Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Betreuer:	Dipl.-Inf. Michael Kinateder
Beginn am:	01.05.2003
Beendet am:	31.10.2003
CR-Klassifikation:	C.2.4, D.4.6, E.2, E.3, H.2.4, H.3.3, K.4.1

Diplomarbeit Nr. 2097

Konzeption und Realisierung eines Peer-to-Peer-Empfehlungsdienstes

Ralf Terdic

Universität Stuttgart
Fakultät für Informatik, Elektrotechnik und Informationstechnik
Institut für Parallele und Verteilte Systeme
Abteilung Verteilte Systeme
Universitätsstraße 38
D-70569 Stuttgart

Inhaltsverzeichnis

EINLEITUNG **1**

1.1 MOTIVATION 1
1.2 PROJEKT UNITEC 2
1.3 GLIEDERUNG DER ARBEIT 3

DAS SYSTEMMODELL VON UNITEC **5**

2.1 SYSTEMGRUNDLAGEN 5
Basisbegriffe *5*
Datenmodelle *6*
2.2 ARCHITEKTUR 8
Nachteile eines zentralisierten Modells *9*
Das UniTEC-Modell *10*
2.3 FUNKTIONSWEISE 11
Anfragen *11*
Nachbarschaft *12*
Vertrauensketten *12*
Reaktion auf Anfragen *12*
Aktualisierung der Modelle *13*

VERWANDTE ANSÄTZE **15**

3.1 GRUNDLAGEN ZUR ANONYMITÄT 15
Flooding *15*
DC-Netze *16*
Mix-Netze und Onion Routing *16*
3.2 PROJEKTE IM BEREICH ANONYMITÄT 18
Crowds *18*
Freenet *19*
Free Haven *20*
Tarzan *20*
3.3 VERTEILTE HASHTABELLEN 20
DHT-Grundlagen *20*
Chord *21*

ANONYMITÄT IN PEER-TO-PEER-UMGEBUNGEN **27**

4.1 ANFORDERUNGEN 27
Dienstart *27*
Anonymität *27*
Peer-to-Peer-Kommunikation *28*
Skalierbarkeit *28*
Dienstgüte *28*
4.2 LÖSUNGSENTWURF 28
Destination Routing *28*
Extended Destination Routing *30*
4.3 PSEUDONYM-IDS 31
4.4 VERZEICHNISDIENST 33
Verteilter Verzeichnisdienst *33*
Routing Update *34*
Lookup *37*

4.5 EXTENDED DESTINATION ROUTING 38
Erweiterung des Routing-Headers *38*
Pattern Matching-Angriff *38*
Last Mix-Angriff *40*
Kaskadenerweiterung mit Onion Routing *41*
4.6 MIX-INFORMATIONEN 42
Verfügbarkeit *43*
Mix Discovery *43*
Load-Wert *43*

DER EMPFEHLUNGSDIENST 45

5.1 ANFORDERUNGEN 45
Empfehlungen *45*
Anfragen *46*
Antworten *47*
Zugriffskontrolle *47*
5.2 LÖSUNGSENTWURF 47
Empfehlungsmodell *47*
Anfragen *49*
Erreichbarkeit *52*
Antworten *52*
Interessensgebiete *55*

REALISIERUNG 57

6.1 VERWENDETE TECHNOLOGIEN 57
XML *57*
Volltextsuche *58*
Remote Procedure Calls *59*
Kryptographie *60*
Logging *60*
6.2 DATENHALTUNG 61
XML-Persistenz *61*
Volltextsuche *61*
Cache-Container *62*
Storage Manager *62*
6.3 KOMMUNIKATIONSSCHICHT 62
Implementierung *63*
Performanzanalyse *65*
6.4 EMPFEHLUNGSDIENST 71

FAZIT 73

7.1 ZUSAMMENFASSUNG 73
7.2 AUSBLICK 74

ANHANG 77

A. KRYPTOGRAPHISCHE GRUNDLAGEN 77
Symmetrische Verfahren *77*
Asymmetrische Verfahren *77*
Hashfunktionen *78*
B. SYNTAX VON VOLLTEXTANFRAGEN 79
Terme *79*
Boolesche Operatoren *79*
Felder *79*
Gruppierung *79*
Term-Modifikatoren *80*
Escaping *80*
C. PARAMETER DER KONFIGURATIONSDATEI `UNITEC.PROPERTIES` 81

D. IMPLEMENTIERUNG 84
Package-Hierarchie *84*
Klassen des Storage-Frameworks *85*
Basisklassen *89*
Klassen des Kommunikationssubsystems *91*
Klassen des Empfehlungsdiensts *98*

LITERATURVERZEICHNIS 101

STICHWORTVERZEICHNIS 105

GLOSSAR 107

Abbildungsverzeichnis

Abbildung 2-1: Kategorien-Abhängigkeitsgraph im Trust-Modell 8
Abbildung 2-2: Beispiel für eine zentralisierte Architektur 9
Abbildung 3-1: Nachrichtentransport entlang einer Mix-Kaskade 17
Abbildung 3-2: Anordnung der Knoten aus Tabelle 3-1 im Chord-Ring 22
Abbildung 3-3: Datenspeicherung im Chord-Ring 23
Abbildung 3-4: Redundanz durch Erhöhung des Parameters `r` (`r=2`) 24
Abbildung 4-1: Routing-Header und davon bestimmte Route 29
Abbildung 4-2: Header- und Key-Lookup 30
Abbildung 4-3: Keine Senderanonymität durch manipulierten Header 31
Abbildung 4-4: Routing Update-Protokoll 36
Abbildung 4-5: Lookup, Versenden einer Nachricht und Bestätigung 37
Abbildung 4-6: Erweiterung des Routing-Headers 39
Abbildung 4-7: Kaskadenerweiterung mit Onion Routing 42
Abbildung 4-8: Mix Discovery und MI-Liste mit Load-Wert 44
Abbildung 5-1: Signierung der Vertrauenskette 51
Abbildung 5-2: Interessensgebiete 55
Abbildung 6-1: Werte für RTT bei lokaler Messung ohne Wartezeit 67
Abbildung 6-2: Werte für RTT bei lokaler Messung mit Wartezeit 67
Abbildung 6-3: Werte für RTT bei Cluster-Messung ohne Wartezeit 69
Abbildung 6-4: Werte für RTT bei Cluster-Messung mit Wartezeit 70

Tabellenverzeichnis

Tabelle 3-1: Knoten-IDs in einem Chord-Ring 22
Tabelle 6-1: Umgebung für lokale Messung 65
Tabelle 6-2: Werte für RTT (in ms) bei lokaler Messung ohne Wartezeit 66
Tabelle 6-3: Werte für RTT (in ms) bei lokaler Messung mit Wartezeit 66
Tabelle 6-4: Umgebung für Cluster-Messung 68
Tabelle 6-5: Werte für RTT (in ms) bei Cluster-Messung ohne Wartezeit 68
Tabelle 6-6: Werte für RTT (in ms) bei Cluster-Messung mit Wartezeit 69

Kapitel 1

Einleitung

Dieses Kapitel bietet eine Einführung in die Thematik der Diplomarbeit, indem es die Gründe für die Entwicklung eines Empfehlungsdiensts aufführt und das Rahmenprojekt UniTEC und dessen Ziele kurz beschreibt.

1.1 Motivation

Das Internet wurde bekanntlich während des Kalten Krieges in den USA als militärische Infrastruktur konzipiert und später als Kommunikationswerkzeug an Universitäten verwendet. Nach dessen Kommerzialisierung Ende der Achtziger und der anschließenden Entwicklung des World Wide Web wurde es auch für die breite Masse zugänglich, sowohl als Kommunikationsplattform als auch für das Abrufen von Informationen verschiedener Art.

Parallel zu dieser explosiven Entwicklung wurde auch die eingesetzte Technologie ausgereifter. Zum einen konnten nun die Webseiten dynamisch gestaltet werden, so dass sie nicht nur zur multimedialen Präsentation von statischen Daten dienen konnten, sondern auch eine interaktive Nutzung ermöglichten. Ferner wurde die Kommunikationssicherheit erhöht, indem neue Protokolle speziell für den sicheren Transport empfindlicher Daten entwickelt wurden, wie etwa SSL (siehe [SSL96]).

Das kommerzielle Potential des World Wide Web, das anfangs hauptsächlich in der Möglichkeit der Veröffentlichung statischer Informationen über ein Unternehmen bestand, wurde durch diese Entwicklungen stark geprägt, da sie die Realisierung von Online-Shops ermöglichten und den Unternehmen dadurch neue, günstige Vertriebswege öffneten. Die weltweite Erkenntnis dieses Potentials hatte als Folge die Entstehung einer riesigen Menge an Online-Shops und hiermit die Entwicklung des E-Commerce, des elektronischen Handels. Jeder, der etwas anzubieten hatte, konnte mit geringem Startkapital ein Unternehmen gründen und diesen neuen Vertriebsweg nutzen. Es entstanden jedoch auch Variationen wie Online-Auktionshäuser, -Tauschbörsen und -Flohmärkte.

Nach der anfänglichen Euphorie trat um die Jahrtausendwende jedoch Ernüchterung ein. Die weltweite Rezession wirkte sich auf die meisten Bereiche der Wirtschaft aus, und der elektronische Handel blieb auch nicht verschont. Eine viel versprechende Möglichkeit, diesem wieder zu einem Aufschwung zu verhelfen, ist die Erhöhung der Akzeptanz, der einige Hürden im Weg stehen.

Eines der wichtigsten Hindernisse, die einer uneingeschränkten Nutzung elektronischen Handels entgegenstehen, ist das fehlende Vertrauen in die Anbieter. Der Kunde stand - und steht nach wie vor - vor einer unübersichtlichen Menge von Angeboten, die meistens von sehr unterschiedlicher Qualität sind. Um eine sinnvolle Entscheidung treffen zu können, benötigt er Informationen bezüglich der Güte der Angebote, die ihm zur Verfügung stehen.

Informationen über die Qualität der Angebote können in verschiedener Form vorliegen; je nach Form haben diese Informationen unterschiedlichen Wert. Am wertvollsten sind sicherlich eigene Erfahrungen mit dem zu erwerbenden Produkt oder der in Anspruch zu nehmenden Leistung, diese sind aber in der Regel nur selten verfügbar. Sehr wertvoll sind auch Empfehlungen, also Beschreibungen fremder Erfahrungen, diese sind jedoch insofern problematisch, als ihr Wert von der Vertrauenswürdigkeit des Empfehlenden abhängt.

Als Entscheidungshilfe können aber auch Erfahrungen und Empfehlungen herangezogen werden, die sich nicht direkt auf das Angebot beziehen, sondern auf eine breitere Klasse wie etwa die Produkt- oder Leistungskategorie, zu der das Angebot gehört. Schließlich kann es auch sinnvoll sein, Erfahrungen und Empfehlungen zu berücksichtigen, die sich auf einen Hersteller oder Leistungserbringer allgemein beziehen. Diese werden häufig dann berücksichtigt, wenn die damit verbundenen einzelnen Erfahrungen stark positiv oder stark negativ waren, da diese das Vertrauen in den Hersteller bzw. Leistungserbringer in hohem Maße beeinflussen.

1.2 Projekt UniTEC

Die vorliegende Diplomarbeit wurde im Rahmen des Projekts *UniTEC* (Universal Trust Architecture for Electronic Commerce) erstellt. Das Projekt entsteht in Zusammenarbeit der Abteilung *Verteilte Systeme* des Instituts für Parallele und Verteilte Systeme der Universität Stuttgart mit der Firma *Hewlett Packard*.

Der Ansatz, den das Projekt verfolgt, basiert auf Empfehlungen, die von den Benutzern des Systems formuliert und gespeichert werden. Diese Empfehlungen werden von anfragenden Benutzern gefunden, worauf sie von diesen gelesen und später bewertet werden. Die Bewertung einer Empfehlung beeinflusst das Vertrauen des Anfragenden gegenüber dem Ersteller der Empfehlung. Anhand der hierdurch entstehenden Vertrauenswerte wird entschieden, an wen neue Anfragen gesendet werden.

Das UniTEC-Projekt versucht Antworten auf zentrale Fragen zu finden, etwa nach der Art und Weise, wie Empfehlungen und Bewertungen verteilt gespeichert werden können, wie Anfragen formuliert und beantwortet werden, auf welche Weise Vertrauen quantifiziert, gespeichert, aktualisiert und bei der Beantwortung von Anfragen berücksichtigt werden kann, und wie bei all diesen Aktionen der Schutz der persönlichen Daten der Benutzer gewährleistet wird. Das entstehende System soll sich bezüglich der Empfehlungen nicht auf einen bestimmten Bereich oder eine Produktgruppe beschränken, sondern allgemein einsetzbar sein.

Das UniTEC-System besteht im Wesentlichen aus drei Teilen. Der Kern des Systems ist der Empfehlungsdienst, der in der vorliegenden Arbeit dokumentiert wird. Er ist zum einen für die Erstellung und die Verwaltung von Empfehlungen verantwortlich,

zum anderen ermöglicht er die Formulierung von Anfragen nach Empfehlungen und die Beantwortung dieser Anfragen.

Der zweite Teil ist für die Verwaltung von Vertrauenswerten und deren Aktualisierung verantwortlich, wird im Rahmen einer Studienarbeit entwickelt und ist ausführlich in [Aich03] beschrieben.

Der Identitätsmanager verwaltet die Daten der Systembenutzer. Er ist dafür verantwortlich, dass die Benutzer stets die Kontrolle über die Weitergabe ihrer persönlichen Daten haben. Dieser Teil wurde im Rahmen einer weiteren Diplomarbeit entwickelt (siehe [Widm03]).

Im nächsten Kapitel wird auf die Struktur des UniTEC-Systems näher eingegangen.

1.3 Gliederung der Arbeit

Nach der Einführung in die Thematik dieser Arbeit wird im nächsten Kapitel auf das Systemmodell des Rahmenprojekts eingegangen. Hierbei wird dessen Architektur und Funktionsweise erläutert.

Anschließend werden in Kapitel 3 einige bereits existierende Ansätze beschrieben, die sich mit Anonymität in Peer-to-Peer-Netzen befassen. Ferner werden Möglichkeiten zur verteilten Speicherung von Daten in Peer-to-Peer-Umgebungen dargestellt.

Kapitel 3 stellt die Anforderungen an die Kommunikation im UniTEC-System auf und beschreibt anschließend ein neues Verfahren, das diese Anforderungen erfüllt. Ferner werden einige Angriffsszenarien analysiert und verschiedene Abwehrmöglichkeiten vorgestellt.

In Kapitel 4 werden die Anforderungen an den Empfehlungsdienst beschrieben und eine Möglichkeit vorgestellt, wie dieser auf der Grundlage der in Kapitel 3 entworfenen Kommunikationslösung implementiert werden kann.

Kapitel 6 schildert einige Aspekte aus der Realisierung der vorgeschlagenen Lösung. Zunächst werden die Technologien beschrieben, die bei der Realisierung eine wichtige Rolle spielen. Anschließend wird auf die Implementierung der einzelnen Teile des entstandenen Prototyps eingegangen.

Kapitel 7 fasst schließlich die erzielten Ergebnisse zusammen und schlägt einige Erweiterungen und Forschungsaspekte vor.

Kapitel 2

Das Systemmodell von UniTEC

Dieses Kapitel beschreibt die Struktur und Funktionsweise des UniTEC-Systems aus Sicht des zu entwickelnden Empfehlungsdienstes und erörtert dabei einige grundlegende Architekturfragen.

2.1 Systemgrundlagen

Dieser Abschnitt beschreibt grundlegende Elemente des UniTEC-Systems, die für das Verständnis der Arbeit eine große Bedeutung haben.

Basisbegriffe

Zunächst werden einige Grundbegriffe des Systems definiert.

Pseudonym

Die Benutzer des UniTEC-Systems verwenden *Pseudonyme* – etwa fiktive Namen – um ihre persönlichen Daten zu schützen. Sie haben dadurch die Möglichkeit, anonym mit anderen Benutzern zu kommunizieren. Bei Bedarf können sie jedoch auch ihre wahre Identität preisgeben. Näheres zum Thema Pseudonyme, wahre Identität, Anonymität und allgemein zum Identitätsmanagement wird in [Widm03] erläutert.

Erfahrung und Bewertung

In [KiRo03] und [KiPe03] hat der Begriff *Erfahrung* (engl.: *experience*) eine doppelte Bedeutung. Zum einen handelt es sich um den Eindruck, der durch die Analyse des zu empfehlenden Objekts zustande kommt und der für die Erstellung einer Empfehlung notwendig ist. Zum anderen kann sich eine Erfahrung aber auch auf eine Empfehlung beziehen; diese Art von Erfahrung beeinflusst das Vertrauen gegenüber dem Empfehlenden.

In der vorliegenden Arbeit werden für diese zwei Bedeutungen verschiedene Begriffe verwendet. Im Kontext des zu empfehlenden Objekts spricht man hier von einer *Erfahrung*, während eine *Bewertung* sich auf eine Empfehlung bezieht.

Aufgrund eigener Erfahrungen mit dem empfohlenen Objekt kann man zu einer Empfehlung eine Bewertung erstellen. Je nachdem, ob die eigene Erfahrung mit der Erfahrung des Empfehlenden übereinstimmt, wird die Bewertung positiv oder negativ ausfallen.

Empfehlung

Im allgemeinen Sprachgebrauch ist der Begriff *Empfehlung* mit einer positiven Einschätzung des empfohlenen Objekts verbunden. Im Rahmen des Projekts spricht man von einer Empfehlung als eine Beschreibung der Eigenschaften oder des Verhaltens eines Objekts. Diese Beschreibung basiert auf einer Erfahrung, die der Empfehlende mit dem Objekt bereits gemacht hat. Diese Erfahrung kann durchaus auch negativ sein. Empfehlungen sind von Natur aus subjektiv, und deren Bedeutung ist abhängig vom jeweiligen Vertrauen gegenüber dem Empfehlenden im Kontext derjenigen Kategorien, zu denen das empfohlene Objekt gehört.

Prinzipiell handelt es sich auch bei Werbung um Empfehlungen, da diese aber selten objektive Beschreibungen enthalten, ist der damit verbundene Vertrauenswert und dadurch die Bedeutung der Empfehlungen meistens sehr gering.

Kategorie

Das Objekt, zu dem eine Empfehlung erstellt wird, gehört mindestens einer *Kategorie* an. Die Bewertung der Empfehlung wirkt sich auf das Vertrauen gegenüber dem Empfehlenden im Kontext aller Kategorien, zu denen das empfohlene Objekt gehört. Wird das Objekt mehr als einer Kategorie zugeordnet, so kann pro Kategorie eine Gewichtung in Prozent angegeben werden. Diese Gewichtungen beeinflussen die Aktualisierung der Vertrauenswerte (siehe hierzu [Aich03]).

Vertrauen

Innerhalb des UniTEC-Systems stellt das *Vertrauen* eine Beziehung zwischen zwei verschiedenen Pseudonymen im Kontext einer bestimmten Kategorie dar. Das Vertrauen in ein Individuum entspricht der Erwartungshaltung gegenüber dem künftigen Verhalten dieses Individuums basierend auf den Erkenntnissen aus früheren Begegnungen (siehe hierzu [MMH03]). Diese Erwartungshaltung ist subjektiv und daher schwierig zu quantifizieren. UniTEC verfolgt hierbei mehrere verschiedene Ansätze (siehe [Aich03]), die die Bewertungen von Empfehlungen in die Berechnung mit einbeziehen.

Die Vertrauensbeziehung ist asymmetrisch: sei `t(A, B, K)` das Vertrauen des Pseudonyms A gegenüber dem Pseudonym B im Kontext der Kategorie K, dann gilt meistens `t(A, B, K)` $\neq$ `t(B, A, K)`.

Datenmodelle

Die wichtigsten Daten des Systems werden in drei Hauptkategorien unterteilt: Empfehlungen, Expertisedaten und Vertrauensdaten. Jede dieser Kategorien hat ihr eigenes Datenmodell.

Im Rahmen dieser Arbeit versteht man unter Datenmodell nicht nur eine abstrakte Beschreibung der zu speichernden Daten, es stellt auch eine Abstraktion der Art und Weise, wie diese gespeichert werden, dar. Unter dieser Prämisse kann man von einer Speicherung von Daten in einem Modell sprechen.

Das Empfehlungsmodell

In diesem Modell werden Empfehlungen abgelegt, wobei zum einen berücksichtigt werden muss, dass die Struktur der Empfehlungen sehr flexibel ist und vom Benutzer beeinflusst werden kann. Andererseits muss eine effiziente und flexible Suche im Datenbestand möglich sein, zum Beispiel durch Volltextsuche.

