angreifbar und das Netzwerk somit kein reines P2P-Netzwerk mehr sein.

Teil III

Distributed Hash Tables - die

Anwendung

1 Einleitung

„Es gibt eine Theorie, die besagt, wenn jemals irgendwer genau rausfindet, wozu das Universum da ist und warum, dann verschwindet es auf der Stelle und wird durch etwas noch Bizzarrereres ersetzt. Es gibt eine andere Theorie, nach der das schon passiert ist.“ (Douglas Adams)

Um ein mögliches Anwendungsgebiet der vorgestellten Netzwerke aufzuzeigen, soll hier ein Modell namens XINC entwickelt werden, das den Austausch von Textnachrichten ermöglicht und auf den vorgestellten Techniken der DHT aufbaut. Dabei wird ein zusätzlicher Schwerpunkt auf die Sicherheit gesetzt, so dass kein Teilnehmer des Netzwerks ohne dessen Willen eindeutig identifiziert werden kann. Auch die eigentliche Kommunikation zwischen zwei Knoten wird verschlüsselt.

Der Beweis der Praxistauglichkeit eines Modells lässt sich am besten durch dessen Umsetzung erbringen. Dazu soll das hier vorgestellte XINC anschließend implementiert werden. Dabei wird auf einige Probleme eingegangen, auf die wir im Zuge dieses Prozesses gestoßen sind.

2 Modell unseres Netzes XINC

2.1 Aufbau

XINC ist ein dezentrales Kommunikationsnetzwerk, das es den Nutzern erlaubt, sicher und weitgehend anonym Informationen auszutauschen. Dazu benutzt XINC neben der Vermittlungsschicht noch eine Datenschicht. Die Vermittlungsschicht ist ein auf Kademia aufbauendes Netzwerk, das für die Suche von Teilnehmern verwendet wird. Die Datenschicht regelt den eigentlichen Datenaustausch zweier Teilnehmer.

Um die Anonymität zu wahren, ist die Vermittlungsschicht so aufgebaut, dass sich Teilnehmer nur finden können, wenn sie eine wichtige Information über den

anderen kennen. Dafür wird ein RSA-Schlüsselpaar genutzt (siehe Kapitel III.2.3: RSA).

Tritt ein Knoten dem Netzwerk bei, erstellt er mithilfe des SHA-256 Hash-Algorithmus einen Hash von seinem RSA-Public-Key. Dieser Hash wird, zusammen mit seiner mittels seines RSA-Private-Key verschlüsselten Adresse und dem Port, in einem Datenpaket gespeichert und nach der oben vorgestellten Methode in dem Netzwerk gespeichert. Unter der angegebenen Adresse und dem Port läuft der Dienst der Datenschicht, der für die eigentliche Kommunikation gedacht ist. Um eine Kommunikation zu einem konkreten Teilnehmer aufzubauen, muss man nun dessen RSA-Public-Key besitzen. Dieser wird wiederum mittels SHA-256 gehasht, sodass man den Hash erhält, mit dessen Hilfe das Daten-4-Tupel, bestehend aus ID, Kontaktadresse und -port und Zeitstempel gespeichert wurde. Anschließend wird eine Suche nach der erhaltenen ID durchgeführt. Als Ergebnis erhält man die mittels des RSA-Private-Key verschlüsselten Kontaktinformationen. Mithilfe des Public-Keys können diese Informationen nun entschlüsselt werden und eine Kommunikation mittels der Datenschicht aufgebaut werden. Der Zeitstempel ist nötig, um Pakete automatisch nach einer bestimmten Zeit (bei XINC 60 Minuten) entfernen zu können.

Durch diesen Aufbau ergibt sich eine hohe Sicherheit jedes Teilnehmers. Da dieser selbst entscheiden kann, wer seinen Public-Key erhält, kann er auch nur von diesen Knoten gefunden werden. Alle anderen Teilnehmer wissen nicht, wer sich hinter einem Knoten verbirgt, selbst wenn sie zufällig für die Speicherung von dessen Kontaktdaten zuständig sind. Sie können durch die gegebenen Informationen weder an den Public-, noch an den Private-Key gelangen. Selbst bei direktem Kontakt im Netzwerk kann nicht auf die Identität geschlossen werden. Ein mögliches Problem ist die Authentizität des Knotens, der die Informationen veröffentlicht. Es ist möglich Knoten zu stören, deren Public-Keys bekannt sind, indem irgendwelche Daten mit dem Hash dieses Public-Keys veröffentlicht werden. So kann zwar die Kontaktadresse selbst nicht gefälscht werden, da diese mit dem Private-Key verschlüsselt werden muss, sucht jedoch ein Teilnehmer nach genau dieser ID, so findet er eventuell die falschen Kontaktdaten und kann sein eigentliches Ziel nicht erreichen.