Das Modell speichert in erster Linie die Empfehlungen, die lokal erstellt wurden. Zusätzlich enthält es eine begrenzte Anzahl an Empfehlungen, die als Antwort auf eine Anfrage empfangen wurden. Dieses ermöglicht die Beantwortung eingehender Anfragen mit Hilfe von fremden Empfehlungen. Dadurch wird die Verfügbarkeit der Empfehlungsdaten im System und dessen Effizienz erhöht.

Bei manchen Empfehlungen ist es sinnvoll, eine Zugriffskontrolle durchzuführen. Diese Kontrolle wird mittels Zugriffskontrolllisten durchgeführt, die beim Zugriff auf das Empfehlungsmodell geprüft werden.

Weitere Details des Empfehlungsmodells werden in Kapitel 5 behandelt.

Das Vertrauensmodell

Das Vertrauensmodell, auch Trust-Modell genannt, enthält die Vertrauenswerte zwischen je zwei Pseudonymen im Kontext jeder existierenden Kategorie. Diese Werte werden als reelle Zahlen zwischen 0 und 1 gespeichert. Der minimale Vertrauenswert 0 gegenüber einem Pseudonym bedeutet absolutes Misstrauen. Er wird aber auch als Startwert für das Vertrauen verwendet, und zwar im Zusammenhang mit neuen oder unbekannten Pseudonymen. Dadurch wird verhindert, dass Benutzer durch die Erstellung eines neuen Pseudonyms Vertrauen gewinnen können – dieses wäre zum Beispiel bei einem Vertrauensbereich zwischen -1 und 1 und einem Startwert von 0 möglich.

Zusätzlich speichert das Vertrauensmodell die Kategorien in einem gewichteten, gerichteten Graphen. Hierbei stellt jeder Knoten eine Kategorie dar. Die Gewichte der Kanten beschreiben das Ausmaß des gegenseitigen Einflusses zwischen Kategorien bei Veränderungen der Vertrauenswerte. Dieses Ausmaß ist abhängig vom semantischen Abstand zwischen den Kategorien.

Abbildung 2-1 zeigt einen Ausschnitt aus einem Abhängigkeitsgraphen. Die enthaltenen Gewichte sind nur als Beispiele zu interpretieren und stellen keinen Anspruch auf Korrektheit.

Eine Bewertung einer Empfehlung in der Kategorie „Books: Security“ beeinflusst das Vertrauen in dieser Kategorie gegenüber dem Empfehlenden. Sie beeinflusst aber auch andere Kategorien. Zum Beispiel wirkt sich diese Bewertung auf die Kategorie „Books: Conspiracy Theory“ aus, die Änderung am Vertrauenswert ist aber nur halb so groß (Faktor 0,5) wie eine direkte Änderung in dieser Kategorie.

Einzelheiten zum Vertrauensmodell und zu Trust-Update-Algorithmen werden in [Aich03] beschrieben.

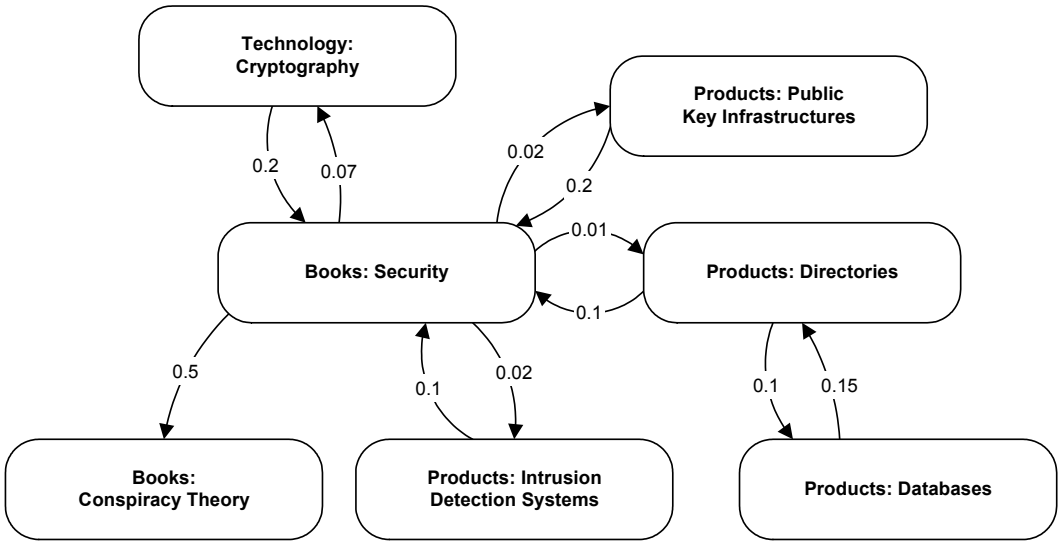


Abbildung 2-1: Kategorien-Abhängigkeitsgraph im Trust-Modell

(Quelle: [KiRo03])

Das Expertisemodell

Dieses Modell speichert Statistikdaten über die Expertise der Pseudonyme. Diese Daten werden im Laufe der Formulierung und Beantwortung von Anfragen aktualisiert.

Die gespeicherten Statistikdaten werden zur Berechnung der Hub- und Authority-Werte verwendet. *Authorities* sind Pseudonyme, deren Expertise im Zusammenhang mit der Beantwortung einer Anfrage besonders relevant ist. Diese Relevanz wird anhand der vom jeweiligen Pseudonym bereits beantworteten Anfragen in einer bestimmten Kategorie berechnet. *Hubs* sind Pseudonyme, die Verbindungen zu vielen Authorities haben. Diese Verbindungen können zum einen direkt sein, sie können aber auch indirekt über weitere Hubs zu Authorities führen.

Die Begriffe Hub und Authority werden allgemein im Bereich des Information Retrieval und speziell im Zusammenhang mit der Beschreibung der Architektur und der Funktionsweise von Suchmaschinen verwendet (siehe hierzu [Klei98]).

Weitere Details zum Expertisemodell und zur Berechnung der Hub- und Authority-Werte im UniTEC-System sind in [Widm03] enthalten.

2.2 Architektur

Das UniTEC-System strebt eine verteilte Architektur an. Der vorliegende Abschnitt beschreibt zunächst die Gründe dieser Entscheidung, indem er auf die Nachteile eines zentralisierten Modells eingeht. Anschließend wird die Architektur des Systems beschrieben.

Nachteile eines zentralisierten Modells

Im Internet haben sich in den letzten Jahren Meinungsportale[1] etabliert. Diese webbasierten Systeme können von beliebigen Benutzern verwendet werden, um ihre Erfahrungen in den verschiedensten Kategorien als Empfehlungen zu veröffentlichen. Der Benutzer registriert sich am System, indem er seine persönlichen Daten und ein Pseudonym seiner Wahl eingibt. Empfehlungen werden unter dem gewählten Pseudonym erstellt.

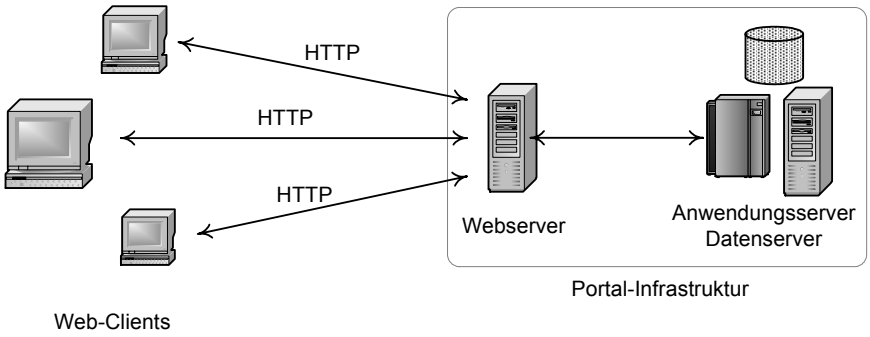


Abbildung 2-2: Beispiel für eine zentralisierte Architektur

Abbildung 2-2 enthält ein Beispiel für eine Architektur eines typischen Meinungsportals. Die Daten sind auf mehreren Datenservern gespeichert, um Redundanz und Effizienz zu gewährleisten. Die Anwendungsserver enthalten die Anwendungslogik. Da Meinungsportale keine aufwendigen Berechnungen benötigen, ist die Anwendungsschicht nicht besonders komplex. Für die Präsentation der Daten wird ein Webserver verwendet, der die Schnittstelle zu den Benutzern des Systems darstellt.

Ein derartiges System ist relativ einfach zu realisieren. Die Hauptproblematik des Systems ist jedoch gerade mit dessen Einfachheit verbunden und speziell mit der Tatsache, dass alle Empfehlungen zentral beim Betreiber des Portals gespeichert werden.

Die Kosten, die durch den Betrieb und die Wartung eines solchen Systems entstehen, sind nicht zu vernachlässigen. Sie könnten theoretisch durch die Einführung einer Nutzungsgebühr gedeckt werden, dieses würde die Akzeptanz des Systems jedoch stark beeinträchtigen. Daher werden sie größtenteils durch Werbeeinnahmen gedeckt, aber teilweise auch durch Provisionen, die der Betreiber von Online-Shops erhält. Meist werden dem Nutzer des Systems im Zusammenhang mit einer Empfehlung ein oder mehrere Links zu Online-Shops angeboten, bei denen das empfohlene Objekt erworben werden kann. Diese Einnahmequelle beeinflusst die Neutralität des Betreibers. Da die Empfehlungen bei diesem gespeichert sind, ist es ihm möglich, Empfehlungen zu ändern oder neue zu erstellen, etwa um den Partner-Shops mehrere Kunden zu liefern und damit die Provisionseinnahmen zu erhöhen. Integritätsprüfungen, die zum Beispiel mit Hilfe von digitalen Signaturen durchgeführt werden könnten, werden von keinem der bekannten Portale ermöglicht. Der Nutzen des Portals hängt hiermit stark von der Vertrauenswürdigkeit des Betreibers ab.

[1] Beispiele: http://www.ciao.com, http://www.dooyoo.de, http://www.epinions.com

Ein potentielles Problem stellt auch eine weitere Einnahmenquelle der Portalbetreiber dar. Die gespeicherten Daten sind für die Erstellung verschiedener Statistiken gut geeignet. Sofern diese nicht auf einzelne Benutzer bezogen sind und nur zur Marktforschung verwendet werden, schaden sie niemandem. Es wäre jedoch auch denkbar, dass der Betreiber Verhaltensprofile der Benutzer anhand der Empfehlungen erstellen und diese zusammen mit den persönlichen Daten der Benutzer an interessierte Firmen verkaufen könnte. Die persönlichen Daten werden meistens bei der Registrierung eingetragen, etwa die E-Mail-Adresse oder sogar die Postanschrift. Als Ergebnis würde der Benutzer unerwünschte Werbepost oder lästige Spam-Mail erhalten.

Das UniTEC-Modell

Wie im letzten Abschnitt erwähnt wurde, hat ein zentralisiertes Modell unter anderem den Nachteil, dass ein Großteil der Kosten, die durch die Verwaltung der Infrastruktur entstehen, durch die Zusammenarbeit mit kommerziellen Partnern gedeckt wird. Diese Zusammenarbeit beeinflusst die Neutralität des Betreibers und damit das Vertrauen in denselben. Aus diesem Grund wird im UniTEC-Projekt trotz einer aufwendigeren Implementierung eine verteilte Architektur angestrebt.

Agent

Der Datenschutz hat im System höchste Priorität. Aus diesem Grund ist besonders darauf zu achten, dass es keine Möglichkeit gibt, ohne den Willen eines Benutzers dessen wahre Identität herauszufinden. Um das Risiko einer Aufdeckung der Identität zu minimieren, ist es ratsam, dass ein Benutzer nicht nur ein einziges Pseudonym verwendet, sondern seine Expertise auf mehrere Pseudonyme verteilt. Jedes Pseudonym eines Benutzers ist für eine bestimmte Kategoriemenge zuständig. Diese Mengen müssen nicht unbedingt paarweise disjunkt sein. Die hiermit verbundene Problematik wird in [Widm03] ausgeführt.

Die Software, die der UniTEC-Benutzer verwendet, wird als *Agent* bezeichnet. Innerhalb eines Agenten können ein oder mehrere Pseudonyme aktiv sein. Wenn ein Pseudonym aktiv ist, kann der Benutzer im Kontext dieses Pseudonyms eine Anfrage formulieren, um nach Empfehlungen in einer bestimmten Kategorie zu suchen. Der Agent kann im Kontext eines aktiven Pseudonyms eine Anfrage beantworten, sofern dieses Pseudonym für diejenige Kategorie zuständig ist, die in der Anfrage enthalten ist.

Peer-to-Peer-Netz

Innerhalb des Agenten sind die Daten in den Datenmodellen gespeichert, die bereits im Abschnitt 2.1 beschrieben wurden: Empfehlungs-, Vertrauens- und Expertisemodell. Diese Daten werden lokal auf dem jeweiligen Rechner des Benutzers gehalten.

Um Anfragen absenden und Antworten auf diese empfangen zu können, müssen die Agenten miteinander kommunizieren können. Da eine verteilte Architektur angestrebt wird, werden die Agenten durch ein Peer-to-Peer-Netz miteinander verbunden.

Anonymität auf Netzwerkebene

Wie bereits erwähnt, ist das Risiko der Aufdeckung der Benutzeridentität so gering wie möglich zu halten. Üblicherweise ist die IP-Adresse mit einer Identität verbunden. Diese Zuordnung wird meistens vom Netzbetreiber zur Abrechnung und zur Vermeidung von Missbrauch gespeichert, in den meisten Ländern wird jedoch ein richterlicher Beschluss für die Herausgabe persönlicher Daten anhand der IP-Adresse benötigt.

Teile dieser Identität können aber unter Umständen auch auf andere Art und Weise entdeckt werden, zum Beispiel durch eine Abfrage einer Whois-Datenbank[2]. Hierdurch kann der Betreiber des Netzes herausgefunden werden, in dem sich der Benutzer befindet. Falls der Benutzer seinen Agenten aus einem kleineren Netz, etwa dem einer kleinen Firma verwendet, so kann durch gezielte Anfragen beim Betreiber möglicherweise die wahre Identität des Benutzers entdeckt werden.

Anhand der IP-Adresse können unter Umständen zwei Pseudonyme, die zum gleichen Benutzer gehören, einander zugeordnet werden, da sie die gleiche Adresse besitzen.

Um das Risiko einer Identitätsaufdeckung zu minimieren, darf die Zuordnung zwischen einem Pseudonym und der Adresse des Agenten, in dessen Rahmen das Pseudonym aktiv ist, zu keinem Zeitpunkt bekannt sein. Die ausgetauschten Daten sollten nicht mit einem Pseudonym in Verbindung gebracht werden können (siehe hierzu auch [Widm03]). Letzteres wird durch eine Verschlüsselung der Daten ermöglicht.

2.3 Funktionsweise

Dieser Abschnitt beschreibt die Funktionsweise des UniTEC-Systems primär aus Sicht des Empfehlungsdienstes. Weitere Details sind in Kapitel 5 enthalten.

Anfragen

Empfehlungen, die im Empfehlungsmodell anderer Agenten gespeichert sind, können mittels *Anfragen* abgerufen werden. Anfragen werden im Kontext einer bestimmten Kategorie und in Form einer Volltextsuche formuliert. Die Anfrage wird mit einem *Hop-Counter* versehen, bevor sie an die Nachbarschaft (siehe Abschnitt *Nachbarschaft*) versendet wird. Jedes Mitglied der Nachbarschaft leitet die Anfrage weiter an seine eigene Nachbarschaft. Durch den Hop-Counter kann zum einen die Anzahl der Pseudonyme begrenzt werden, die von der Anfrage erreicht werden, und zum anderen wird dadurch sichergestellt, dass die Anfrage nicht unendlich oft weitergeschickt wird. Vor jeder Weiterleitung wird ein Eintrag zur Vertrauenskette hinzugefügt.

[2] Zum Beispiel http://www.ripe.net/db

Nachbarschaft

Nachdem die Anfrage formuliert wurde, wird sie an eine Menge von Pseudonymen bestimmter Größe und Zusammensetzung geschickt, die dem Anfragenden bekannt sind. Diese Menge wird *Nachbarschaft* (engl. *neighborhood*) genannt.

Sowohl die Größe als auch die Zusammensetzung der Nachbarschaft beeinflusst die Ergebnisse der Anfrage. Die Nachbarschaftsgröße beeinflusst, ähnlich dem Hop-Counter, die Anzahl der Pseudonyme, die von der Anfrage erreicht werden und möglicherweise die Anfrage beantworten können.

Die Nachbarschaft setzt sich zusammen aus vier Kategorien von Pseudonymen: vertrauenswürdige Pseudonyme, Authorities, Hubs und zufällige Pseudonyme. Der Anteil jeder dieser Kategorien kann durch den Benutzer bestimmt werden. Vertrauenswürdige Pseudonyme sind diejenigen Pseudonyme, denen im Vertrauensmodell die höchsten Vertrauenswerte im Kontext der Anfragekategorie entsprechen. Authorities und Hubs werden anhand des Authority- bzw. Hub-Ratings aus dem Expertisemodell bestimmt. Die vierte Kategorie, die zufälligen Pseudonyme, trägt dazu bei, dass auch neue Pseudonyme eine Chance zur Beantwortung von Anfragen bekommen. Dadurch werden sie in das Vertrauens- und Expertisemodell aufgenommen und haben dadurch die Möglichkeit, bei der Bestimmung einer neuen Nachbarschaft in eine der drei anderen Kategorien zu kommen.

Vertrauensketten

Bevor eine Anfrage an ein Pseudonym aus der Nachbarschaft geschickt wird, wird der zu diesem Pseudonym gehörende Vertrauenswert in der entsprechenden Kategorie zusammen mit der ID des Pseudonyms der Anfrage hinzugefügt. Da die Anfrage rekursiv weitergesendet wird, entsteht eine Reihe von ID-Wert-Paaren, die *Vertrauenskette* (engl. *trust chain*) genannt wird.

Bevor eine Antwort an den Anfragenden gesendet wird, wird die Vertrauenskette in die Antwortnachricht kopiert. Der Agent des Anfragenden interpretiert die Vertrauenskette jeder Antwort und berechnet daraus einen Vertrauenswert zwischen dem Anfragenden und dem Antwortenden. Der Wert wird bei der Anzeige der eintreffenden Antworten verwendet. Die in den Antworten enthaltenen Empfehlungen werden zum Beispiel nach diesem Wert absteigend sortiert aufgelistet.

Um Manipulationen an der Vertrauenskette zu verhindern, wird diese nach jedem Hinzufügen eines neuen Ketteneintrags signiert. Die entstehenden Signaturen werden nach Empfang der Antwort geprüft. Falls nicht alle gültig sind, wird die Antwort verworfen. Details hierzu werden in Kapitel 5 beschrieben.

Reaktion auf Anfragen

Erhält ein Pseudonym eine eingehende Anfrage, so wird zunächst das Empfehlungsmodell durchsucht. Werden eine oder mehrere passende Empfehlungen gefunden, so wird eine Antwortnachricht an den Anfragenden gesendet, die die gefundenen Empfehlungen und die Vertrauenskette aus der entsprechenden Anfrage enthält. Anschließend wird geprüft, ob der Hop-Counter den Wert 0 erreicht hat. Ist dieses

nicht der Fall, so wird die Anfrage an die Nachbarschaft weitergesendet. Ansonsten wird die Anfrage verworfen.

Aktualisierung der Modelle

Nach dem Eingang einer Antwort wird anhand der in der Vertrauenskette enthaltenen Pseudonyme das Expertise-Modell aktualisiert. Für Pseudonyme, die die Anfrage erfolgreich weitergeleitet haben, wird der Hub-Wert erhöht, während für das Pseudonym, das die Antwort versendet hat, der Authority-Wert inkrementiert wird.

Empfehlungen können bewertet werden, nachdem eigene Erfahrungen mit dem empfohlenen Objekt gemacht wurden. Anhand der Bewertung, die eine Empfehlung erhält, wird das Vertrauen gegenüber dem Ersteller der Empfehlung im Kontext der Kategorie, zu der diese gehört, aktualisiert und im Vertrauensmodell gespeichert. Algorithmen, die hierzu verwendet werden, werden im innerhalb des UniTEC-Projekts *Trust-Update-Algorithmen* genannt. Die Algorithmen, die im System eingesetzt werden, werden in [Aich03] detailliert beschrieben.

Kapitel 3

Verwandte Ansätze

Dieses Kapitel beschreibt bestehende Lösungsansätze im Bereich der Anonymität und der verteilten Datenspeicherung in Peer-to-Peer-Netzen. Die vorliegende Arbeit basiert auf einigen dieser Ansätze.

3.1 Grundlagen zur Anonymität

Zunächst werden bereits existierende Möglichkeiten zur Realisierung anonymer Kommunikation analysiert.

Flooding

Flooding basiert auf dem gleichnamigen Routing-Verfahren und stellt eine der einfachsten Möglichkeiten zur anonymen Kommunikation dar.

Beschreibung

Der Sender verschlüsselt das Datenpaket mit dem öffentlichen Schlüssel des Empfängers und schickt es an alle ihm bekannten Knoten. Diese versuchen das Paket zu entschlüsseln und leiten es anschließend weiter an die jeweilig bekannten Knoten. Damit das Paket nicht endlos im Netz weitergesendet wird, enthält es einen Hop-Counter, der vor jedem Sendevorgang dekrementiert wird. Eine Paket-ID sorgt dafür, dass Pakete beim zweiten Eintreffen ignoriert und nicht mehrfach weitergegeben werden.

Wenn das Paket den echten Empfänger erreicht, so kann dieser es als einziger entschlüsseln, ohne dass andere Netzteilnehmer erfahren, dass er der Empfänger ist. Auch der Sender kann sich eines gewissen Anonymitätsgrades erfreuen, da jeder aktive Knoten im Netz das Paket sendet. Sobald man eine detailliertere Verkehrsanalyse durchführt, wird allerdings erkennbar, dass der Sender des Pakets im Gegensatz zu anderen Knoten das Paket versendet hat, ohne es vorher empfangen zu haben. Falls der Empfänger an dieser Analyse beteiligt ist, kann er das Pseudonym einem Knoten zuordnen und unter Umständen dessen Identität aufdecken.

Bewertung

Von Vorteil ist der geringe Implementierungsaufwand, der mit der Realisierung dieses Verfahrens verbunden ist. Dessen wesentlicher Nachteil ist jedoch der sehr hohe Datenverkehr, der durch den Versand eines einzigen Pakets entsteht. Hinzu kommt die Tatsache, dass jeder Knoten ein potentieller Empfänger ist und daher versuchen

muss, jedes eintreffende Paket zu entschlüsseln. Aus diesem Grund ist die Skalierbarkeit des Verfahrens unbefriedigend.

DC-Netze

Der Name dieses Verfahrens ist zum einen auf das Problem der *dining cryptographers* zurückzuführen (siehe hierzu [Chau88]), zum anderen auf die Tatsache, dass es von David Chaum 1988 entwickelt wurde.

Beschreibung

Das Verfahren basiert auf dem Prinzip des überlagernden Sendens, bei dem zufällige Schlüssel, die zwischen jeweils zwei Netzteilnehmern ausgetauscht wurden, der Nachricht bitweise (z.B. per XOR-Verknüpfung) überlagert und anschließend gesendet werden. Möchte ein Teilnehmer nichts senden, so überlagert er den Schlüssel einer leeren Nachricht und versendet das Ergebnis. Beim Empfänger heben sich je zwei Überlagerungen auf und dieser kann anschließend die Nachricht lesen. Hierbei bleibt der Sender anonym. In [Pfit97] wird die Senderanonymität bewiesen und es wird beschrieben, wie man mittels der Knack-Schnapp-Verteilung auch die Empfängeranonymität gewährleisten kann.

Bewertung

Obwohl das Verfahren vor geraumer Zeit entwickelt wurde, ist zurzeit noch keine effiziente und skalierbare Implementierung eines DC-Netzes gelungen. In [Stam03] wird innerhalb des *GnuDCN*-Projekts bestätigt, „dass robuste DC-Netze einen enormen Nachrichten- und Kommunikationsaufwand in der Größenordnung $O(n^5)$ verursachen“ und damit leider eine allgemeine Praktikabilität nicht gegeben ist.

Mix-Netze und Onion Routing

Onion Routing verfolgt einen anderen Ansatz als die bisher beschriebenen Verfahren. Es basiert auf dem Konzept der *Mix-Netze*, das von David Chaum entwickelt und erstmals in [Chau81] im Zusammenhang mit dem Versenden anonymer Nachrichten per E-Mail erwähnt wurde. Onion Routing (siehe [GRS99]) verwendet das gleiche Prinzip für die Echtzeitkommunikation und ist hiermit für verschiedene Netzwerkprotokolle geeignet, etwa für das Surfen im WWW.

Beschreibung

Nachrichten gleicher Größe werden über mehrere Zwischenstationen gesendet, mit dem Ziel, die Kommunikationsbeziehung zwischen Sender und Empfänger geheim zu halten, also um die Erkenntnis zu verhindern, dass der Sender eine Nachricht an den Empfänger schickt. Diese Zwischenstationen werden *Mixe* genannt, da zu deren Aufgaben neben dem Umcodieren auch die Änderung der Reihenfolge der Nachrichten (das „Mixen“) mit dem Zweck der Verschleierung zeitlicher Zusammenhänge gehört. Jeder Mix verfügt über ein Schlüsselpaar, das für die Umcodierung der Nachrichten verwendet wird.

Der Sender wählt eine Folge von mehreren Mixen aus. Diese Folge nennt Chaum eine *Mix-Kaskade*. Sie stellt den Weg dar, dem die Nachricht bis zum Empfänger folgt. Danach präpariert der Sender die Nachricht so, dass als Ergebnis ein Paket mit

mehreren Schichten entsteht. Jede Schicht enthält zum einen die Adresse des nächsten Hops auf dem Weg vom Sender zum Empfänger – entweder ein Mix oder der Empfänger der Nachricht – im Klartext, und zum anderen die Daten, die für den nächsten Hop bestimmt sind, verschlüsselt mit dessen öffentlichen Schlüssel.

Das Paket entsteht folgendermaßen:

1. Die Nachricht wird mit dem öffentlichen Schlüssel des Empfängers verschlüsselt

2. Zum entstandenen Paket wird die Adresse des Empfängers hinzugefügt

Dann werden diese Schritte für den letzten Mix wiederholt, anschließend für den vorletzten etc. bis man am Anfang der Mix-Kaskade angekommen ist. Das dabei entstandene Paket wird dann an den ersten Mix versendet, dessen Adresse in der äußersten Hülle des Pakets im Klartext verfügbar ist. Der Mix behandelt das Paket folgendermaßen:

1. Das empfangene Paket wird mit dem geheimen Schlüssel des Mixes entschlüsselt

2. Die Adresse des nächsten Hops, d.h. des nächsten Knotens der Route, wird ausgelesen und entfernt

3. Das entstehende Paket wird in einer Warteschlange gespeichert, zusammen mit der Adresse des nächsten Hops

4. Falls die Warteschlange eine festgelegte Länge innerhalb eines bestimmten Intervalls nicht erreicht, so wird sie mit bedeutungslosen (so genannten Dummy-) Nachrichten gefüllt

5. Alle Nachrichten werden in einem Schub an den jeweils nächsten Hop gesendet, und zwar in zufälliger Reihenfolge, um die Erkennung zeitlicher Zusammenhänge zwischen ein- und ausgehenden Paketen auszuschließen

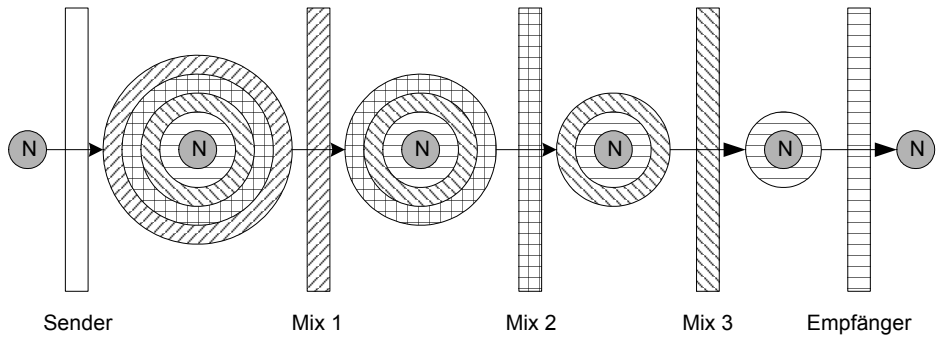


Abbildung 3-1: Nachrichtentransport entlang einer Mix-Kaskade

Abbildung 3-1 zeigt, wie das Paket entlang der Mix-Kaskade gesendet wird. Die Schraffuren der verschiedenen Schichten des Pakets sind so gewählt, dass sie identisch mit denen der Knoten sind, an denen diese Schichten entschlüsselt werden können.

Bewertung

Die sukzessive Verschlüsselung bewirkt, dass jeder Knoten in der Kaskade höchstens seinen Vorgänger und seinen Nachfolger kennt. Falls der Angreifer nicht im Besitz aller Mixe der Kaskade ist, so kann er nur dann herausfinden, dass der Sender mit dem Empfänger kommuniziert, wenn er den gesamten Weg der Nachricht nachvollziehen kann. Eventuell könnte ihm dieses durch die Analyse der Paketgröße beim Ein- und Austritt an den Mixen gelingen. Diesem kann man vorbeugen, indem man verhindert, dass das Paket nach jedem Mix durch die Entfernung einer Schicht kleiner wird. Eine Methode hierzu wird in [Chau81] vorgeschlagen, die auf einer Blockstruktur des Pakets basiert.

Ferner kann in einem „ruhigen" Netz, in dem nur selten Nachrichten versendet werden, die Nachricht leicht verfolgt werden: der erste Knoten, der eine Nachricht sendet, ist der Sender, während der letzte Knoten, der eine Nachricht empfängt, als der Empfänger erkannt wird. Diese Beobachtung kann durch die Versendung von Dummy-Nachrichten durch alle beteiligten Knoten in zufälligen Abständen erschwert werden.

[Chau81] beschreibt ferner eine Möglichkeit zur Erstellung einer einmaligen anonymen Rückadresse, die zusammen mit der Nachricht mitgeschickt und zur Versendung einer Antwort, wie zum Beispiel einer Bestätigungsnachricht, durch den Empfänger verwendet werden kann. Da die Rückadresse immer nur dann generiert wird, wenn der Sender eine Nachricht für den Empfänger erstellt, kann der Empfänger vor Ankunft dieser Nachricht den Sender nicht kontaktieren, ohne Kenntnis seiner Adresse. Zur Erstellung und Versendung einer Nachricht benötigt der Sender die Adresse des Empfängers. Hierin liegt der wesentliche Nachteil des Verfahrens – es bietet keine Empfängeranonymität.

3.2 Projekte im Bereich Anonymität

Auf dem Gebiet der anonymen Peer-to-Peer-Systeme wurden in den letzten Jahren bereits Fortschritte verzeichnet. Einige der bekannteren Projekte, deren Zielsetzung in diesem oder in verwandten Bereichen liegt, werden in diesem Abschnitt kurz beschrieben.

Crowds

Das Ziel des Crowds-Systems (siehe [ReRu98]), das 1998 bei AT&T entwickelt wurde, ist es, anonyme Verbindungen zu Webservern aufzubauen. Der Benutzer ist Teil einer Gruppe, deren Teilnehmer Instanzen eines Software-Agenten namens Jondo (von „John Doe": unbekannte, anonyme Person) verwenden.

Der lokale Jondo leitet eine Anfrage, die für einen bestimmten Webserver bestimmt ist, an einen zufälligen Jondo der gleichen Gruppe weiter. Dieser versendet die An-

frage entweder an den Server oder an einen weiteren Jondo – die Entscheidung wird zufällig anhand einer einstellbaren Wahrscheinlichkeit getroffen. Die Antwort des Webservers folgt der gleichen Route in umgekehrter Richtung. Innerhalb der Gruppe werden die Nachrichten verschlüsselt versendet.

Gegenüber dem Webserver kann jeder Benutzer eines Jondo abstreiten, dass er die Anfrage formuliert hat, indem er behauptet, er hätte sie lediglich weitergeleitet. Der Anonymitätsgrad der Benutzer ist hiermit abhängig von der Teilnehmeranzahl der Gruppe.

Dieses Verfahren bietet keinen Schutz gegenüber Angreifern, die den gesamten Verkehr einer Gruppe beobachten können, auch bietet es keine Empfängeranonymität, da die Adresse des angesprochenen Servers bekannt sein muss.

Freenet

Während beim Crowds-System die Trennung zwischen Clients und Servern deutlich zu erkennen ist, wird diese Trennung bei Freenet (siehe [CSWH01]) aufgehoben. Das Ziel des Freenet-Projekts ist es, die anonyme, zensurresistente Veröffentlichung von Daten durch deren dezentrale Speicherung in einem Peer-to-Peer-Netz zu ermöglichen.

Die Speicherorte der veröffentlichten Dateneinheiten werden anhand ihrer IDs bestimmt. Dateneinheiten mit ähnlichen IDs werden nahe aneinander gespeichert: sowohl Einfüge- als auch Anfragenachrichten werden an Knoten gesendet, von denen bekannt ist, dass sie bereits Daten mit ähnlichen IDs speichern. Diese leiten sie nach demselben Kriterium weiter. Ein hop-to-live-Wert bestimmt die maximale Ausbreitung dieser Nachrichten. Sowohl das Einfügen als auch eine erfolgreiche Anfrage hat als Folge die Replizierung der Daten auf diejenigen Knoten, die die entsprechenden Nachrichten weitergeleitet haben.

Da die ID durch die Berechnung eines Hashwerts über den Inhalt der Daten oder deren Beschreibung entsteht, und die Daten verschlüsselt sind, können die Benutzer keinen Einfluss darauf nehmen, welche Inhalte auf deren Rechner gespeichert werden, sie haben in der Regel auch keine genaue Kenntnis darüber, wo welche Inhalte gespeichert werden.

Wurde ein Dokument einmal veröffentlicht, so kann man dieses nicht mehr aus dem System entfernen. Die Benutzer können jedoch den reservierten Platz für die Speicherung von Daten beschränkten. Bei Erreichen der eingestellten oberen Grenze werden die Daten gelöscht, die weniger beliebt sind, also in letzter Zeit nicht angefragt wurden (Least Recently Used-Strategie). Dieses führt dazu, dass unbeliebte oder veraltete Daten nach und nach aus dem System verschwinden.

Die Anonymität von Freenet kann durch den Einsatz von Mix-Netzen zusätzlich erhöht werden. Das System ist für die anonyme Veröffentlichung von Daten und zu deren dezentralen Speicherung geeignet, jedoch nicht zur Punkt-zu-Punkt-Kommunikation zwischen zwei Benutzern.

Free Haven

Free Haven (siehe [Oram01], Kapitel 12) verfolgt das gleiche Ziel wie Freenet, nämlich die anonyme Veröffentlichung von Daten in einem Peer-to-Peer-System. Ähnlich dem Freenet-System ist auch Free Haven für die Anonymisierung einer direkten Kommunikation zwischen zwei Benutzern nicht geeignet.

Im Gegensatz zu Freenet basiert bei Free Haven die Kommunikation auf der Versendung von E-Mails; zur Anonymisierung wird ein Mix-Netz verwendet. Zusätzlich wird das System durch ein Reputationssystem ergänzt, das das Vertrauen gegenüber den speichernden Knoten verwaltet. Die Anfragen basieren auf Flooding, hierdurch leidet die Skalierbarkeit des Systems.

Tarzan

Tarzan (siehe [FrMo02]) realisiert eine anonymisierende Transportschicht basierend auf einem Peer-to-Peer-Netz. Diese Schicht setzt auf dem IP-Protokoll auf und verwendet Mixe zur Realisierung der Unverkettbarkeit zwischen je zwei Kommunikationspartnern. Die Mixe werden zufällig aus einer Menge von Peers ausgewählt. Am Ende der jeweiligen Mix-Kaskade befindet sich der Network Address Translator. Dieser ändert die Quelladresse des IP-Pakets und kommuniziert direkt mit dem Empfänger. Hierdurch wird ein Tunnel zwischen Sender und Empfänger aufgebaut, der auch durch konventionelle Software genutzt werden kann. Tarzan ist jedoch nicht für gleichzeitige Sender- und Empfängeranonymität geeignet.

3.3 Verteilte Hashtabellen

In der vorliegenden Arbeit wird ein Ansatz zur Realisierung anonymer Kommunikation verfolgt, der die Speicherung von Daten der Kommunikationspartner voraussetzt. Diese können zum einen zentral gespeichert werden, sie können jedoch auch verteilt im Netz abgelegt werden.

Verteilte Datenspeicherung kann in Peer-to-Peer-Netzen mit Hilfe spezieller Datenstrukturen, die in der Literatur als *verteilte Hashtabellen*, *distributed hashtables* oder kurz *DHTs* bezeichnet werden, effizient implementiert werden.

Der vorliegende Abschnitt beschreibt zunächst die Grundlagen dieser Datenstrukturen, um anschließend auf einen der bekanntesten Ansätze zur Implementierung einer verteilten Hashtabelle einzugehen.

DHT-Grundlagen

Eine verteilte Hashtabelle hat die gleiche Aufgabe wie eine klassische Hashtabelle: sie speichert Schlüssel-Wert-Paare und ermöglicht das effiziente Nachschlagen eines Werts anhand des dazugehörigen Schlüssels.

Verteilte Hashtabellen implementieren eine *Verteilfunktion*

$$d : I \rightarrow A$$

wobei I die Menge aller IDs darstellt und A die Agenten im Netz enthält. Sie schreibt den Speicherort eines Schlüssel-Wert-Paares vor, indem sie anhand einer ID den Agenten berechnet, der für die Speicherung dieses Paares verantwortlich ist. Diese Abbildung ist nicht injektiv (d.h. dass zu jedem Bild der Funktion nicht notwendigerweise nur ein Urbild existiert), da mehrere Schlüssel-Wert-Paare auf einem Agenten gespeichert werden können. Sie ist auch nicht surjektiv (d.h. dass nicht unbedingt jedes Element der Zielmenge ein Bild ist), da es durchaus Agenten geben kann, die für keine Daten verantwortlich sind. Nichtsdestotrotz ist man bei der Implementierung von DHTs bemüht, die Daten so gleichmäßig wie nur möglich auf die Agenten zu verteilen, da hierdurch die Wahrscheinlichkeit sinkt, dass durch einzelne Ausfälle die Verfügbarkeit der Daten stark beeinflusst wird.

Die meisten in der Literatur dokumentierten DHT-Implementierungen, wie etwa Chord (siehe [SMK+01], [DKK+01]), Pastry (siehe [RoDr01]) oder Tapestry (siehe [ZKJ01]), bieten eine hohe Effizienz sowohl beim Speichern als auch beim Nachschlagen der gespeicherten Werte. Diese Operationen sind mit einem logarithmischen Aufwand verbunden.

Chord

Die meisten verteilten Hashtabellen weisen ähnliche Eigenschaften bezüglich der Effizienz auf. Chord wird in der vorliegenden Arbeit verwendet, da es zum einen in mehreren Arbeiten hervorragend dokumentiert ist und sich außerdem in diversen Projekten als zuverlässig erwiesen hat. Das CFS-Projekt (Chord File System) hat ein verteiltes, redundantes und fehlertolerantes Dateisystem auf der Grundlage von Chord entwickelt, dessen Eigenschaften in [DKK+01] beschrieben sind.

Bei Chord bilden die teilnehmenden Knoten einen logischen Ring. Hierzu erhält jeder Knoten eine eindeutige, 160 Bit lange ID, anhand derer der Knoten seinen Platz im Ring erhält. Diese ID ergibt sich durch Hashing der IP-Adresse mit Hilfe der kryptographischen Hashfunktion SHA-1 (siehe Abbildung 3-2). Da diese Funktion die Eingangswerte gleichverteilt auf den Bereich $[0; 2^{160}]$ abbildet, werden die Knoten gleichmäßig auf den Ring verteilt.

Tabelle 3-1 und Abbildung 3-2 enthalten acht Knoten, deren IP-Adressen, IDs und die jeweilige Position im Chord-Ring. Aus Platzgründen werden in den Abbildungen nur die ersten vier Stellen der ID angegeben.

Das Konzept des Nachfolgers ist bei Chord von essenzieller Bedeutung, da es zur Implementierung der Verteilfunktion verwendet wird.

Die Dateneinheiten, die im Chord-Ring gespeichert werden, erhalten auch IDs aus demselben 160 Bit-Bereich (zum Beispiel durch Hashing des Inhalts der Daten). Verantwortlich für die Speicherung einer Dateneinheit mit der ID i ist derjenige Knoten, dessen ID auf i folgt (der Nachfolger von i, siehe Abbildung 3-3).