Ein Nachteil dieses gesamten Aufbaus ist sicherlich die Schwierigkeit sicherzustellen, dass der RSA-Public-Key nur den vertrauenswürdigen Personen zugänglich

gemacht wird. Dazu wird eine andere Form der Kommunikation benötigt, bei der sichergestellt sein muss, dass niemand an relevante Daten gelangen kann. XINC regelt diesen Austausch nicht. Außerdem muss die daraufhin eingeleitete, direkte Kommunikation noch gesichert werden, da ansonsten Dritte diese mithören können und nicht klar ist, ob der Kommunikationspartner wirklich der ist, für den er sich ausgibt. Dazu wird zunächst dessen Identität überprüft, indem eine Reihe von Nachrichten ausgetauscht werden, auch *Handshake* genannt. Möchten die Knoten *A* und *B* kommunizieren, müssen sie dazu jeweils den Public-Key des anderen kennen. *A* verschlüsselt eine Zufallszahl x mit dem Public-Key B_e von *B* und schickt sie diesem. *B* kann die Zahl x nur dann entschlüsseln, wenn er den Private-Key B_d kennt, also wenn er wirklich *B* ist. Den SHA-256-Hash x' von dem so entschlüsselten x schickt *B* an *A*, der prüfen kann, ob der übermittelte Hash zu der Zahl x passt. Danach berechnet *B* eine Zufallszahl y und führt dasselbe Verfahren mit *A* durch, um dessen Identität zu prüfen.

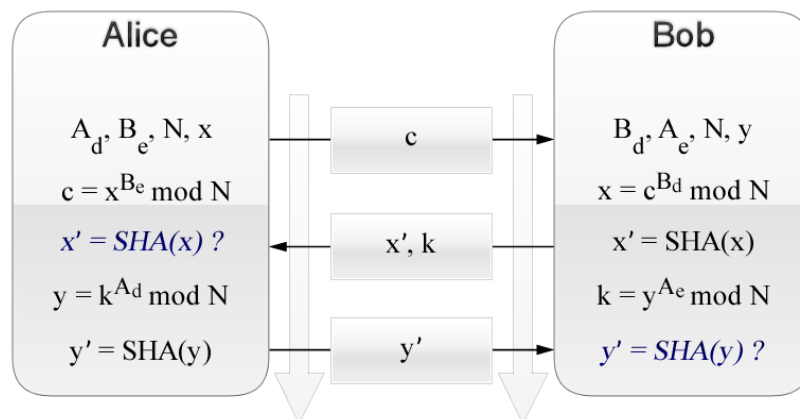


Abbildung 7: Ablauf des RSA-Handshake

Für die Sicherung der folgenden Kommunikation verwenden wir den *Diffie-Hellman-Merkle-Schlüsselaustausch* und *Rijndael*. Rijndael ist eine weitere Verschlüsselungsmethode, die in vielen Systemen genutzt wird, in denen ein hoher Anspruch an Sicherheit besteht. Anders als RSA nutzt Rijndael nur einen Schlüssel, der beiden Kommunikationspartnern bekannt sein muss. Hier stellt sich also das Problem, einen Schlüssel festzulegen, der für die Kommunikation verwendet wird, ohne dass Außenstehende diesen erfahren.