#	IP-Adresse	ID (SHA-1-Hashwert der IP-Adresse)
1	192.168.1.2	47b45a2710f977f44be86430c652195584de7d6d
2	10.1.1.2	5d24799200fda73b5270bf5a5baa4d666557b433
3	131.188.3.4	6ee3de708483d071c4f2eab0de1276b5398f5f80
4	10.1.1.1	81e0fb1230266b5f338346513d6adf506decda08
5	192.168.1.1	88dc1c83e5e3da8094423005d240b470c61e9a11
6	192.168.1.3	910734f0fd2aaf43dd5cc11b0377f8787115ddf9
7	172.16.1.2	cfc69d767feee8113eacf81102d181fb3d39bba0
8	172.16.1.1	da168c3e486b5e9618ef7aaa4b6cd14bb3217ae4

Tabelle 3-1: Knoten-IDs in einem Chord-Ring

(Quelle: [Gern02])

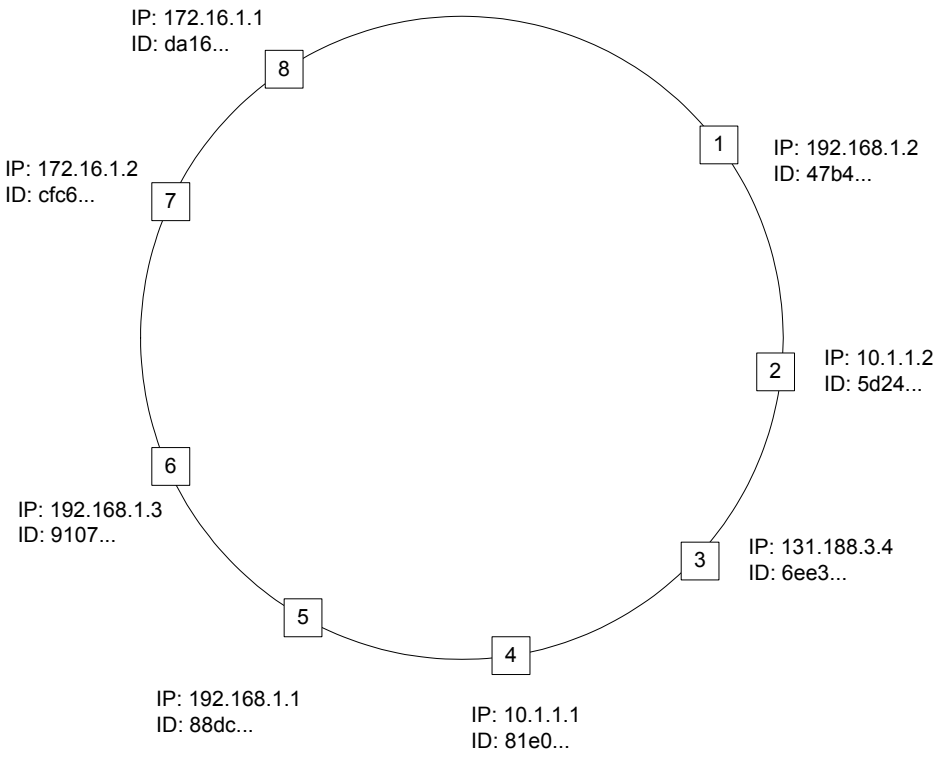


Abbildung 3-2: Anordnung der Knoten aus Tabelle 3-1 im Chord-Ring

Im Ring wird modular gerechnet: bei einem Überlauf der ID erfolgt ein erneuter Durchlauf des Rings. Das erklärt in Abbildung 3-3 zum Beispiel, warum die Dateneinheit f auf dem Knoten 1 gespeichert wird, obwohl die ID von f größer als die des Knotens 1 ist.

Jeder Knoten verwaltet eine Liste von Nachfolgern der Länge r. Der Chord-Ring aus Abbildung 3-3 enthält Knoten, die lediglich die Adresse des ersten Nachfolgers speichern (r = 1).

Zusätzlich speichert jeder Knoten seinen Vorgänger. Aus Platzgründen wird dieses jedoch auf den Abbildungen nicht angezeigt.

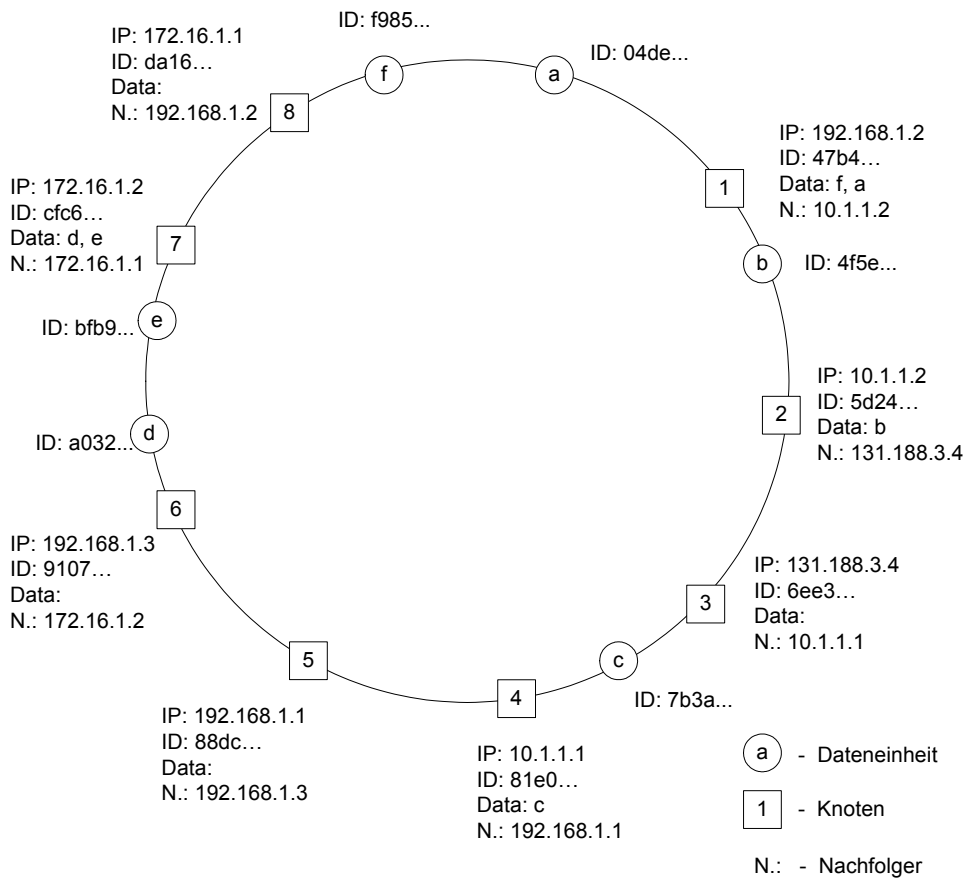


Abbildung 3-3: Datenspeicherung im Chord-Ring

Möchte zum Beispiel Knoten 7 den Speicherort der Dateneinheit c herausfinden, so befragt er nacheinander seinen Nachfolger, dann dessen Nachfolger usw., bis er zu einem Knoten gelangt, dessen ID gerade größer als die von c ist, also Knoten 4. Diese Suche kann sowohl beim Lesen als auch beim Speichern einer Dateneinheit angewendet werden.

Dieses Verfahren führt zum Ziel, hat jedoch den Nachteil, dass sowohl Lookup als auch Speicherung einen Aufwand der Größenordnung $O(n)$ voraussetzen, da man

sich bei beiden Operationen in linearen Schritten vom initiierenden Knoten entfernt. Bei n handelt es sich um die Anzahl der Knoten im Ring.

Die Leistungsfähigkeit des Verfahrens kann durch die Verwaltung einer *Fingertabelle* verbessert werden. Diese enthält eine Liste derjenigen Knoten, die sich in exponentiell wachsenden Abständen vom Initiatorknoten befinden. Die Nutzung dieser Tabelle gewährleistet eine Halbierung des Abstandes zum gesuchten Knoten nach jedem Schritt und damit logarithmischen Suchaufwand. Details hierzu, inklusive Anleitungen zur Implementierung in Pseudocode kann man [SMK+01] entnehmen.

Der Chord-Ring, der in Abbildung 3-3 dargestellt ist, speichert lediglich direkte Nachfolger der Knoten. Dieses führt dazu, dass das Suchverfahren nicht mehr funktioniert, wenn ein Knoten ausfällt, da der Chord-Ring „reißt". Diesem kann durch die Erhöhung des Parameters r vorgebeugt werden.

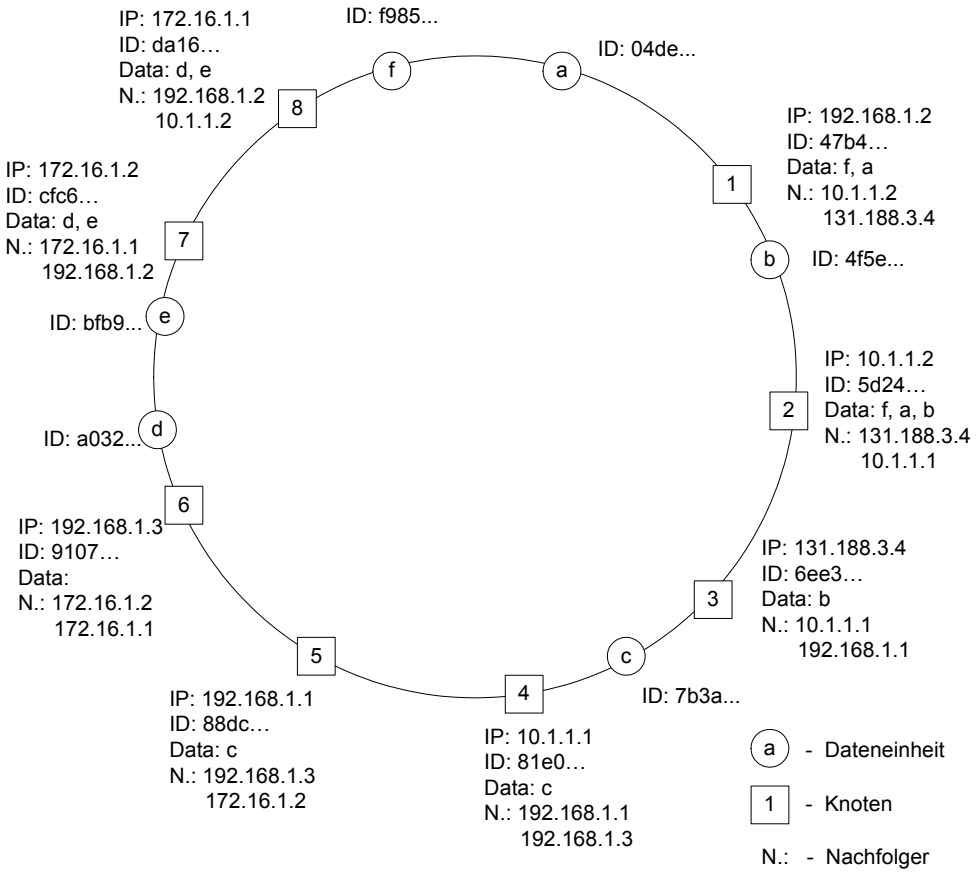


Abbildung 3-4: Redundanz durch Erhöhung des Parameters r (r=2)

In Abbildung 3-4 hat die Nachfolgerliste, die auf jedem Knoten verwaltet wird, die Länge r = 2. Die Integrität des Rings wird in diesem Fall durch einen einzigen Ausfall nicht beeinträchtigt. Auch wenn mehrere, nicht aufeinander folgende Knoten ausfallen, droht kein Riss im Ring. Die Größe des Parameters r bestimmt den Redundanzgrad des Rings.

Ferner werden in diesem Ring die Dateneinheiten nicht nur auf dem zuständigen Knoten gespeichert, sondern werden anschließend auch auf dessen Nachfolger repliziert. Während in Abbildung 3-3 zum Beispiel beim Ausfall des Knotens 1 die Dateneinheiten f und a verloren gehen, bleiben sie in Abbildung 3-4 erhalten, da sie auch auf dem Knoten 2 gespeichert sind.

Um die Verfügbarkeit der Daten zu erhöhen, empfiehlt es sich, nach dem Speichern diese auf bis zu `r` Nachfolger zu replizieren. Die Menge der Replikate bestimmt die Anzahl aufeinander folgender Knoten, die ausfallen können, ohne einen Datenverlust zu verursachen. Da `r` die Länge der Nachfolgerliste ist, sind mehr als `r` Replikate nicht sinnvoll, da der Ausfall von `r` aufeinander folgenden Knoten den Riss des Rings verursacht und dadurch ein Lookup nicht mehr funktioniert.

Möchte ein Knoten dem Ring beitreten, so muss er einen Knoten ansprechen, der bereits Teil des Rings ist (einen so genannten Bootstrap-Knoten). Beim ersten Anblick stellt dieses eine starke Einschränkung dar. Das Problem ist in reinen Peer-to-Peer-Umgebungen, in denen keine zentralen Server vorkommen, als das *Bootstrapping-Problem* bekannt. Es kann zum Beispiel durch die Verwendung eines Caches gelöst werden, der einige Knoten speichert, die in der Vergangenheit Teil des Rings waren. Dieser Cache wird in manchen Peer-to-Peer-Netzen wie etwa Gnutella[3] *Kontaktliste* genannt. Falls keiner der Knoten im Cache verfügbar ist, oder falls der Cache leer ist, so ist Benutzerinteraktion notwendig: der Benutzer muss über andere Kanäle (zum Beispiel WWW) eine aktuelle Kontaktliste beziehen und diese dem Agenten mitteilen.

Wurde ein Knoten gefunden, der schon im Ring ist, so wird diesem die eigene ID mitgeteilt. Er berechnet dessen Nachfolger und damit den Platz des neuen Knotens im Ring. Der neue Knoten holt sich von seinem Nachfolger die Daten, für die er zuständig ist, und veranlasst die Aktualisierung der Nachfolger- und Vorgängerzeiger, die durch das Beitreten beeinträchtigt wurden.

Durch den Ausfall von Knoten werden diese Zeiger zusätzlich beeinträchtigt: sie zeigen nämlich auf einen Knoten, den es nicht mehr gibt. Auch die Einträge der Fingertabellen sind unter Umständen nicht mehr aktuell. Aus diesem Grund wird auf jedem Knoten in regelmäßigen Abständen ein Vorgang namens *Stabilisierung* gestartet, durch den der Vorgänger, die Nachfolger und die Fingertabelle bei Bedarf aktualisiert werden. Die genaue Art und Weise, wie die Stabilisierung zu implementieren ist, kann man [SMK+01] entnehmen.

[3] http://www.gnutella.com

Kapitel 4

Anonymität in Peer-to-Peer-Umgebungen

Bisher wurden das UniTEC-System und dessen Komponenten beschrieben und die Motivation für das zu entwickelnde anonyme Kommunikationssubsystem geliefert. Im letzten Kapitel wurden einige existierende Ansätze beschrieben, die bei der Entwicklung dieses Subsystems hilfreich sein können.

In diesem Kapitel werden die Anforderungen an die Kommunikationsschicht aufgestellt und anschließend wird eine geeignete Lösung zu deren Realisierung vorgeschlagen. Zum Schluss werden einige wichtige Sicherheitsaspekte dieses Ansatzes untersucht und Erweiterungen vorgeschlagen, die die Sicherheit des Systems erhöhen.

4.1 Anforderungen

Dieser Abschnitt beschreibt die Anforderungen an dieses Subsystem, die während der Anforderungsanalyse aufgestellt wurden.

Dienstart

Das Subsystem soll den verschlüsselten Datentransport zwischen je zwei Pseudonymen generell innerhalb von TCP/IP-Netzen und speziell im Internet ermöglichen. Die Implementierung eines verbindungsorientierten Dienstes wird nicht verlangt, da ausschließlich diskrete Datenpakete – vor allem Anfragen nach Empfehlungen und Antworten auf diese Anfragen, die Empfehlungen enthalten – transportiert werden. Der Sender eines Datenpakets sollte jedoch darüber informiert werden, ob der Datentransfer erfolgreich war, etwa durch den Empfang eines Bestätigungspakets.

Anonymität

Das Subsystem soll die Anonymität aller Pseudonyme gewährleisten. Zu keinem Zeitpunkt darf die Zuordnung zwischen dem Pseudonym und dessen persönliche Daten – unter anderem auch die IP-Adresse des Agenten, innerhalb dessen das Pseudonym aktiv ist – ohne Absicht des Pseudonyms anderen Instanzen als dem Besitzer des Pseudonyms bekannt sein. Diese Zuordnung, und damit verbunden die wahre Identität des Pseudonyms, darf mit vertretbarem Aufwand nicht herausgefunden werden, es sei denn, die Identität wird explizit mitgeteilt. Es ist also ein Kommunikationsmechanismus zu entwickeln, das sowohl Sender- als auch Empfängeranonymität bietet.

Peer-to-Peer-Kommunikation

Die Kommunikation sollte idealerweise ohne den Einsatz zentraler Komponenten auskommen, da durch diese das gesamte Subsystem empfindlich gegenüber Ausfällen wird und deren Betrieb und Wartung mit hohen Kosten verbunden ist. Die Verwendung zentraler Komponenten setzt ein hohes Maß an Vertrauen gegenüber diesen voraus.

Zudem wird angenommen, dass die Topologie des Netzes, das zur Kommunikation verwendet wird, stabil ist, und die durchschnittliche Aktivität der Agenten sich über mehrere Stunden erstreckt, um die entstehende Fluktuation im Netz gering zu halten.

Skalierbarkeit

Das Subsystem ist skalierbar zu gestalten, so dass es sich unter hoher Last nur geringfügig anders verhält als unter geringer Auslastung. Hieraus folgt unter anderem, dass die Kommunikation keine hohe Bandbreitenauslastung verursachen darf, da nicht davon ausgegangen werden darf, dass alle Benutzer im Internet über Breitbandverbindungen verfügen.

Dienstgüte

Da die Implementierung von Anonymität generell die Latenzzeiten der Kommunikation erhöht, sollte ein Mechanismus gefunden werden, durch den der Benutzer sich nach Wunsch zwischen einem hohen Anonymitätsgrad und einer geringen Latenzzeit entscheiden kann.

4.2 Lösungsentwurf

Im letzten Kapitel wurden einige Verfahren zur Wahrung der Anonymität beschrieben. Leider erfüllt keines dieser Verfahren gleichzeitig die Anforderungen an Skalierbarkeit und Anonymität. Onion Routing hat jedoch entscheidende Vorteile bezüglich der Skalierbarkeit im Vergleich zu anderen Verfahren, da die Pakete einer bestimmten Route folgen und dadurch die Überflutung großer Teile des zur Verfügung stehenden Netzes vermieden wird. Obwohl es die Anforderungen an die Anonymität nicht vollständig erfüllt, kann es dahingehend erweitert werden, dass es ein brauchbares Verfahren für die Implementierung des Kommunikationssubsystems darstellt.

Dieser Abschnitt beschreibt einen neuen Ansatz, der auf Onion Routing basiert, jedoch auch Empfängeranonymität bietet.

Destination Routing

In Kapitel 3 wurde Onion Routing als ein Source-Routing-Verfahren beschrieben: der Sender wählt eine Route durch mehrere Mixe und präpariert anschließend die Nachricht, so dass anhand der Routing-Informationen, die im entstandenen verschlüsselten Paket enthalten sind, die Nachricht entlang dieser Route versendet werden kann und schließlich den Empfänger erreicht, wenn alle Mixe der Route verfügbar sind.

Der Sender kann den letzten Abschnitt der Route jedoch nur dann bestimmen, wenn er die Adresse des Empfängerpseudonyms kennt, was gegen die Anforderungen an die Anonymität verstößt.

Die einzige Instanz im Netz, der die Zuordnung zwischen einem Pseudonym und der Adresse, unter der dieses Pseudonym erreichbar ist, bekannt sein darf, ist das Pseudonym selbst bzw. der Benutzer, der sich hinter diesem Pseudonym verbirgt. Basiert die Kommunikation auf Routing, so muss aus diesem Grund das Empfängerpseudonym an der Bestimmung der Route beteiligt sein.

Routing-Header

Jeder Agent des Netzes kann prinzipiell als Mix fungieren, der Benutzer des Agenten entscheidet, ob der Mix-Dienst aktiviert wird oder nicht.

Möchte ein Pseudonym erreichbar sein, so wählt es eine bestimmte Anzahl Mixe und bildet daraus eine Mix-Kaskade. Die Mixe dieser Kaskade stellen die Zwischenstationen der Route zu diesem Pseudonym dar. Anhand der gewählten Kaskade und der IP-Adresse des Pseudonyms wird gemäß des gleichen Verfahrens, das auch Onion Routing verwendet, ein verschlüsseltes Paket erstellt. Hierbei werden jedoch keine Nutzdaten verschlüsselt, sondern die Adresse des Empfängerpseudonyms. Diese wird nacheinander mit den öffentlichen Schlüsseln der Mixe verschlüsselt. Jede Schicht enthält zusätzlich die Adresse der nächsten Zwischenstation (siehe Abbildung 4-).

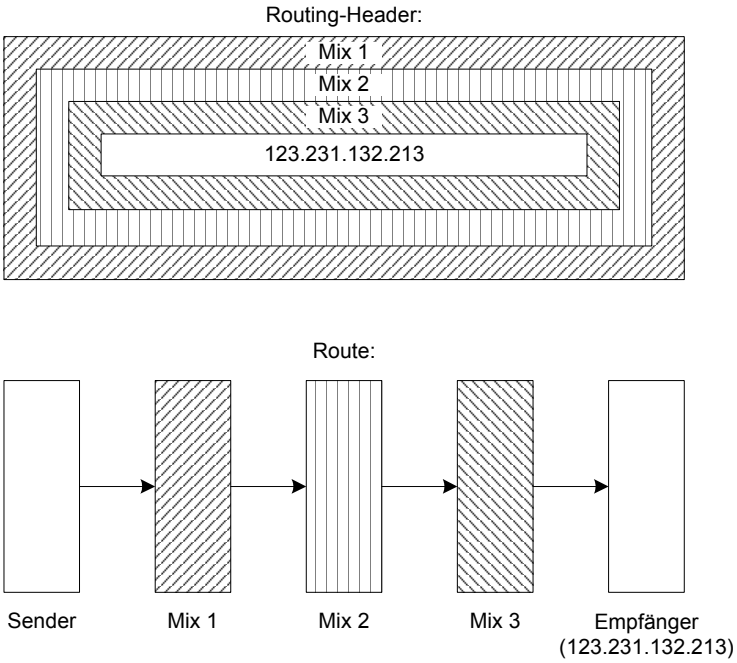


Abbildung 4-1: Routing-Header und davon bestimmte Route

Verzeichnisdienst

Möchte ein beliebiges Pseudonym eine Nachricht an ein anderes Pseudonym versenden, so benötigt es dessen Routing-Header. Die verschlüsselte Nachricht wird zu-

sammen mit dem Routing-Header entlang der im Header enthaltenen Route transportiert, bis sie den Empfänger erreicht. Jeder Mix entfernt dabei eine Schicht des Routing-Headers und leitet die Nachricht zusammen mit dem veränderten Routing-Header weiter.

Voraussetzung hierfür ist jedoch, dass das Empfängerpseudonym seinen Routing-Header allen potentiellen Sendern zur Verfügung stellt, etwa in einem Verzeichnisdienst (siehe Abbildung 4-2). Zusätzlich wird auch der öffentliche Schlüssel des Empfängerpseudonyms für die Verschlüsselung der Nachricht benötigt – dieser kann zusammen mit dem Routing-Header im Verzeichnisdienst verfügbar gemacht werden.

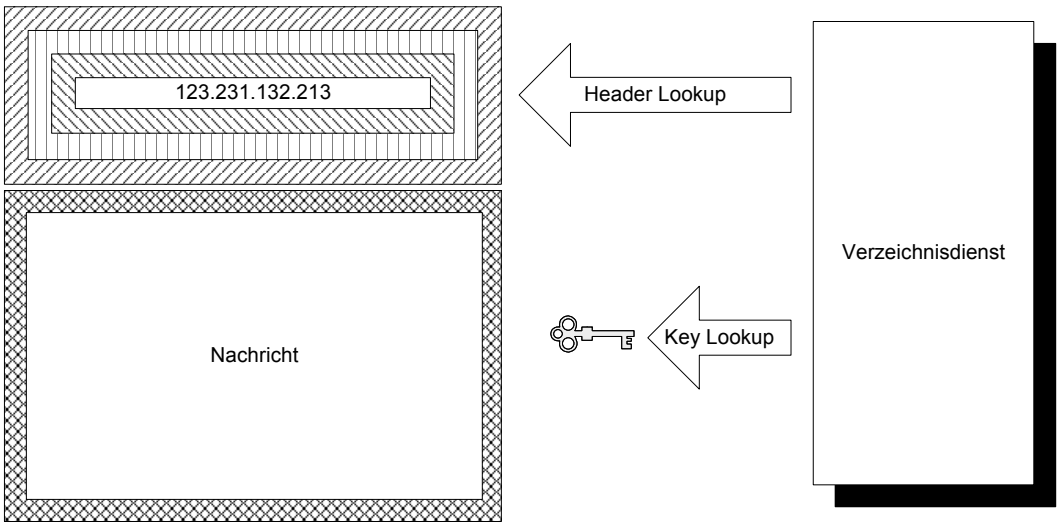


Abbildung 4-2: Header- und Key-Lookup

Der Verzeichnisdienst kann prinzipiell zentral oder dezentral gespeichert werden. Da jedoch eine komplett verteilte Architektur angestrebt wird, werden die Daten des Verzeichnisdiensts auf mehrere Knoten in einem Peer-to-Peer-Netz verteilt abgelegt. Weitere Details zum Verzeichnisdienst werden in Abschnitt 4.4 beschrieben.

Extended Destination Routing

Ein potentieller Angreifer könnte seinen Routing-Header so manipulieren, dass dieser ihn selbst als einzigen Mix enthält. Die Route zwischen Sender- und Empfängerpseudonym würde nur deren Agenten enthalten, sie wäre also äquivalent mit einer direkten Verbindung (siehe Abbildung 4-3) und würde bedeuten, dass jedes Pseudonym, das eine Nachricht an den Angreifer sendet, dadurch seine Anonymität gefährdet.

Da der Routing-Header verschlüsselt ist und nur die Adresse des ersten Mixes im Klartext enthält, kann dieser Angriff nicht als solcher erkannt werden. Um diesen zu verhindern, muss der Sender die vom Empfänger gewählte Kaskade um einen eigenen Teil erweitern. Dieses Verfahren wird in Abschnitt 4.5 näher beschrieben.

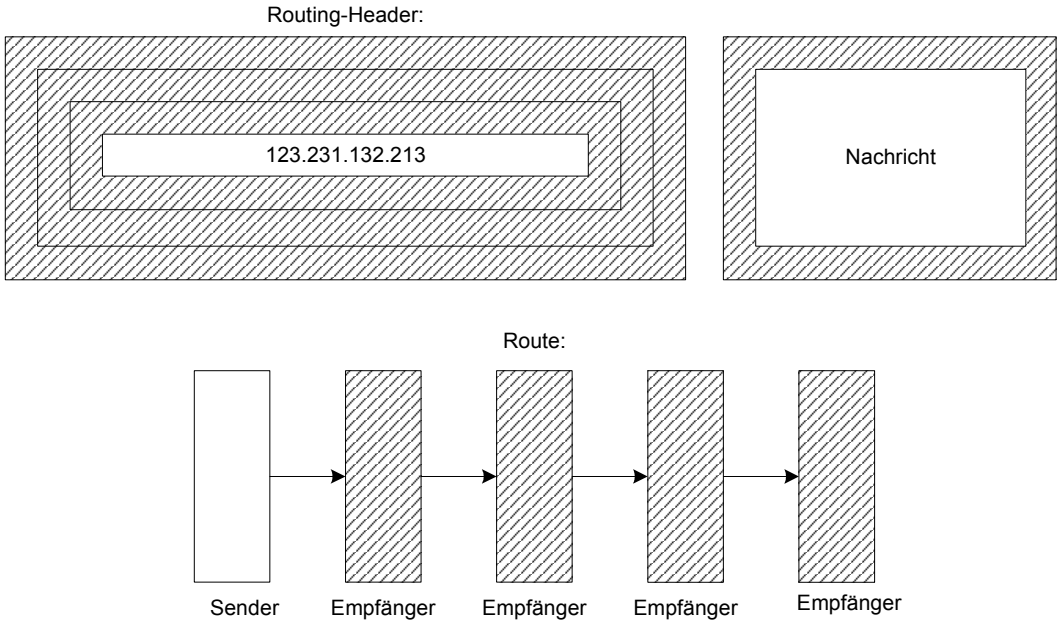


Abbildung 4-3: Keine Senderanonymität durch manipulierten Header

4.3 Pseudonym-IDs

Der Verzeichnisdienst speichert eine Zuordnung zwischen den Pseudonymen und deren Erreichbarkeitsdaten (d.h. öffentlicher Schlüssel und Routing-Informationen). Hierzu benötigt jedes Pseudonym eine eindeutige *ID*. Die ID eines Pseudonyms ist demnach ein Merkmal, das das Pseudonym im Netz eindeutig kennzeichnet, durch das es sich also von den anderen Pseudonymen unterscheidet – etwa eine Zahl oder eine alphanumerische Zeichenkette. Die Eindeutigkeit der ID kann jedoch in einer reinen Peer-to-Peer-Umgebung ohne den Einsatz zentraler Komponenten, etwa einer zentralen ID-Vergabestelle, nicht auf effiziente und gleichzeitig sichere Weise gewährleistet werden.

Öffentlicher Schlüssel als ID

Jedes Pseudonym besitzt ein Schlüsselpaar, das unter anderem zur Kommunikation verwendet wird. Die öffentlichen Schlüssel haben eine beträchtliche Länge – üblich sind heutzutage Längen von mindestens 1024 Bits.

Die Wahrscheinlichkeit, dass zwei Pseudonyme den gleichen öffentlichen Schlüssel besitzen, ist verschwindend gering, daher kann man von der Eindeutigkeit der öffentlichen Schlüssel ausgehen. Man könnte also den öffentlichen Schlüssel als Pseudonym-ID verwenden. Die erhebliche Länge dieser Schlüssel ermöglicht jedoch keine effiziente Kommunikation. Daher ist eine kürzere ID erwünscht, die idealerweise in einer 1:1-Beziehung zum öffentlichen Schlüssel steht.

ID-Erzeugung durch Hashing

Die 1:1-Beziehung zwischen ID und Schlüssel kann zum Beispiel durch die Generierung der ID als Hashwert des öffentlichen Schlüssels annähernd erreicht werden. Bei der Erstellung eines Pseudonyms wird dessen öffentlicher Schlüssel generiert und

daraus die ID berechnet. Kryptographische Hashfunktionen erzeugen gleichverteilte Werte: die Wahrscheinlichkeit, dass die Berechnung des Hashs eines zufälligen Eingangswerts ein bestimmtes Element aus der Wertmenge der Hashfunktion liefert, ist für alle Elemente der Wertmenge gleich groß. Ferner generieren asymmetrische Verschlüsselungsverfahren wie zum Beispiel RSA gleichverteilte öffentliche Schlüssel. Dieses hat zur Folge, dass IDs, die als Hashwerte von öffentlichen Schlüsseln generiert werden, gleichverteilt sind.

Theoretisch sind Kollisionen möglich, da der Wertebereich dieser Abbildung um Größenordnungen geringer als deren Definitionsmenge ist. Die Wahrscheinlichkeit einer Kollision ist jedoch auch bei einer sehr hohen Pseudonymanzahl verschwindend gering. In [Widm03] wird die Wahrscheinlichkeit für eine Kollision beim Einsatz der SHA-1-Funktion berechnet. Bei 30 Billionen Pseudonymen beträgt die Wahrscheinlichkeit einer Kollision $2{,}05 * 10^{-35}$.

Die Zugehörigkeit des öffentlichen Schlüssels zum Pseudonym kann jederzeit durch eine Hashberechnung geprüft werden. Beim Einsatz einer sicheren Hashfunktion wie z.B. SHA-1 kann man zu einer bestimmten ID nur durch einen Brute Force-Angriff einen passenden öffentlichen Schlüssel finden, dieser Angriff ist bei einer Schlüssellänge von 1024 Bit jedoch nicht in vertretbarer Zeit durchführbar.

Kollisionsvermeidung

Möchte man trotz der geringen Kollisionswahrscheinlichkeit eine Kollision ausschließen, so kann man hierzu die ID zusammen mit dem öffentlichen Schlüssel dauerhaft im Verzeichnisdienst speichern, unabhängig vom Aktivitätsstatus des Pseudonyms. Bei der Durchführung eines Updates der Routing-Informationen wird sowohl die ID als auch der Schlüssel des Pseudonyms an den zuständigen Knoten in der DHT gesendet. Dieser prüft vor dem Speichern der Informationen, ob die ID bereits gespeichert ist. Ist dieses der Fall, so vergleicht er den dazu gehörenden gespeicherten Schlüssel mit dem Schlüssel, der in der Update-Nachricht enthalten ist. Das Update wird nur bei Übereinstimmung der Schlüssel durchgeführt, oder wenn die ID noch nicht gespeichert wurde, es sich also um ein neues Pseudonym handelt. In letzterem Fall wird vor dem Speichern die Gültigkeit des Schlüssels durch die Berechnung dessen Hashwerts geprüft.

Ist das Update fehlgeschlagen, so wird das initiierende Pseudonym darüber in Kenntnis gesetzt. Bei der Versendung dieser Nachricht wird einer der Routing-Header aus der Update-Nachricht verwendet. Nach Empfang der Nachricht weiß der Benutzer, dass eine Kollision aufgetreten ist, und kann ein neues Pseudonym erstellen.

Die dauerhafte Speicherung von IDs und öffentlichen Schlüsseln hat den Nachteil, dass nach einer längeren Zeitspanne die Menge der in der DHT gespeicherten Daten sehr groß wird. Je höher der Redundanzgrad des Netzes ist, desto mehr Replikate existieren und dadurch muss jeder einzelne Agent mehr Daten lokal speichern. Eine wesentlich effizientere Lösung ist die Auswahl eines geeigneten ID-Raums, der eine sehr geringe Kollisionswahrscheinlichkeit gewährleistet. Der 160-Bit-Wertebereich der SHA-1-Funktion erfüllt diese Voraussetzung (siehe hierzu [Widm03]). Pseudonym-IDs aus diesem Bereich können außerdem direkt als IDs der Routing-

Informationen beim Speichern im Chord-Ring verwendet werden (siehe Abschnitt 4.4). Daher ist die SHA-1-Funktion zur Erstellung von IDs gut geeignet.

4.4 Verzeichnisdienst

Der Verzeichnisdienst hat die Aufgabe, diejenigen Informationen über Pseudonyme zu speichern, die bei der Kommunikation gebraucht werden, nämlich den öffentlichen Schlüssel und den Routing-Header eines Pseudonyms. Der öffentliche Schlüssel wird für die Verschlüsselung der Nachricht verwendet, so dass nur der Besitzer des entsprechenden geheimen Schlüssels – der Empfänger – diese lesen kann. Der Routing-Header wird für den Transport der Nachricht zum Empfänger unter Wahrung der Anonymität von Sender und Empfänger benötigt.

Der Verzeichnisdienst muss vier Operationen ermöglichen: das Speichern des öffentlichen Schlüssels, die sichere Aktualisierung des Routing-Headers, sowie den *Lookup*, also das Nachschlagen von Routing-Header und Schlüssel.

Verteilter Verzeichnisdienst

Eine Möglichkeit der Implementierung eines Verzeichnisdienstes wäre die Speicherung der zu verwaltenden Daten in einer zentralen Datenbank. Da der Dienst hochverfügbar sein muss, wäre diese Lösung jedoch mit Kosten in nicht vertretbarer Höhe verbunden. Zudem erfordert die zentrale Speicherung ein hohes Maß an Vertrauen gegenüber dem Dienstbetreiber, da sich Manipulationen an den Daten auf die Verfügbarkeit aller Pseudonyme auswirken können.

Eine weitere Möglichkeit ist die verteilte Speicherung der Daten. Hierbei werden die Daten nicht zentral auf einem oder mehreren Rechnern gehalten, sondern sie werden gleichmäßig auf das Peer-to-Peer-Netz verteilt und lokal von den einzelnen Agenten verwaltet. Ein verteilter Verzeichnisdienst ist zwar wesentlich aufwendiger zu implementieren als ein zentraler, er hat aber den Vorteil, dass er weder aufwendige Hardware noch hohe Kosten für den Betrieb und die Wartung voraussetzt. Er ist außerdem weniger empfindlich gegenüber Netz-, Software- oder Hardwareausfällen. Durch die Verteilung der Daten auf alle Agenten des Peer-to-Peer-Netzes erreicht man eine hohe Verfügbarkeit des Datenbestands als Ganzes, da einzelne Ausfälle sich nur auf die Verfügbarkeit einzelner Dateneinheiten auswirken. Durch Replikation der Daten auf mehrere Agenten wird die Verfügbarkeit der Daten zusätzlich erhöht.

Zur verteilten Datenspeicherung bietet sich der Einsatz einer verteilten Hashtabelle (DHT) wie etwa Chord an. Chord wurde in Kapitel 3 ausführlich beschrieben. Die IDs der Werte, die in der Hashtabelle gespeichert werden, entsprechen den IDs der Pseudonyme. Diese werden durch Hashing des öffentlichen Schlüssels mittels der Funktion SHA-1 berechnet. Hierdurch wird die Zugehörigkeit eines öffentlichen Schlüssels zu einer ID und dadurch zu einem Pseudonym überprüfbar (siehe Abschnitt 4.3). Die Werte, die zu einer ID gespeichert werden, sind zum einen der öffentliche Schlüssel des Pseudonyms, und zum anderen Routing-Informationen, die Routing-Header enthalten, durch die das Pseudonym im Netz erreichbar wird.

Routing Update

Möchte ein Pseudonym im Netz von anderen erreicht werden, so führt es ein *Routing Update* durch. Hierbei speichert es seine Routing-Informationen und seinen öffentlichen Schlüssel im Verzeichnisdienst. Unter Routing-Informationen versteht man einen oder mehrere Routing-Header, je nach dem gewählten Grad der Verfügbarkeit, der vom Pseudonym gewählt wurde.

Das Update muss zum einen so durchgeführt werden, dass dabei die Anonymität des Pseudonyms, das das Update durchführt, nicht kompromittiert wird. Ferner müssen Vorkehrungen getroffen werden, die verhindern, dass ein Angreifer die Routing-Informationen aus dem Verzeichnisdienst durch ungültige Daten überschreibt.

Verfügbarkeitsgrad

Um die Verfügbarkeit der Pseudonyme zu erhöhen, ist es sinnvoll, mehr als einen Routing-Header im Verzeichnisdienst bereitzuhalten.

Der Sender wählt zufällig einen davon aus, und verwendet ihn beim Versenden der Nachricht. Kommt sie beim Empfänger an, so sendet dieser ein Bestätigungspaket zurück (siehe Abbildung 4-5). Falls ein Mix ausfällt, der in einem Header enthalten ist, so erfährt der Sender der Nachricht dieses dadurch, dass nach Ablauf eines festgelegten Timeouts keine Bestätigungsnachricht eingetroffen ist. Danach kann ein anderer Header verwendet werden.

Update-Proxy

Das Aktualisieren der Routing-Header eines Pseudonyms, also das Schreiben der Header in die DHT, kann ohne weitere Maßnahmen unter Umständen zur Aufdeckung der Identität dieses Pseudonyms führen, wenn der für die Speicherung des Headers zuständige Knoten während der Datenübertragung die IP-Adresse des Senders – d.h. des Agenten, innerhalb dessen das Pseudonym aktiv ist – erfährt.

Aus diesem Grund muss das Update des Routing-Headers eines Pseudonyms über einen so genannten *Update-Proxy* erfolgen. Der Update-Proxy ist ein Knoten, der im Auftrag des Pseudonyms den Header in der DHT speichert.

Das Pseudonym sendet den Routing-Header an den Update-Proxy. Dabei wird das klassische Onion Routing verwendet. Dieses ist deshalb möglich, weil die Adresse des Knotens, der als Update-Proxy fungiert, bekannt ist. Als Update-Proxy kann zum Beispiel ein beliebiger Mix verwendet werden.