Um dieses Problem zu lösen, wird der Diffie-Hellman-Merkle-Schlüsselaustausch genutzt. Möchten die Knoten *A* und *B* eine sichere Verbindung aufbauen, generiert *A*

zwei Zahlen: p , welche eine möglichst große Primzahl ist, und eine Primitivwurzel $g \bmod p$, die kleiner p sein muss. g ist genau dann eine Primitivwurzel modulo p , wenn gilt: $l = g^k \bmod p$ für alle $1 \leq l < n$ und beliebige natürliche Zahlen k . Diese Zahlen werden über die unverschlüsselte Leitung B mitgeteilt. Nun generieren beide jeweils eine zufällige Zahl X_A bzw. X_B , die geheim bleibt. Anschließend werden jeweils die Zahlen

$$Y_{A,B} = g^{X_{A,B}} \bmod p$$

berechnet und über die unsichere Verbindung Y_A an B und Y_B an A geschickt. Nun kann von beiden der zu verwendende Schlüssel K berechnet werden, ohne dass potenzielle Mithörer aus den direkt übertragenen Zahlen diesen in Erfahrung bringen können:

$$K = Y_B^{X_A} \bmod p = Y_A^{X_B} \bmod p$$

Mithilfe dieses Schlüssels können A und B mittels Rijndael eine sichere Verbindung aufbauen und kommunizieren (Vgl. [Ewa06]).

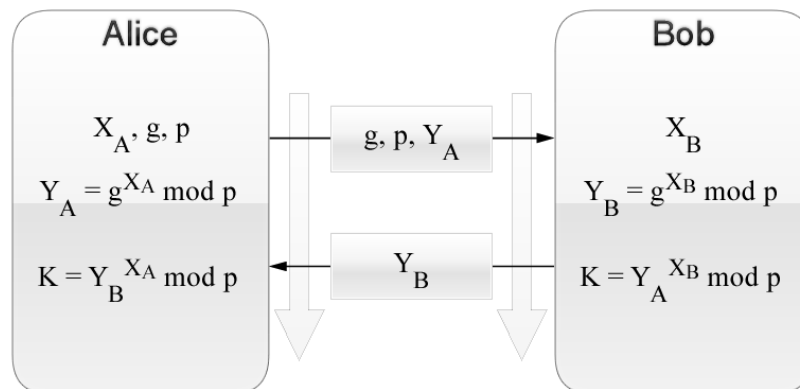


Abbildung 8: Ablauf des Diffie-Hellman-Merkle-Schlüsselaustauschs

2.2 Unterschiede zu Chord und Kademia

Ein wesentlicher Unterschied in der Vermittlungsschicht zu Kademia besteht im Ausschluss der in Kademia verwendeten SHA-1-Hashfunktion. Diese ist nicht mehr sicher genug, um ausschließen zu können, dass anhand des Hashs ein mögliches Datum berechnet werden kann. Damit wäre es möglich das Netzwerk zu stören. Des-

halb haben wir SHA-1 durch die kryptografisch sichere SHA-256-Funktion ersetzt und dabei den virtuellen Adressraum für IDs von 160 bit auf 256 Bit erweitert. Die kryptografische Sicherheit von SHA-256 ergibt sich daraus, dass es bisher keinen bekannten Angriff zur Erzeugung einer Kollision gibt.

Des Weiteren wird die ID eines XinC-Daten-4-Tupel nicht aus dem Hashwert des Inhaltes, sondern aus dem Hash des öffentlichen RSA-Schlüssels des Erstellers gebildet und der Inhalt mittels RSA vom Ersteller mit seinem privaten RSA-Schlüssel verschlüsselt. So ist es möglich die Authentizität eines Daten-4-Tupels zu prüfen, wodurch auch sichergestellt ist, dass nur der Besitzer des zugehörigen privaten RSA-Schlüssels den Inhalt des Daten-4-Tupels geschrieben haben kann. Dabei muss beachtet werden, dass der Zeitstempel nicht mit verschlüsselt wird, da ansonsten der das Paket verwaltende Knoten diesen nicht prüfen kann.

Da es bei Kommunikationsnetzwerken allgemein üblich ist sich häufiger ab- und anzumelden als in Netzwerken, die Hintergrundservices wie den Dateientausch anbieten, haben wir außerdem die Zeit, die ein Datum gespeichert wird, von 24 Stunden auf 60 Minuten gesenkt, da ansonsten durch einen IP-Adressenwechsel eines Knotens dieser durch alte im Netzwerk gespeicherte Kontaktdaten möglicherweise nicht mehr auffindbar ist.