Update-Zeitpunkt

Von großer Bedeutung ist die Frage nach dem Zeitpunkt der Durchführung eines Updates der Routing-Informationen. Wenn ein Pseudonym aktiv wird, so muss ein Update vorgenommen werden. Es ist aber auch sinnvoll, Routing-Informationen nach einer bestimmten Zeitspanne, in der diese nicht aktualisiert oder abgerufen wurden, aus dem Verzeichnisdienst zu entfernen, da dieses mit hoher Wahrscheinlichkeit bedeutet, dass das Pseudonym nicht mehr aktiv ist. Hierzu kann zur Speicherung der Routing-Header eine Cache-Struktur verwendet werden, die in regelmäßigen Abständen geprüft und von veralteten Einträgen gesäubert wird. Damit dieser An-

satz funktioniert und die Pseudonyme trotzdem verfügbar bleiben, muss das Pseudonym in regelmäßigen Abständen prüfen, ob die im Verzeichnisdienst gespeicherten Routing-Informationen noch nutzbar sind, also ob die Mixe der Kaskaden, auf denen die Routing-Header basieren, noch verfügbar sind. Ist dieses der Fall, so wird dem Verzeichnisdienst mitgeteilt, dass das Vorhalten der aktuellen Routing-Informationen erwünscht ist. Diese Mitteilung muss an alle Knoten gesendet werden, die die Replikate der Routing-Informationen speichern. Um die Anonymität des Pseudonyms zu wahren, muss diese Mitteilung auch über einen Update-Proxy versendet werden.

Eine implementierungstechnisch weniger aufwendige, wenn auch etwas rechenaufwendigere Möglichkeit ist die in regelmäßigen Abständen wiederholte Erstellung neuer Routing-Header anhand der aktuell und lokal verfügbaren Mix-Informationen und die anschließende unbedingte Durchführung eines Updates basierend auf den neu berechneten Routing-Informationen.

Der öffentliche Schlüssel eines Pseudonyms bleibt über längere Zeit gleich, daher muss dieser nicht aktualisiert werden. Wie wir jedoch weiter unten sehen werden, wird der Schlüssel bei einem Update der Routing-Informationen implizit versendet.

Authentizität

Routing-Informationen werden mit Hilfe einer *Update-Nachricht* aktualisiert. Diese Nachricht enthält in erster Linie die Routing-Header, die der Verzeichnisdienst speichern soll. Ferner ist die ID des Pseudonyms enthalten, das das Update durchführt. Um die Authentizität der Daten prüfen zu können, wird der Inhalt der Nachricht signiert – der Hashwert der Nachricht wird berechnet und mit dem geheimen Schlüssel des Pseudonyms verschlüsselt. Die Signatur wird anschließend an die Nachricht angehängt.

Die Update-Nachricht wird über mehrere Mixe an einen Update-Proxy gesendet. Dieser ermittelt den Knoten in der DHT, der für die Speicherung der Routing-Informationen zuständig ist, und versendet die Update-Nachricht an diesen. Dieser prüft die Signatur und anschließend sendet er die Nachricht an seine Nachfolger, um die darin enthaltenen Daten zu replizieren. Auch die Nachfolgerknoten prüfen die Signatur vor dem Speichern.

Zur Prüfung der Signatur wird der öffentliche Schlüssel des Pseudonyms benötigt. Dieser wird in der Update-Nachricht mitgeschickt. Wird eine ID-Kollisionsprüfung durchgeführt (siehe Abschnitt 4.3), so muss berücksichtigt werden, dass ein kollidierendes Pseudonym eine ID hat, die im Verzeichnis bereits gespeichert ist, es aber einen anderen öffentlichen Schlüssel als den gespeicherten besitzt. Eine Kollision kann demnach nur erkannt werden, wenn der Schlüssel in jeder Update-Nachricht enthalten ist.

Wird keine Kollisionsprüfung implementiert, so muss nicht jedes Mal der Schlüssel mitgesendet werden. Er wird beim Aktivieren des Pseudonyms vom verantwortlichen Knoten gespeichert und künftige Update-Nachrichten können anhand des gespeicherten Schlüssels überprüft werden.

Die alleinige Prüfung der Signatur gewährleistet aber nicht die Authentizität der Routing-Informationen. Sehr wichtig ist auch die Prüfung der Gültigkeit des Schlüssels, der in der Nachricht enthalten ist. Hierzu wird geprüft, ob der Hash des Schlüssels identisch mit der ID ist. Ist dieses der Fall, so wurde die Update-Nachricht tatsächlich von dem Pseudonym mit der entsprechenden ID erstellt.

Aktualität

Allein die Überprüfung der Authentizität einer Update-Nachricht garantiert noch keine ausreichende Sicherheit bei der Update-Durchführung. Ein Angreifer kann einen *Replay-Angriff* durchführen, indem er die Update-Nachricht zwischenspeichert, die der Update-Proxy an den für die Speicherung zuständigen Knoten sendet. Zu einem späteren Zeitpunkt sind die in der Nachricht enthaltenen Mix-Informationen möglicherweise nicht mehr gültig, da einige Mixe mittlerweile ausgefallen oder inaktiv sind. Der Angreifer kann die gespeicherte Nachricht nun an den zuständigen Knoten versenden. Dieser prüft die Signatur, die nach wie vor gültig ist, und speichert die enthaltenen Routing-Informationen. Dieses hat zur Folge, dass das betroffene Pseudonym nicht mehr erreichbar ist.

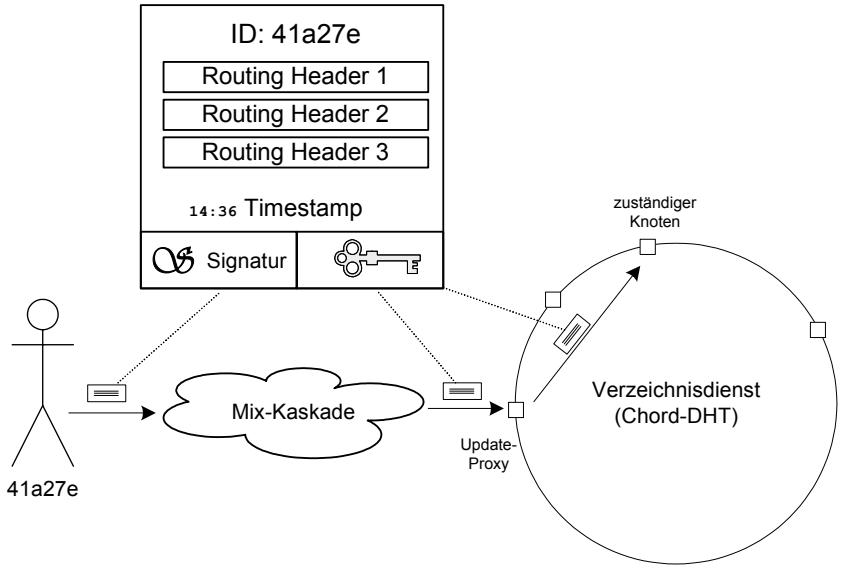


Abbildung 4-4: Routing Update-Protokoll

Um Replay-Attacken vorzubeugen, muss die Aktualität der Routing-Informationen geprüft werden. Zu diesem Zweck enthält die Update-Nachricht einen *Zeitstempel*, der den Zeitpunkt der Erstellung der Nachricht enthält. Der für die Speicherung zuständige Knoten prüft das Alter der Nachricht und verwirft diese, falls das Alter einen eingestellten Toleranzwert übersteigt. Diese Methode funktioniert jedoch nur, wenn die Systemuhren der Agenten synchronisiert sind. Eine Synchronisation kann zum Beispiel über Zeitserver erfolgen. Diese können mithilfe des standardisierten Network Time Protocol (NTP[1]) über das Internet kontaktiert werden.

Sowohl das Routing Update-Protokoll, als auch das Format der Update-Nachrichten werden in Abbildung 4-4 dargestellt.

[1] Informationen, Serverlisten und Referenzimplementierung unter http://www.ntp.org

Lookup

Möchte sich ein Pseudonym mit einem anderen in Verbindung setzen, so benötigt es dessen öffentlichen Schlüssel und Routing-Informationen (siehe Abbildung 4-5). Hierzu fragt es den Verzeichnisdienst nach diesen Informationen, es führt also einen Lookup durch. Während eines Lookups wird der Knoten ermittelt, der für die Speicherung dieser Informationen zuständig ist, und anschließend wird dieser befragt.

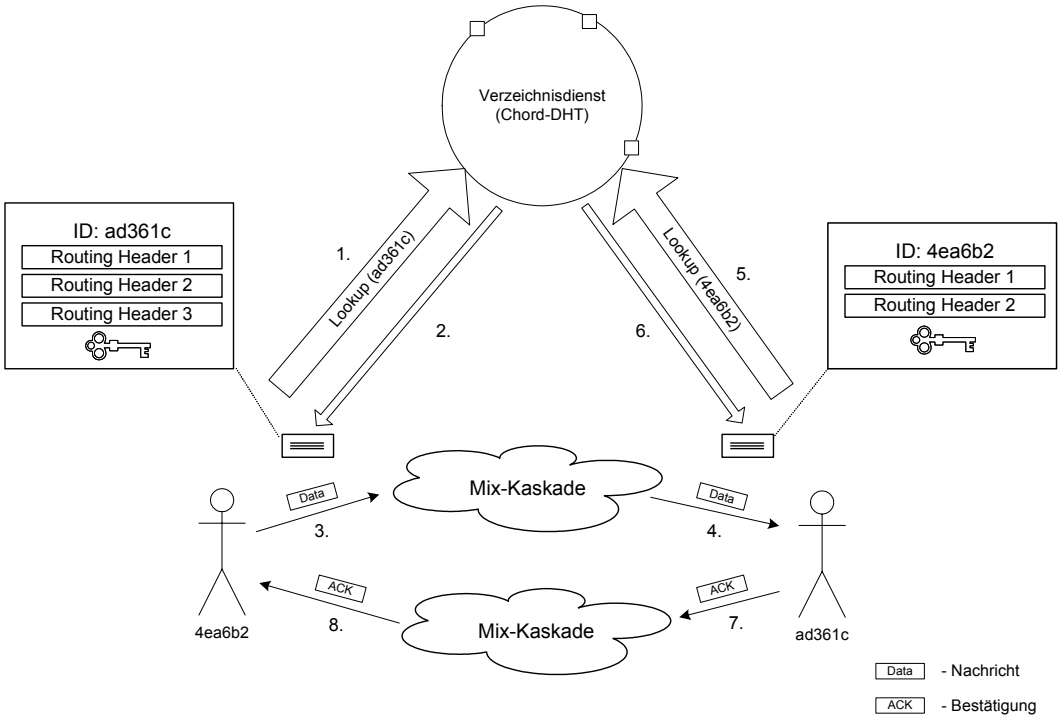


Abbildung 4-5: Lookup, Versenden einer Nachricht und Bestätigung

Dummy-Lookups

Der Lookup von Routing-Informationen und öffentlichen Schlüssel wird im Normalfall ohne anonymisierende Maßnahmen und unverschlüsselt durchgeführt. Es wird eine direkte Verbindung zu dem Knoten aufgebaut, der die relevanten Informationen speichert. Eine Verkehrsanalyse würde verraten, dass ein Pseudonym auf dem Agenten, der die Lookup-Verbindung aufgebaut hat, mit demjenigen Pseudonym kommunizieren möchte, das Gegenstand der Lookup-Anfrage war. Diese Information würde nicht direkt die Verschleierung der Identität der Pseudonyme gefährden, unter gewissen Umständen könnte sie aber als ergänzender Tipp verwendet werden.

Um dieses zu erschweren, kann man die Agenten in zufälligen Zeitabständen so genannte *Dummy-Lookups* durchführen lassen. Diese Operationen fragen Informationen über zufällige (bekannte) Pseudonyme ab mit dem Ziel, auf diese Weise die echten Lookups von Dummy-Lookups schwer unterscheidbar zu machen.

Lookup-Proxy

Eine andere Möglichkeit wäre, die Lookup-Anfrage über mehrere Mixe an einen *Lookup-Proxy* (analog zum Update-Proxy) zu versenden, der die Anfrage an den zu-

ständigen DHT-Knoten weiterleitet. Dieser sendet die angefragte Routing-Information an das anfragende Pseudonym. Dieses funktioniert nur, wenn die Lookup-Anfragenachricht den Routing-Header des Anfragenden enthält. Dieses Verfahren ist aber weniger performant, da es mit mehr Overhead verbunden ist – sowohl die Lookup-Anfrage, als auch die -Antwort werden über Mixe versendet.

4.5 Extended Destination Routing

In Abschnitt 4.2 wurde ein Angriffsszenario beschrieben, in dem der Empfänger seinen Routing-Header so manipuliert, dass er sich selbst als einziger Mix der Kaskade auswählt. Das Szenario wurde in Abbildung 4-3 dargestellt. Dieser Angriff kann dadurch verhindert werden, dass der Sender die vom Empfänger gewählte Mix-Kaskade um einen zusätzlichen Teil erweitert.

Dieses Verfahren wird als *Extended Destination Routing (EDR)* bezeichnet. Es verhindert, dass der Empfänger den Anonymitätsgrad der gesamten Route bestimmt und hierdurch Angriffe durch böswillige Empfänger ermöglicht. Sowohl der Sender als auch der Empfänger können nun über die Variierung der Länge der jeweiligen Teilkaskade den Anonymitätsgrad der Kommunikation bestimmen.

Erweiterung des Routing-Headers

Eine Möglichkeit zur Realisierung des EDR ist die Erweiterung des Routing-Headers nach dem Lookup. Hierbei wählt der Sender eine bestimmte Anzahl von Mixen aus und anhand dieser Mixe ergänzt er den Routing-Header um weitere äußere Schichten. Hierbei geht er wie bei der Erstellung des Headers durch den Empfänger vor: für jeden Mix kommt eine weitere Schicht hinzu, die durch die Verschlüsselung mit dem öffentlichen Schlüssel des jeweiligen Mixes und dem Hinzufügen der Mixadresse zustande kommt. Die Mixkaskade, die anschließend verwendet wird, entsteht durch Zusammenfügen der Teilkaskade des Senders mit der des Empfängers.

In Abbildung 4-6 ist die Entstehung eines erweiterten Routing-Headers dargestellt. Die Empfänger-Teilkaskade hat die Länge 3, während die Teilkaskade des Senders nur zwei Mixe enthält. Insgesamt summieren sich die erwünschten Anonymitätsgrade. Entscheiden sich sowohl Sender als auch Empfänger aus Performance-Gründen gegen Anonymität, so haben beide Teilkaskaden und hiermit auch die Gesamtkaskade die Länge 0. Der entstehende Routing-Header enthält allein die Adresse des Empfängers im Klartext.

Pattern Matching-Angriff

Durch eine genaue Verkehrsanalyse, bei der der ein- und der ausgehende Verkehr an den Mixen untersucht wird, kann die Route der Nachricht anhand der Erkennung des verschlüsselten Nachrichtenteils aus Abbildung 4-6 verfolgt werden: im Gegensatz zum Routing-Header, der von jedem Mix entschlüsselt und dadurch verändert wird, bleibt der verschlüsselte Nutzdatenteil während des Transports zwischen Sender und Empfänger gleich Der Angriff kann durch einen Angreifer zum Beispiel durch eine Mustererkennung (Pattern Matching) mit Hilfe der Berechnung des Hashwerts über den Nutzdatenteil durchgeführt werden. Der Angriff setzt voraus,

dass der Angreifer alle Mixe beobachten kann und somit in der Lage ist, der gesamten Route vom Sender zum Empfänger zu folgen und daher die Kommunikationsbeziehung zwischen Sender und Empfänger aufzudecken.

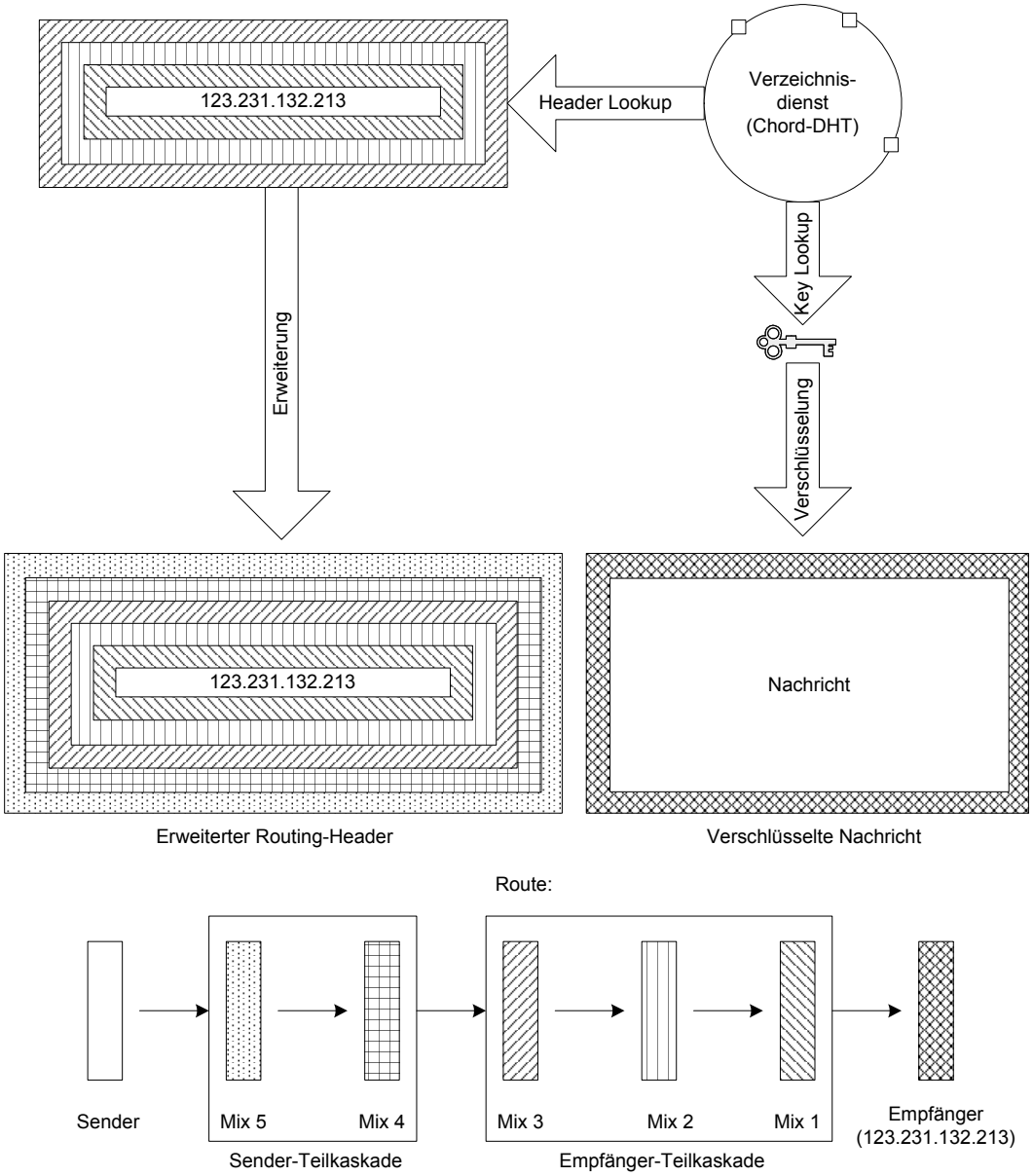


Abbildung 4-6: Erweiterung des Routing-Headers

Ein mächtiger Angreifer, der weite Teile des Netzes beobachten kann, ist in der Lage, Nachrichten an mehrere Pseudonyme zu senden und durch Verfolgung der Route mittels Pattern Matching die Identität dieser Pseudonyme aufzudecken, da er als Sender der Ersteller der Nachricht ist und damit das Muster des Nutzdatenteils einem Pseudonym zuordnen kann.

Bei Onion-Routing kann dieser Angriff nicht durchgeführt werden, da sich hier die Nutzdaten unterhalb der Verschlüsselungsschichten befinden und daher die gesamte Nachricht ihr Aussehen von Mix zu Mix ändert.

Hop-to-Hop-Verschlüsselung

Um einen Pattern Matching-Angriff zu vermeiden, sollte der Nutzdatenteil laufend, bei jedem Mixdurchlauf verändert werden. Eine Möglichkeit wäre die Verschlüsselung der gesamten Nachricht, die aus Header und Nutzdaten besteht, mit dem öffentlichen Schlüssel des jeweils nächsten Mixes in der Kaskade. Bei diesem Mix angekommen, wird die Nachricht entschlüsselt, die erste Schicht des Headers entschlüsselt und entfernt und anschließend wird die gesamte Nachricht erneut mit dem öffentlichen Schlüssel des nächsten Mixes verschlüsselt.

Hop-to-Hop-Verschlüsselung mit Padding

An Mixen können ein- und ausgehende Nachrichten bei geringer Auslastung eventuell anhand der Nachrichtengröße trotz Verschlüsselung einander zugeordnet werden. Um die Wahrscheinlichkeit dafür zu senken, kann bei der Verschlüsselung der Nachricht ein zufälliges Padding, also ein Auffüllen mit einer zufälligen Anzahl von Füllzeichen durchgeführt werden.