2.3 RSA

RSA ist ein asymmetrisches Kryptosystem, das 1977 von RONALD L. RIVEST, ADI SHAMIR und LEONARD ADLEMAN am Massachusetts Institute of Technology entwickelt wurde. Es basiert auf der Idee, zwei unterschiedliche Schlüssel zum Ver- und Entschlüsseln einer Nachricht zu benutzen, einen sogenannten öffentlichen und privaten Schlüssel. Es ist jedoch nur mit sehr großem Aufwand möglich, aus dem öffentlichen den privaten Schlüssel zu erhalten. Sind die Schlüssel gut gewählt, ist es praktisch unmöglich diesen zu berechnen, da es bei bisherigen Rechenkapazitäten zu lange dauert.

Beim RSA-Verfahren wird eine Falltürfunktion verwendet, also eine mathematische Funktion, welche im Sinne der Komplexitätstheorie nur „schwer“ ohne gewisses Zusatzwissen (dem andern Key) umkehrbar ist. Das daraus resultierende

Problem ist als RSA-Problem bekannt: Das Finden einer Ganzzahl P , so dass gilt:

$$P^e \equiv C \pmod{N}$$

bei gegebenen Ganzzahlen N , e und C , wobei N das Produkt zweier großer Primzahlen mit $2 < e < N$ teilerfremd zu $\varphi(N)$ und $0 \leq C < N$ ist. C ist eine zufällig gewählte Zahl aus diesem Bereich. $\varphi(N)$ bezeichnet dabei die Eulersche φ -Funktion, welche für jede natürliche Zahl N angibt, wie viele Zahlen $a \leq N$ zu ihr teilerfremd sind. Es gilt:

$$\varphi(N) := |\{1 \leq a \leq N \mid \gcd(a, N) = 1\}|$$

Dabei bezeichnet $\gcd(a, N)$ den größten gemeinsamen Teiler von a und N .

Durch diesen Aufbau ist es möglich, einen Klartext mithilfe eines der beiden Schlüssel so zu verschlüsseln, dass man ihn nur mit dem jeweils anderen wieder entschlüsseln kann. Der Nachteil ist der hohe Rechenaufwand, der für die Ver- und Entschlüsselung nötig ist. Die Sicherheit dieses Verfahrens hängt stark von der Wahl der für die Bildung von N genutzten Primzahlen ab. Hier müssen entsprechend hohe Zahlen gewählt werden, da ansonsten die für die unautorisierte Dechiffrierung nötige Primfaktorzerlegung in einer sehr kurzen Zeit durchführbar ist (Vgl. [XUE06]).

3 Umsetzung von XINC

3.1 Verwendete Programmiersprache

Wir haben XINC mithilfe der Programmiersprache ObjectPascal implementiert. ObjectPascal nutzt eine erweiterte Pascal-Syntax und findet (in leicht abgeänderter Form) zum Beispiel in der Windows-Entwicklungsumgebung Delphi Anwendung. ObjectPascal unterstützt außerdem, wie der Name bereits vermuten lässt, objektorientierte Programmierung. Außerdem wird Operatorenüberladung unterstützt, was besonders für die kryptografischen Funktionen von Bedeutung ist, in denen mit besonders großen Zahlen gerechnet wird. Hier reichen die bereitgestellten primitiven Datentypen nicht mehr aus, so dass ein spezieller, für beliebig große Zahlen ausgelegter Datentyp genutzt wird.

Als Compiler wird der quelloffene und kostenlose Free Pascal Compiler verwendet, da dieser für viele Plattformen, unter anderem Unix und Linux, Windows

und Mac OS, verfügbar ist. Dementsprechend wird auch der geschriebene Programmcode plattformunabhängig gehalten. Neben diesen Vorteilen gibt es für FreePascal außerdem schon eine Reihe von Netzwerkkomponenten, die bequem zu verwenden sind. Somit konzentriert sich die Entwicklung auf XINC selbst.