Um statistischen Analysen bei sehr unterschiedlichen Nutzdatenmengen in der Nachricht vorzubeugen, können die Warteschlangen der Mixe mehrere Größenklassen beinhalten, die Nachrichten ähnlicher Größe enthalten. Die Klassen werden einzeln in Schüben geleert, wobei darauf geachtet wird, dass eine Klasse mehrere Nachrichten enthält; bei Bedarf werden Dummy-Nachrichten der entsprechenden Größe erstellt, die anschließend an zufällige Mixe gesendet werden.

Last Mix-Angriff

Es ist sinnvoll zu entscheiden, ob der Mixdienst auf einem Agenten gestartet wird oder nicht, da der Betrieb eines Mixes erhebliche Rechenleistung und Bandbreite erfordert. Jedoch birgt die Abschaltung des Mix-Dienstes ein erhebliches Sicherheitsrisiko.

Beschreibung

Erkennt ein Mix, dass der nächste Agent der Route den Mix-Dienst nicht aktiviert hat, so kann er davon ausgehen, dass das Empfängerpseudonym innerhalb dieses Agenten aktiv ist, und dass er selbst der letzte Mix der Kaskade ist.

Handelt es sich beim letzten Mix der Kaskade zufällig um den Sender der Nachricht, und hat er erkannt, dass der nächste Knoten der Route der Empfänger ist, so kann er durch Pattern Matching anhand der verschlüsselten Nutzdaten dem Empfängerpseudonym eine IP-Adresse zuordnen. Dieses ist auch dann möglich, wenn Hop-to-Hop-Verschlüsselung eingesetzt wird, da jeder Mix die Nachricht nach deren Empfang entschlüsselt und hiermit die verschlüsselten Nutzdaten in der ursprünglichen Form vorliegen.

Abwehrmaßnahmen

Um diesen Angriff zu verhindern, muss vermieden werden, dass der Empfänger als solcher erkennbar ist. Hierzu sind folgende Maßnahmen notwendig:

1. Jeder Agent sollte den Mix-Dienst aktiviert haben, oder es sollte keine Möglichkeit geben, zwischen Mixen und Nicht-Mixen zu unterscheiden

2. Im Inneren des Routing-Headers darf sich nicht allein die IP-Adresse des Empfängers befinden. Der Empfänger sollte vor der Verschlüsselung der Adresse einen Datenblock zufälliger Größe hinzufügen. Dieser Block kann zum Beispiel die ID des Empfängerpseudonyms enthalten, verschlüsselt mit dem öffentlichen Schlüssel des lokalen Mixes (dieses setzt jedoch eine Aktivierung des Mix-Diensts auf allen Agenten voraus).

3. Der Empfänger muss verhindern, dass der letzte Mix der Kaskade durch eine Verkehrsanalyse erkennt, dass der Empfänger die Nachricht nicht weitersendet, zum Beispiel durch Versenden von Dummy-Nachrichten verschiedener Größe oder durch die Auswahl eines Mixes, der mit hoher Wahrscheinlichkeit nicht die Möglichkeiten einer Verkehrsanalyse hat (z.B. weil er sich in einem entfernten Subnet befindet)

Kaskadenerweiterung mit Onion Routing

Analog dem Last Mix-Angriff ist auch ein *First Mix-Angriff* denkbar. Falls der erste Mix gleichzeitig der Empfänger ist, darf er einer Nachricht nicht gleichzeitig einen Knoten und ein Pseudonym als Sender zuordnen, da hiermit die Adresse des Senders aufgedeckt wäre. Zur Verhinderung des Angriffs kann er einige Maßnahmen verwenden, die auch beim Last Mix-Angriff hilfreich sind. Zum einen kann er den Mix-Dienst aktivieren, er kann aber auch einen Mix aus einem entfernten Subnet wählen und Dummy-Nachrichten versenden. Dummy-Nachrichten sind vor allem in einem „stillen“ Netz sinnvoll, da sonst die Kommunikationsbeziehung zwischen Sender und Empfänger durch eine potentielle Verkehrsanalyse aufgedeckt werden kann, und dieses ein Risiko sowohl für den Sender als auch für den Empfänger darstellt.

Eine zusätzliche Möglichkeit ist die Verwendung des klassischen Onion Routing-Verfahrens bei der Erweiterung der Mix-Kaskade durch den Sender. Hierbei wird nicht nur der Routing-Header sukzessive mit den öffentlichen Schlüsseln der gewählten Mixe verschlüsselt, sondern die gesamte Nachricht, die sowohl den Routing-Header als auch die verschlüsselten Nutzdaten beinhaltet (siehe Abbildung 4-7).

Dieses Verfahren verhindert ein Pattern Matching innerhalb der vom Sender bestimmten Teilkaskade. Eine Verfolgung der Nachricht anhand ihrer (abnehmenden) Größe kann durch Hop-to-Hop-Verschlüsselung mit Padding oder durch längentreue Umcodierung (siehe [Pfit97], [Chau81]) kombiniert mit Dummy-Verkehr erschwert werden.

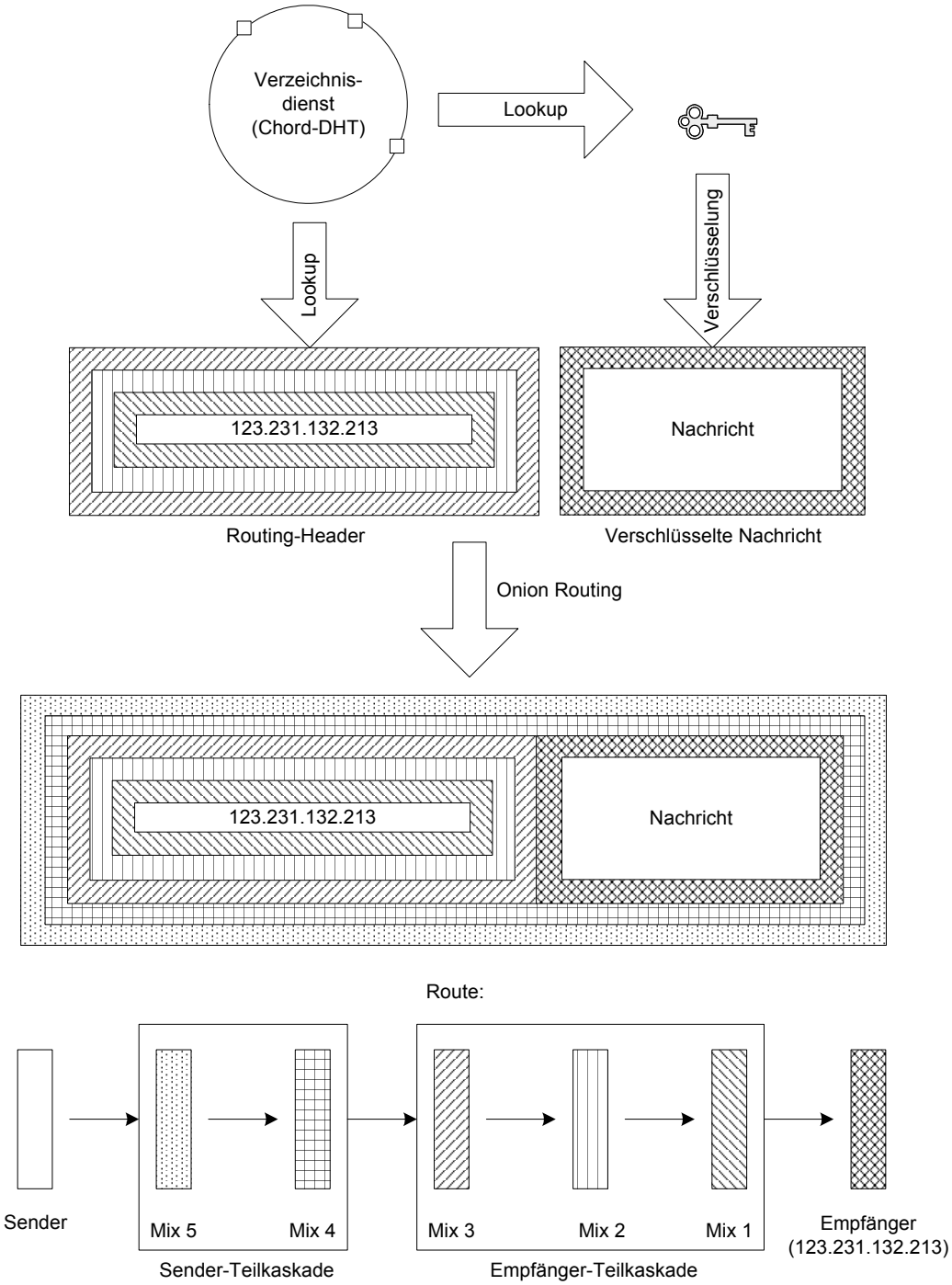


Abbildung 4-7: Kaskadenerweiterung mit Onion Routing

4.6 Mix-Informationen

Um eine Mix-Kaskade auswählen zu können, verwaltet jeder Agent eine *Mix-Informationsliste* (kurz: *MI-Liste*), in der ein Eintrag für jeden bekannten Mix enthalten ist. Zu den gespeicherten Informationen gehören die Adresse des Mixes und dessen öffentlicher Schlüssel.

Verfügbarkeit

Entschließt sich ein Agent dazu, den lokalen Mix-Dienst zu starten, um als Mix zu fungieren, so muss er seine Mix-Informationen den anderen Agenten verfügbar machen. Hierzu schreibt er sie in seine lokale MI-Liste, repliziert sie aber auch auf die nächsten `r` Nachfolger des lokalen Knotens im Chord-Ring. Um die Verfügbarkeit der Mix-Informationen zu erhöhen, wird dieser Vorgang in regelmäßigen Intervallen wiederholt.

Mix Discovery

Ein Agent kann auch aktiv nach neuen Mix-Informationen suchen. Hierzu wählt er regelmäßig einen zufälligen Knoten im Chord-Ring und fordert die Mix-Informationen aus dessen MI-Liste an. Erfährt er dadurch von neuen Mixen, so werden deren Informationen in der lokalen MI-Liste gespeichert. Dieser Vorgang wird als *Mix Discovery* bezeichnet. Ein Mix Discovery wird nur dann durchgeführt, falls die lokale MI-Liste nicht die erwünschte, einstellbare Länge hat – entweder weil der Agent noch nicht lange im Netz ist und aus diesem Grund die erwünschte Länge noch nicht erreicht wurde, weil erkannt wurde, dass einige der Mixe ausgefallen sind und die MI-Liste anschließend gesäubert wurde oder weil im Netz nicht genügend Mixe aktiv sind.

Die erwünschte Länge der MI-Liste sollte sorgfältig vom Benutzer des Agenten eingestellt werden. Die Anzahl der bekannten aktiven Mixe sollte mindestens so groß sein wie die Mindestanzahl der Mixe, die für die Erfüllung des erwünschten Anonymitätsgrades aller innerhalb des lokalen Agenten aktiven Pseudonyme notwendig ist. Falls der Benutzer entscheidet, dass pro Pseudonym mehrere Routing-Header erstellt werden, so ist es sinnvoll, dass die Länge der MI-Liste mindestens doppelt so groß wie die erwähnte Mindestanzahl ist, damit die Schnittmenge der in den Kaskaden verwendeten Mix-Mengen nicht allzu groß ist. In größeren Netzen mit vielen Mixen ist ein höherer Wert für die Länge durchaus sinnvoll, da dadurch eventuell disjunkte Mix-Kaskaden ermöglicht werden und dadurch die Verfügbarkeit der Pseudonyme höher ist. In kleineren Netzen verursacht ein zu hoher Wert jedoch unnötigen Datenverkehr, da regelmäßig erfolglose Discovery-Vorgänge durchgeführt werden.

Load-Wert

Einerseits kann jeder Agent entscheiden, ob er als Mix fungieren soll oder nicht. Andererseits wurde in Abschnitt 4.5 gezeigt, dass die Abschaltung des Mix-Dienstes ein Sicherheitsrisiko bezüglich der Anonymität darstellt.

Dieses Problem kann zum Beispiel dadurch gelöst werden, dass der Mix-Dienst zwar auf jedem Agenten aktiv ist, aber auf eintreffenden Mix-Discovery-Anfragen mit eigenen Mix-Informationen mit geringerer Wahrscheinlichkeit geantwortet wird. Eine Möglichkeit hierzu wäre die Einführung eines Werts, der in den Mix-Informationen mit aufgenommen wird und der die erwünschte Auslastung des jeweiligen Mixes beschreibt, ein so genannter *Load-Wert*. Die Mix-Informationsliste wird auf jedem Agenten absteigend nach diesem Wert sortiert und auf Discovery-Anfragen zuerst mit den Einträgen geantwortet, die oben in der Liste stehen.

In Abbildung 4-8 wird die MI-Liste des Knotens N_7 dargestellt. Knoten N_1 führt ein Mix Discovery durch. Hierbei wählt er als zufälligen Knoten N_7 aus. N_1 fordert von diesem vier Einträge an, darauf antwortet N_7 mit den ersten vier Einträgen der Liste.

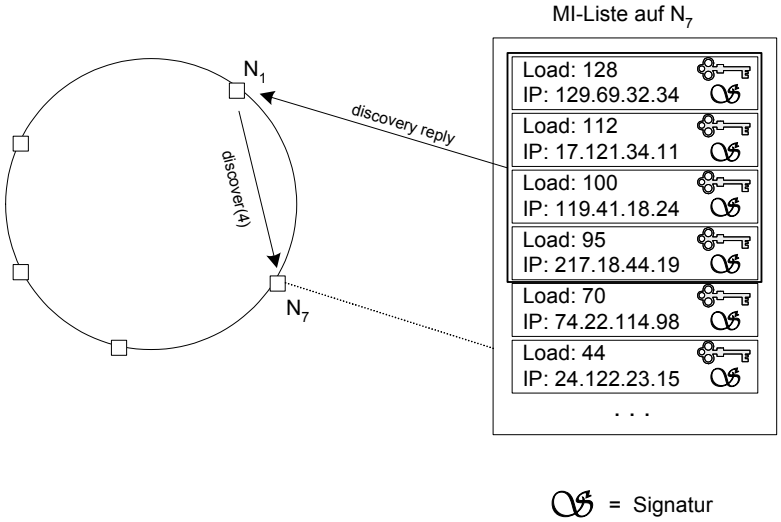


Abbildung 4-8: Mix Discovery und MI-Liste mit Load-Wert

Die Einführung des Load-Werts birgt die Gefahr eines Denial-of-Service-Angriffs auf Mixe, da der Wert auf einem Agenten geändert – etwa um ein Vielfaches erhöht – und bei Anfragen weitergegeben werden kann. Wird dieser erhöhte Wert im Netz verbreitet, so hat das die Erhöhung des eintreffenden Verkehrs am angegriffenen Mix zur Folge.

Eine regelmäßige Weitergabe der eigenen, richtigen Mix-Informationen kann hier nur bedingt Abhilfe schaffen. Eine zusätzliche Maßnahme kann jedoch ergriffen werden, die die Verwendung und Weitergabe von manipulierten Mix-Informationen verhindert, nämlich deren Signierung. Hierzu berechnet der Mix vor der Speicherung und Weitergabe der Informationen den Hashwert des Inhalts der Informationen, zu denen auch der oben erwähnte Wert gehört, und verschlüsselt diesen Wert mit seinem geheimen Schlüssel. Die dadurch entstandene digitale Unterschrift wird zusammen mit den Mix-Informationen weitergegeben. Da zu diesen Informationen auch der öffentliche Schlüssel des Mixes gehört, kann deren Richtigkeit jederzeit durch die Prüfung der Signatur untersucht werden.

Kapitel 5

Der Empfehlungsdienst

Nachdem die Struktur und Funktionsweise des UniTEC-Systems beschrieben wurde und die Gründe für die Verwendung eines Kommunikationssubsystems basierend auf einem anonymisierenden Peer-to-Peer-Netz dargelegt wurden, wurde im letzten Kapitel eine Möglichkeit zur Realisierung dieses Subsystems dargestellt und einige Erweiterungen vorgeschlagen, die die Sicherheit des Systems zusätzlich erhöhen. Der zu entwickelnde Empfehlungsdienst setzt auf der beschriebenen Kommunikationsschicht auf.

Dieses Kapitel beschreibt die Anforderungen an den Empfehlungsdienst und anschließend die Art und Weise, wie dieser realisiert und in das System integriert werden kann.

5.1 Anforderungen

Der Empfehlungsdienst hat eine Vielfalt von Aufgaben. Diese werden im Folgenden detailliert beschrieben.

Empfehlungen

Die primäre Aufgabe des Empfehlungsdienstes ist die Verwaltung von Empfehlungen. Dazu gehört das Erstellen, Speichern, Ändern und Löschen von Empfehlungen.

Die Struktur einer Empfehlung sollte bei deren Erstellung idealerweise völlig frei definierbar, später aber auch änderbar sein. Einige Datenfelder werden jedoch für das Funktionieren des Systems vorausgesetzt, diese werden im Folgenden beschrieben.

Eine Empfehlung wird entsprechend des empfohlenen Objekts in mindestens eine Kategorie eingestuft. Bei mehreren Kategorien wird pro Kategorie ein Prozentsatz angegeben. Dieser beschreibt, in welchem Maße die Empfehlung einer Kategorie zugehört. Der Wert fließt bei der Aktualisierung des Vertrauenswerts als Gewichtungsfaktor mit ein (siehe [Aich03]).

Der Grad der Überzeugung, mit der der Ersteller der Empfehlung zu seinen eigenen Aussagen steht, kann in einem Konfidenzwert angegeben werden. Dieser Wert kann variieren, je nachdem, ob das empfohlene Objekt nur flüchtig oder sehr detailliert analysiert wurde, bevor die Empfehlung abgegeben wurde. Auch dieser Wert wird als Gewichtungsfaktor innerhalb der Trust-Update-Algorithmen verwendet, so dass Empfehlungen mit geringerer Sicherheit weniger Einfluss auf die Aktualisierung des Vertrauens haben.

Um das Alter einer Empfehlung bestimmen zu können, ist die Speicherung eines Zeitstempels sinnvoll, der die Zeit der Erstellung enthält.

Empfehlungen werden zur Steigerung der Effizienz des Systems und der Verfügbarkeit nicht nur auf dem Agenten des Erstellers, sondern auch auf anderen Agenten zwischengespeichert werden. Um zu verhindern, dass die Empfehlungen dabei manipuliert werden, wird deren Inhalt nach der Erstellung mit Hilfe einer Signatur geschützt.

Anfragen

Um Empfehlungen im Netz zu finden, muss die Möglichkeit bestehen, Anfragen im Rahmen einer bestimmten Kategorie zu formulieren und in das Netz zu versenden.

Damit zum einen nicht das gesamte Netz mit Anfragenachrichten überflutet wird, und zum anderen in erster Linie diejenigen Pseudonyme befragt werden, von denen qualitativ hochwertige Antworten erwartet werden, wird eine Anfrage gezielt an eine begrenzte Menge von Pseudonymen – die Nachbarschaft – gesendet.

Die Nachbarschaft besteht einerseits aus Pseudonymen aus dem Vertrauensmodell, nämlich diejenigen, die in der Anfragekategorie am vertrauenswürdigsten sind. Andererseits werden aber auch Pseudonyme befragt, von denen dem Anfragenden anhand des Expertisemodells bekannt ist, dass sie in der angegebenen Kategorie Experten (Authorities) sind, und Pseudonyme, denen viele Experten bekannt sind (Hubs). Sendet man Anfragen an einen Hub, so ist die Wahrscheinlichkeit hoch, dass dieser die Anfrage an einen Experten weiterleitet. Zusätzlich werden Anfragen auch an eine bestimmte Anzahl an zufälligen Pseudonymen weitergeleitet. Hierdurch haben sowohl Pseudonyme, die anhand der im Expertise- und Vertrauensmodell gespeicherten Daten nicht in die Nachbarschaft kommen würden, als auch neue Pseudonyme, über die noch keine Daten erfasst wurden, die Chance, Anfragen zu beantworten und dadurch Vertrauen aufzubauen beziehungsweise sich als Experte oder als Hub auszuzeichnen. Damit letzteres funktioniert, muss ein Mechanismus realisiert werden, durch den neu erstellte Pseudonyme bekannt werden.

Die Anfragen enthalten einen Hop-Counter, der vor jeder Weiterleitung der Anfrage dekrementiert wird. Hat der Hop-Counter den Wert 0 noch nicht erreicht, so wird die Anfrage an die Nachbarschaft des Pseudonyms weitergeleitet, das die Anfrage empfangen hat, ansonsten wird sie verworfen. In jeden Fall wird versucht, die Anfrage zu beantworten.