3.2 Umsetzung der k -Buckets

Der Hauptkritikpunkt an Kademia, die Komplexität der verwendeten k -Buckets, soll hier noch einmal aufgegriffen werden. Für eine Klasse, die diese Funktionalität anbietet, ergeben sich aus den durch Kademia definierten Anforderungen folgende Operationen:

1. Ist ein Knoten mit gegebener ID bereits in den k -Buckets enthalten?
2. Ist der für die gegebene ID zuständige Bucket voll?
3. Bewege den Knoten der gegebenen ID an das Ende des Buckets
4. Füge den gegebenen Knoten an das Ende des zuständigen Buckets ein
5. Liefere die k der gegebenen ID nächsten Einträge

Intern werden zur Speicherung der Knoten 256 einfache Reihungen verwendet. Zwar muss dann im Falle einer Größenänderung eines Buckets, wenn neue Knoten hinzukommen, die noch nicht in den k -Buckets sind, der Speicher des entsprechenden Buckets an eine neue, größere Position im Hauptspeicher kopiert werden. Allerdings können die Reihungen maximal k Einträge groß werden und ein Kleinerwerden ist ohnehin ausgeschlossen. Somit lässt sich der maximale Speicheraufwand mit maximal $256 * k * n$ Byte zuverlässig abschätzen, wobei k wie in Kademia 20, und n der konstante Platzbedarf eines einzelnen Knotens in Byte ist. Aus unserer Implementation ergibt sich n aus der Größe der ID, die mit 256 Bit genau 32 Byte groß ist, der IP-Adresse, die 4 Byte belegt, sowie der Portnummer, die 2 Byte nutzt. Der maximal verbrauchte Hauptspeicher hat mit $256 * 20 * (32 + 4 + 2) = 194560 = 190$ Kilobyte Platzbedarf überschaubare Maße.

Die Implementierungen der Operationen eins bis vier ergeben sich analog aus der Nutzung der Reihungen. Zunächst wird mittels XOR der zuständige Bucket ermittelt und Anhand von Reihungs-Operationen die Einträge verschoben, eingefügt

oder der Füllstand des Buckets abgefragt. Für die konkrete Umsetzung siehe Appendix, Unit `uXinCBucket`. Die Suche nach den k nächsten Einträgen wird mittels Quicksort gelöst. Dazu werden alle verwalteten Knoten in eine Reihung gespeichert und diese mittels Quicksort anhand der Abstände der IDs der Knoten zur Ziel-ID sortiert. Die ersten 20 Einträge werden dann weiter genutzt (siehe Appendix: Unit `uXinCBucket`).

Als weitere Optimierung könnte die Klasse so modifiziert werden, dass sie bereits eine nach dieser Form vorsortierte Liste bereit stellt, oder den Platz für die gesamten Listen bereits beim Programmstart alloziert.

3.3 Netzwerk-Protokoll

Für die Datenübertragung nutzt XINC ein Netzwerkprotokoll, das speziell für diesen Zweck entwickelt wurde. Jedes Paket besteht dabei aus der eigenen ID, einer Nummer für den Befehl und einer Datensektion mit variabler Länge. Dabei belegt die ID 32 Byte, da sie aus 256 Bit besteht. Aufgrund der wenigen benötigten Funktionen genügt für den Befehl ein einzelnes Byte, das die numerische Repräsentation des Befehls enthält. Bei jeder empfangenen Nachricht wird außerdem der entsprechende Knoten anhand der übertragenen ID gespeichert oder aktualisiert. Es werden folgende Befehle genutzt:

- ▷ ACK ist die Antwort auf die PING- und STORE-Anfragen. Die Datensektion ist leer.
- ▷ PING veranlasst den Empfänger, mit einer ACK-Nachricht zu antworten.
- ▷ FIND_NODE fragt einen Knoten nach dessen zur gegebenen ID nächsten Knoten. Die Datensektion enthält dabei die 32 Byte große gesuchte ID. Der Empfänger antwortet mit FIND_NODE_ANSWER.
- ▷ FIND_NODE_ANSWER enthält das Ergebnis einer Suchanfrage in Form einer Knoten-Liste. Die Datensektion enthält die Anzahl der übermittelten Knoten als 4 Byte Ganzzahl und danach die entsprechenden Knoten, bestehend aus ID (32 Byte), Adresse (4 Byte) und Port (2 Byte).
- ▷ FIND_VALUE fragt den Empfänger nach einem Datum der ID in der Datensektion. Dieser antwortet mit FIND_VALUE_ANSWER.

- ▷ FIND_VALUE_ANSWER enthält entweder das angefragte Datum, oder, falls er dieses nicht besitzt, eine Liste der ihm bekannten der ID nächsten Knoten. Zur Differenzierung zwischen den beiden Typen beginnt die Datensektion mit einem Byte, das angibt, welche Informationen folgen. Ist es 0, folgt die gleiche Datensektion wie bei FIND_NODE_ANSWER, sonst das angefragte Datum, bestehend aus dessen ID, dem Zeitstempel, der Adresse und dem Port.
- ▷ STORE weist den Empfänger an, das in der Datensektion übertragene Datum zu speichern. Dieser antwortet mit ACK.

Teil IV

Fazit/Ausblick

„Ich bin vielleicht nicht dort, wo ich hin wollte, aber ich denke doch, dass ich da bin, wo ich sein muss.“ (Douglas Adams)

Die Technik der Distributed Hash Tables steckt noch in den Kinderschuhen und man darf gespannt sein, wohin sie sich im Laufe der nächsten Jahre entwickeln wird. Ein Anwendungsbereich wäre zum Beispiel das World Wide Web. Durch ein entsprechendes P2P-Netz ließen sich so die Inhalte über viele Teilnehmer verteilen und Zensur erschweren. Allerdings kann das auch zur Verbreitung illegaler Inhalte verwendet werden. Außerdem können einmal in das Netzwerk gestellte Dokumente nicht ohne Weiteres wieder herausgenommen werden.

Eine weitere wichtige Frage ist die nach der rechtlichen Situation. P2P-Netzwerke sind durch Tauschbörsen für urheberrechtlich geschütztes Material populär geworden und werden auch weiterhin ein Schauplatz für Rechtsverletzungen sein. Eine Kontrolle der Strukturen ist schwierig und die rechtliche Situation noch unklar.

XINC stellt durch die Verwendung von Kademia ein robustes Netzwerk dar, das durch die zusätzliche Sicherheit sowohl Anonymität als auch Sicherheit in der Kommunikation gewährleistet. Ein bisher nicht gelöstes Problem ist die Komplexität der Kontaktsuche. Damit ein Teilnehmer gefunden werden kann, muss dessen RSA-Public-Key bekannt sein. Wie genau diese Information sicher übertragen wird, regelt XINC nicht.

Die Implementierung von XINC konzentrierte sich bisher auf das verwendete Peer-to-Peer-Netzwerk in der Vermittlungsschicht, welche inzwischen auch vollständig funktionstüchtig ist. Die Implementierung der Datenschicht verzichtet bislang auf sämtliche Verschlüsselung, da dafür die Zeit fehlte. Stattdessen bietet sie minimalistische Chatfunktionen, so dass bereits die Funktionalität der Vermittlungsschicht gezeigt werden kann. Als nächste Schritte sind die vollständigen Umsetzungen der Verschlüsselungen geplant, damit XINC möglichst bald eine ernstzunehmende Alternative zu etablierten Kommunikationsprogrammen darstellt.

Teil V

Appendix

A Programmcode

Folgende Units werden bereit gestellt:

- ▷ *uXinCBucket.pas*: Enthält die Klasse TXinCBucket, welche die einzelnen *k*-Buckets verwaltet.
- ▷ *uXinCBucketList.pas*: Stellt eine typisierte Liste zur Verfügung, welche die XinC-Knoten enthält.
- ▷ *uXinCCommon.pas*: Beinhaltet diverse oft genutzte Funktionen, wie das Umwandeln von eigenen Datentypen in Zeichenketten.
- ▷ *uXinCConst.pas*: In dieser Unit werden alle Konstanten definiert.
- ▷ *uXinCDataList.pas*: Wie auch in *uXinCBucketList* wird hier eine typisierte Liste implementiert, jedoch für Daten-Pakete.
- ▷ *uXinCDHT.pas*: Die wohl wichtigste Unit. Sie stellt die Klasse TXinCDHT bereit, in welcher das XinC-DHT-Netzwerk verwaltet wird.
- ▷ *uXinCExceptions.pas*: Definiert die Exception EXinCException, die die Basis für weitere spezielle Exceptions zur Behandlung von Laufzeitfehlern darstellt.
- ▷ *uXinCGlobal.pas*: Enthält die in XinC benutzten Datentypen.
- ▷ *uXinCNetIndy.pas*: Implementiert die abstrakte Klasse TXinCNetManager der Unit *uXinCNetManager* auf Basis der Komponentensammlung Internet Direct.
- ▷ *uXinCNetManager.pas*: Beschreibt eine abstrakte Klasse, die die Funktionalität der von ihr abgeleiteten Netzwerk-Klassen festlegt.
- ▷ *uXinCNetwork.pas*: Stellt die Schnittstelle zwischen Vermittlungs- und Datenschicht dar.

Unvollständige Units:

- ▷ *uXinCBigInt.pas*: Stellt einen speziellen Datentyp zum Umgang mit sehr großen Ganzzahlen im Bereich von 0 bis $2^{4294967296} - 1$ bereit (Begrenzt durch den Arbeitsspeicher und die verwendete 32Bit-Architektur).
- ▷ *uXinCBigIntBase.pas*: Implementiert die Basisfunktionalität für *uXinCBigInt.pas*.
- ▷ *uXinCCryptRSA.pas*: Bietet Funktionen zur Ver- und Entschlüsselung von Daten nach dem RSA-Verfahren.
- ▷ *uXinCNetLnet.pas*: Entspricht in ihrer Funktion der Unit *uXinCNetIndy*, jedoch werden die INet-Komponenten genutzt.

B Beispielanwendung

Um die Funktionalität des Netzwerks zu demonstrieren und eine minimalistische Kommunikation zu ermöglichen, liegt eine kleine Beispielanwendung bei. Bei dem ersten Programmstart wird eine zufällige ID generiert und im Arbeitsverzeichnis unter dem Namen „id.dat“ gespeichert. Der eigene Name, der angezeigt wird, kann in der Datei „settings.txt“ verändert werden. Es werden zwei aufeinanderfolgende Ports genutzt, der kleinere für die UDP-Verbindung der Vermittlungsschicht, der größere für die auf TCP basierende Datenschicht. Diese müssen bei der Verwendung eines Routers eventuell erst entsprechend freigegeben werden.

Mit Klick auf *Info*→*Network* erfährt man, mit welchen anderen Knoten man verbunden ist und welche Daten gespeichert werden. Ein Klick auf einen einzelnen Eintrag offenbart den Inhalt. Mit *Connect to custom node* kann manuell zu einem Knoten verbunden werden. Um das Bootstrapping zu erleichtern, haben wir unter der Adresse *inion.no-ip.org*, Port 23456 einen Knoten eingerichtet, der genutzt werden kann. Ist man zu mindestens einem Knoten verbunden, können mittels *Make me public* die eigenen Kontaktdaten im Netzwerk veröffentlicht werden.

Um mit einem anderen Teilnehmer bekannter ID zu kommunizieren, kann mithilfe von *Buddys*→*Add* unter Angabe der ID eine Verbindung zu diesem aufgebaut werden.

Literatur

- [Bac08] BACHFELD, DANIEL: *DDoS-Attacke auf InternetX [Update]*. In: <http://www.heise.de/newsticker/DDoS-Attacke-auf-InternetX-Update-/meldung/119274> (Stand 16.03.2009). 2008.
- [Bej05] BEJAN, ALINA/GHOSH, SUKUMAR: *Self-optimizing DHTs Using Request Profiling*. In: TERUO HIGASHINO (ED.) (Herausgeber): *Principles of Distributed Systems*, Seiten 140–153. Springer-Verlag, Berlin 2005.
- [Ens02] ENSER, FRANK: *Konzepte von Betriebssystem-Komponenten: Denial of Service-Attacken, Firewalltechniken*. In: http://www4.informatik.uni-erlangen.de/Lehre/SS02/PS_KVBK/talks/handout-sifrense.pdf (Stand 15.03.2009). 2002.
- [Ewa06] EWALD, GERD: *Asymmetrische Verfahren: Diffie/Hellmann*. In: http://www.regechsen.de/phpwcms/index.php?krypto_asym_dh (Stand 15.03.2009). 2006.
- [Fal08] FALKNER, JARRET/PIATEK, MICHAEL/JOHN JOHN P./KRISHNAMURTHY ARVIND/ANDERSON THOMAS: *Profiling a Million User DHT*. In: <http://www.cs.washington.edu/homes/arvind/papers/dht.pdf> (Stand 19.03.2009). Washington, 2008.
- [Fie99] FIELDING, R./GETTYS, J./MOGUL J. C./FRYSTYK H./MASINTER L./LEACH P./BERNERS-LEE T.: *Hypertext Transfer Protocol – HTTP/1.1*. In: <http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf> (Stand 14.03.2009). 1999.
- [Gow06] GOW, CHRISTOPH/LEEMANN, ADRIAN/SADAT AMIR: *Security in Peer to Peer Systemen*. In: <http://www.csg.uzh.ch/teaching/ss06/comsys/extern/talk10.pdf> (Stand 18.03.2009), Seite 18. 2006.
- [Kon09] KONDO, DERRICK/JAVADI, BAHAM/MALECOT PAUL/CAPELLOFRANCK/ANDERSON DAVID P.: *Cost-Benefit Analysis of Cloud Computing versus Desktop Grids*. In:

http://mescal.imag.fr/membres/derrick.kondo/pubs/kondo_hcw09.pdf
(Stand 14.03.2009). 2009.

- [Li04] LI, JINYANG/STRIBLING, JEREMY/GIL THOMER M.: *Comparing the Performance of Distributed Hash Tables under Churn*. In: GEOFFREY M. VOELKER, SCOTT SHENKER (EDS.) (Herausgeber): *Peer-to-Peer Systems III*, Seiten 87–99. Springer-Verlag, Berlin 2004.
- [Mah07] MAHLMANN, PETER/SCHINDELHAUER, CHRISTIAN: *Peer-to-Peer Netzwerke: Algorithmen und Methoden*. Springer-Verlag, Berlin 2007.
- [May02] MAYMOUNKOV, PETAR/MAZIÈRES, DAVID: *Kademlia: A Peer-to-Peer Information System Based on the XOR Metric*. In: PETER DRUSCHEL, FRANS KAASHOEK, ANTONY ROWSTRON (EDS.) (Herausgeber): *Peer-to-Peer Systems*, Seiten 53–65. Springer-Verlag, Berlin 2002.
- [Rat02] RATNASAMY, SYLVIA/STOICA, ION/SHENKER SCOTT: *Routing Algorithms for DHTs: Some Open Questions*. In: PETER DRUSCHEL, FRANS KAASHOEK, ANTONY ROWSTRON (EDS.) (Herausgeber): *Peer-to-Peer Systems*, Seiten 45–52. Springer-Verlag, Berlin 2002.
- [Sch05] SCHÜTT, THORSTEN/SCHINTKE, FLORIAN/REINEFELD ALEXANDER: *Chord#: Structured Overlay Network for Non-Uniform Load-Distribution*. Technischer Bericht, Konrad-Zuse-Zentrum für Informatik, Berlin 2005.
- [Sch09] SCHULZE, HENDRIK/MOCHALSKI, KLAUS: *Internet Study 2008/2009*. In: http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009 (Stand 14.03.2009). ipoque, 2009.
- [Wan05] WANG, XIAOYUN/YIN, YIQUN LISA/YU HONGO: *Finding Collisions in the Full SHA-1*. In: http://cryptome.org/wang_sha1_v2.zip (Stand 14.03.2009). 2005.
- [Xue06] XUE, YUAN: *Network Security: RSA Algorithm*. In: <http://vanets.vuse.vanderbilt.edu/~xue/cs291fall06/lecture12.pdf> (Stand 15.03.2009). 2006.

Abbildungsverzeichnis

1	<i>Verteilung des Internet-Traffics nach Protokollen in Deutschland 2009</i>	2
2	<i>Schematische Darstellung des Napster-Netzwerks</i>	5
3	<i>Eine Anfrage im Gnutella-Netzwerk</i>	6
4	<i>Das Chord-Netzwerk und die Finger des Knotens c_4</i>	10
5	<i>Schematische Suche nach ID 20 durch Knoten c_4</i>	11
6	<i>Ein DRDoS-Angriff</i>	18
7	<i>Ablauf des RSA-Handshake</i>	22
8	<i>Ablauf des Diffie-Hellman-Merkle-Schlüsselaustauschs</i>	23