Vor jeder Versendung der Anfrage an ein Pseudonym wird das Vertrauen diesem gegenüber im Kontext der Anfragekategorie zur Anfrage hinzugefügt, hierdurch entsteht die Vertrauenskette. Da diese in die Antworten kopiert und vom Anfragenden bei Erhalt der Antwort evaluiert wird, muss sie vor Änderungen geschützt sein (z.B. durch eine Signatur). Sie sollte keine Schleifen enthalten, daher sollte sie darauf überprüft werden, ob das nächste Pseudonym, an das die Anfrage gesendet wird, nicht bereits schon enthalten ist.

Antworten

Bei eintreffenden Anfragen wird das Empfehlungsmodell nach passenden Empfehlungen durchsucht und aus den passenden Ergebnissen wird eine Antwortnachricht erstellt, die an das anfragende Pseudonym gesendet wird.

Die Vertrauenskette wird aus der entsprechenden Anfrage in die Antwortnachricht kopiert, bevor diese an den Anfragenden gesendet wird. Zusätzlich wird die Information zur Überprüfung der Integrität der Vertrauenskette (etwa eine oder mehrere digitale Unterschriften) an die Nachricht drangehängt.

Antworten werden beim Anfragenden in einer Cache-Struktur zwischengespeichert, deren Größe und Vorhaltezeit von diesem konfiguriert werden kann. Sie werden später zum Beantworten eintreffender Anfragen benutzt; dieses sorgt für eine erhöhte Verfügbarkeit der Empfehlungen – ohne Zwischenspeichern wäre eine Empfehlung nur dann verfügbar, wenn ihr Ersteller mit dem Netz verbunden ist.

Zugriffskontrolle

Der Ersteller der Empfehlung soll deren Verbreitung auf Wunsch beschränken können. Er entscheidet, ob eine Beschränkung der Weitergabe stattfinden soll, und falls dieses der Fall ist, welche Pseudonyme für den Empfang der Empfehlung als Antwort auf eine Anfrage vorgesehen sind. Hierzu sind Zugriffskontrolllisten zu führen.

5.2 Lösungsentwurf

Dieser Abschnitt beschreibt die Art und Weise, wie die Anforderungen an den Empfehlungsdienst innerhalb dieser Arbeit erfüllt wurden.

Empfehlungsmodell

Wie bereits in Kapitel 2 ausgeführt, enthält das Empfehlungsmodell sowohl lokal erstellte als auch zwischengespeicherte Empfehlungen, die in Antworten auf Anfragen enthalten waren.

Der im Empfehlungsmodell enthaltene Datenbestand wird auf dem Agenten persistent gespeichert, damit er auch nach Absturz oder Beenden des Agenten erhalten bleibt. Die Empfehlungen müssen jedoch nicht nur gespeichert werden, es muss auch eine Möglichkeit geben, diese wieder ausfindig zu machen, um einerseits eintreffende Anfragen beantworten zu können, und andererseits um die Verwaltung der Empfehlungen zu ermöglichen.

Datenhaltung

Um Empfehlungen auf bequeme und intuitive Art und Weise wieder finden zu können, wird das Empfehlungsmodell in einer Datenbank gespeichert, die auf Anfrage eine Liste aller gespeicherten Empfehlungen zu Verwaltungszwecken zurückgibt, zusätzlich aber auch flexible Anfragen ermöglicht, um gezielt nach bestimmten Empfehlungen zu suchen. Da Empfehlungen sehr flexibel strukturiert werden können,

aber in der Regel Fließtext enthalten, ist die Ergänzung der Datenbank durch eine Möglichkeit zur Volltextsuche besonders sinnvoll.

Für viele kommerzielle Datenbanksysteme gibt es Erweiterungen, die eine effiziente Volltextsuche ermöglichen. Die damit verbundenen Lizenzkosten und Anforderungen an die Rechenleistung sind jedoch zu hoch für einen sinnvollen Einsatz in UniTEC.

Einige Open Source-Produkte (z.B. MySQL[1]) unterstützen Volltextanfragen, können dabei ohne Lizenzgebühren eingesetzt werden und benötigen in der Regel weniger Ressourcen als die kommerzielle Konkurrenz. Keines dieser Produkte bietet jedoch die Flexibilität, die für die Speicherung von Empfehlungen beliebiger Struktur notwendig ist, verbunden mit einer effizienten Datensuche. Daher wurde eigens hierfür eine Lösung zur Datenhaltung entwickelt, die in Kapitel 6 beschrieben wird.

Die zwischengespeicherten Empfehlungen, die als Antwort auf lokal formulierte Anfragen erhalten wurden, werden innerhalb der Datenbank in einem persistenten Cache gespeichert. Ein persistenter Cache ist ein Datencontainer im nichtflüchtigen Speicher mit begrenzter Größe und einer Ersetzungsstrategie, die bestimmt, welche Einträge bei Überschreiten der Maximalgröße überschrieben werden. Eine sinnvolle Möglichkeit hierfür wäre die Ersetzung derjenigen Einträge, auf die in letzter Zeit nicht zugegriffen wurde (Least Recently Used-Strategie, LRU). Denkbar wäre stattdessen auch die Ersetzung der Einträge, auf die am seltensten zugegriffen wurde (Least Frequently Used-Strategie, LFU).

Struktur der Empfehlungen

Wie bereits erwähnt, sollten Empfehlungen idealerweise eine beliebige Struktur haben können. Um dieses zu ermöglichen, kann der Benutzer eine beliebige Anzahl an Feldern definieren. Jedes Feld wird durch drei Eigenschaften beschrieben: der *Feldname*, der *Feldtyp* und der *Feldinhalt*.

Der Feldtyp beschreibt den Typ des Feldinhalts, also der Daten, die in dem Feld gespeichert werden können. Zu diesen zählen folgende:

- Fließtext
- Zahlen
- Boolesche Elemente
- Binäre Objekte

Fließtext-Felder enthalten alphanumerische Zeichenketten beliebiger Länge. Ein typisches Beispiel für ein solches Feld ist eine textuelle Bewertung, es kann aber auch beliebige andere Informationen enthalten, wie zum Beispiel den Namen des Herstellers eines empfohlenen Objekts.

Bei Zahlen wird zwischen ganzen und reellen Zahlen unterschieden. Ganze Zahlen können beispielsweise für eine Benotung verwendet werden – hierbei könnte man eine Skala zwischen eins und zehn verwenden. Reelle Zahlen werden zum Beispiel

[1] http://www.mysql.com

bei der Angabe eines Preises benötigt, aber auch für andere Werte wie die Dauer der Nutzung des Produkts durch den Empfehlenden (z.B. 2,5 Jahre) oder prozentuelle Angaben wie etwa die Zufriedenheit mit einem Produkt.

Boolesche Elemente können zur Beschreibung von Eigenschaften mit genau zwei Zuständen verwendet werden, zum Beispiel, ob ein Anbieter oder ein Produkt empfehlenswert ist oder ob der Empfehlende selbst das Produkt besitzt. Speziell auf eine Kategorie bezogen können boolesche Elemente konkrete Eigenschaften eines Produkts beschreiben, zum Beispiel, ob eine Digitalkamera ein optisches Zoomobjektiv besitzt oder ob ein LCD-Monitor mit einem digitalen Eingang ausgestattet ist.

Binäre Objekte können binäre Repräsentationen beliebiger Daten enthalten. Typische Vertreter solcher Daten sind Bilder von empfohlenen Produkten, aber auch die Verwendung anderer multimedialer Ressourcen ist denkbar, wie etwa Video- oder Audiodaten.

Jedes Feld hat einen Namen, der innerhalb eines Feldtyps eindeutig ist. Kennt man die Struktur der Empfehlung und damit den Namen der Felder, so kann man gezielt nach dem Inhalt eines bestimmten Feldes suchen, indem bei der Formulierung der Volltextanfrage der Name des Feldes angegeben wird.

Anfragen

Möchte ein Benutzer nach einer bestimmten Empfehlung suchen, so formuliert er eine Anfrage, die entsprechende Suchkriterien enthält, und sendet diese an die Nachbarschaft. Diese sendet die Anfrage rekursiv weiter, bis die maximale Anzahl der Hops erreicht ist, die im Hop-Counter definiert wurde. Vor jedem Versenden der Anfrage wird das Vertrauen gegenüber dem nächsten Pseudonym eingefügt zusammen mit einer Signatur, die den Vertrauenswert schützt. Jedes Pseudonym, das eine Anfrage empfängt, versucht, diese zu beantworten, und dann wird die Anfrage gegebenenfalls weitergeleitet. Diese Schritte werden im Folgenden genauer beschrieben.

Formulierung von Anfragen

Der Benutzer formuliert die Suchkriterien der Anfrage als eine Volltextanfrage. Diese kann mehrere Elemente enthalten, die durch boolesche Operatoren verknüpft sind. Ein Anfrageelement enthält ein einzelnes Suchkriterium, das sich auf den Inhalt der gesamten Empfehlung, aber auch auf den einzelner Felder beziehen kann.

Zum Beispiel kann man durch ein Suchkriterium angeben, dass man nach einer Empfehlung sucht, die das Wort „Digitalkamera“ enthält. Bezieht sich das Suchkriterium auf ein einzelnes Feld, so kann man dadurch beispielsweise nach einer Empfehlung suchen, die im Feld namens „Beschreibung“ die Wortfolge „optischer Sucher“ beinhaltet. Die Anfrageelemente, die diese beiden Suchkriterien enthalten, kann man mit einem booleschen Operator verbinden. Dieses hat beim „AND“-Operator die Folge, dass bei der Ermittlung der Ergebnisse die Schnittmenge der Ergebnismengen gebildet wird, die von den einzelnen Suchkriterien geliefert werden. Bei „OR“ wird hingegen die Vereinigungsmenge gebildet.

Die Anfrage aus obigem Beispiel sieht folgendermaßen aus:

```
Digitalkamera AND Beschreibung:"optischer Sucher"
```

Auf die Syntax von Volltextanfragen wird in Kapitel 6 näher eingegangen.

Anfragen werden im Kontext einer einzelnen Kategorie formuliert, da die Zusammensetzung der Nachbarschaft von der Kategorie stark abhängt. Sowohl die Menge der vertrauenswürdigen Pseudonyme, als auch die der Authorities und der Hubs sind mit hoher Wahrscheinlichkeit für jede Kategorie verschieden. Für die Anfragekategorie wird implizit ein Anfrageelement erstellt, das anschließend durch den booleschen Operator „AND" mit den restlichen Anfrageelementen verknüpft wird. Bei lokalen Anfragen, die zur Verwaltung der gespeicherten Empfehlungen verwendet werden, entfällt dieses, da diese direkt an die Datenbank gesendet werden und dadurch keine kategorienspezifische Nachbarschaft verwendet werden muss.

Unter Umständen ist auch die Formulierung einer Anfrage im Kontext mehrerer Kategorien sinnvoll. Dieses kann ermöglicht werden, indem intern mehrere Anfragenachrichten versendet werden – eine Anfrage für jede Kategorie.

Sinnvoll ist eine Angabe der maximalen Anzahl Treffer, die von einem Pseudonym in einer Antwortnachricht an den Anfragenden gesendet werden sollen. Hohe Werte dieses Parameters bei gleichzeitiger hoher Anzahl von Treffern (etwa bei trivialen Anfragen) können jedoch sehr hohen eintreffenden Verkehr beim anfragenden Agenten verursachen, daher ist bei der Verwendung dieses Parameters Vorsicht geboten; er sollte nur beim Erreichen unbefriedigender Ergebnisse erhöht werden.

Vertrauenskette

Bevor eine Anfrage an ein Pseudonym versendet wird, wird der Wert des Vertrauens gegenüber diesem Pseudonym dem Vertrauensmodell entnommen und in die Anfrage eingefügt, zusammen mit der ID des Pseudonyms. Ohne weitere Maßnahmen könnten diese Informationen jedoch von anderen Pseudonymen verändert werden. Aus diesem Grund wird nach jedem Hinzufügen des Vertrauenswerts eine Signatur erstellt. Die Entstehung der Signaturen ist in Abbildung 5-1 dargestellt.

Jede Zeile enthält die Daten, die vor dem Versenden zur Vertrauenskette hinzugefügt werden: die ID des nächsten Pseudonyms in der Kette, das Vertrauen gegenüber diesem, der eigene öffentliche Schlüssel und eine Signatur über die ID und den Vertrauenswert. Manipulationen an einzelnen Feldern werden durch die Überprüfung der Signatur erkannt.

Das Austauschen ganzer Zeilen kann auch erkannt werden, indem der Hash des Schlüssels aus jeder Zeile mit der ID aus der vorhergehenden Zeile verglichen wird. Im letzten Kapitel wurde darauf eingegangen, dass die Generierung der ID eines Pseudonyms durch Berechnung des Hashwerts aus dem öffentlichen Schlüssel dieses Pseudonyms eine sichere Methode zur Zuordnung eines Schlüssels zu einem Pseudonym darstellt.

Möchte ein Angreifer eine Zeile austauschen, so muss er alle vorhergehenden Zeilen auch austauschen. Dieses wird jedoch vom Anfragenden erkannt, da der öffentliche Schlüssel aus der ersten Zeile identisch mit seinem eigenen Schlüssel, bzw. der Hash

des Schlüssels identisch mit der eigenen ID sein muss. Dieses ist bei Austauschen der ersten Zeile sicherlich nicht mehr der Fall, so dass eine Manipulation erkannt und die Antwort verworfen werden kann.

ID_1	t(0; 1)	Key_0	Sig_1
ID_2	t(1; 2)	Key_1	Sig_2
ID_3	t(2; 3)	Key_2	Sig_3
ID_4	t(3; 4)	Key_3	Sig_4

$Sig_n = Sig(ID_n + t(n-1; n))_n$, $n>0$

ID_n - ID des Pseudonyms n (0=Anfragender)
t(m; n) - Vertrauen von m in n
Key_n - öffentlicher Schlüssel von n
Sig_n - n-te Signatur in der Kette
$Sig(x)_n$ - Signatur des Inhalts x mit Hilfe des öffentlichen Schlüssels von n

Abbildung 5-1: Signierung der Vertrauenskette

Die Vertrauenskette enthält IDs von Pseudonymen, die möglicherweise nicht alle bekannt sind. Nach dem Empfang einer Anfrage kann ein Pseudonym die Gültigkeit der Vertrauenskette prüfen und anschließend die in der Kette enthaltenen, ihm noch nicht bekannten Pseudonyme in einer Liste aufnehmen. Diese Liste kann später bei der Berechnung der Nachbarschaft zur Auswahl von zufälligen Pseudonymen verwendet werden.

Weiterleitung von Anfragen

Sowohl lokal erstellte als auch eintreffende Anfragen werden an die Nachbarschaft gesendet, falls der Hop-Counter noch nicht den Wert 0 erreicht hat. Vor dem Versenden der Anfrage an die Nachbarschaft wird dieser dekrementiert.

Die Zusammensetzung der Nachbarschaft wird von mehreren Parametern bestimmt. Die Nachbarschaftsgröße bestimmt die Anzahl der Pseudonyme, die in der Nachbarschaft enthalten ist. Jede der vier Pseudonymklassen (d.h. Authorities, Hubs, vertrauenswürdige oder zufällige Pseudonyme) wird durch einen Parameter beschrieben, der den Anteil der Klasse an der Nachbarschaft darstellt. Falls alle Klassen gleich groß sein sollen, so nimmt dieser Parameter den Wert ¼ an. Die Angabe von vier Parametern hierfür macht keinen Sinn, da der vierte Parameter aus der Subtraktion der anderen drei vom Wert 1 berechnet werden kann. Hiermit hat man also vier Parameter – einen für die Größe und drei für die Anteile – anhand derer man die Nachbarschaft jederzeit neu berechnen kann. Diese Parameter sind innerhalb des Agenten für jedes Pseudonym individuell einstellbar.

Damit Änderungen an den Datenmodellen und an den Parametern der Nachbarschaft sich so bald wie möglich an der Zusammensetzung der Nachbarschaft auswirken, wird die Nachbarschaft vor jeder Weiterleitung neu berechnet. Da die Nachbarschaft kategorienspezifisch ist, wäre eine Zwischenspeicherung im Gegensatz zu einer wiederholten Neuberechnung mit hohem Speicherplatzbedarf verbunden. Der

rechenaufwendigere Teil der Neuberechnung besteht in der Sortierung der Datenmodelle: das Vertrauensmodell nach Vertrauenswerten und das Expertisemodell nach dem Authority- bzw. Hub-Rating. Durch den Einsatz effizienter Sortieralgorithmen kann die Sortierzeit jedoch minimiert werden..

Die wiederholte Neuberechnung der Nachbarschaft hat zudem noch einen weiteren Vorteil. Damit sich innerhalb einer Vertrauenskette keine Schleifen bilden, sollte die Anfrage nicht an Pseudonyme gesendet werden, die bereits in der Vertrauenskette enthalten sind. Diese Pseudonyme können in einer Liste an die Routine übergeben werden, die für die Berechnung der Nachbarschaft verantwortlich ist. Diese Routine sorgt dafür, dass in der neu berechneten Nachbarschaft diese Pseudonyme nicht vorkommen.

Erreichbarkeit

Unter Umständen ist es sinnvoll, Statistiken über die Erreichbarkeit der Pseudonyme zu erstellen und diese eventuell auch bei der Berechnung der Nachbarschaft zu berücksichtigen.

Hierzu wurde eine *Erreichbarkeitsmetrik* eingeführt (engl. *reachability*). Pro Pseudonym wird ein Erreichbarkeitswert gespeichert, der den Startwert 0 hat und automatisch inkrementiert beziehungsweise dekrementiert wird, je nachdem, ob das Pseudonym erreicht werden konnte.

Anhand der Erreichbarkeit kann entschieden werden, ob das Pseudonym in die Nachbarschaft aufgenommen werden soll. Denkbar wäre auch eine Implementierung, die dafür sorgt, dass Anfragen seltener an Pseudonyme, die nur schwer erreichbar sind, versendet werden. Dieses hätte den Vorteil, dass Pseudonyme trotz einer geringen Erreichbarkeit die Chance haben, auf Anfragen zu antworten und gleichzeitig ihre Erreichbarkeit zu erhöhen. Möglich wäre die Verwendung des Erreichbarkeitswerts als Gewichtung bei der Sortierung der Datenmodelle zur Erstellung der Nachbarschaft.

Sinnvoll wäre eventuell auch die Einführung einer fünften Pseudonym-Klasse innerhalb der Nachbarschaft, die besonders gut erreichbare Pseudonyme enthält.

Weitere Details zur Erreichbarkeitsmetrik sind in [Widm03] enthalten.

Antworten

Empfängt ein Pseudonym eine Anfrage, so versucht es, diese zu beantworten. Hierzu verwendet es in erster Linie die selbst erstellten Empfehlungen aus dem Empfehlungsmodell. Kann es die Anfrage damit nicht beantworten, etwa weil keine passenden Empfehlungen unter diesem Pseudonym erstellt wurden, oder weil die Zugriffskontrolliste (siehe Seite 53) die Weitergabe passender Empfehlungen an den Anfragenden verbietet, so wird der Cache-Teil des Empfehlungsmodells gesucht, der zwischengespeicherte Antworten enthält. Die Antwort enthält schließlich höchstens diejenige Anzahl an Treffern, die vom Anfragenden als maximal erwünscht angegeben wurde.

Sicherheitsmaßnahmen

Durch die rekursive Weitergabe von Anfragen wird je nach Wahl der Nachbarschaftsgröße und des Hop-Counters möglicherweise eine sehr hohe Anzahl an Pseudonymen erreicht. Da jedes erreichte Pseudonym möglicherweise eine Antwort formuliert, kann man durch Manipulieren einer Anfrage eine gezielte Denial of Service-Attacke auf ein bestimmtes Pseudonym durchführen. Hierbei stellt man einen hohen Wert für den Hop-Counter ein und setzt die ID des Anfragenden auf die ID des anzugreifenden Pseudonyms. Enthält der Anfrage-String eine triviale Anfrage (zum Beispiel nach einem Wort, das häufig vorkommt), und die maximal erwünschte Trefferanzahl ist sehr hoch, so wird das angegriffene Pseudonym unter Umständen mit einer riesigen Menge großer Nachrichten bombardiert.

Prüft ein antwortendes Pseudonym nach, ob der Hash des ersten öffentlichen Schlüssels aus der Vertrauenskette ($\mathtt{Key}_0$ in Abbildung 5-1) identisch mit der ID des Anfragenden ist, so werden die Möglichkeiten der beschriebenen DoS-Attacke etwas eingeschränkt, da der Angreifer eine gültige Vertrauenskette benötigt, die beim anzugreifenden Pseudonym beginnt.

Empfängt er jedoch eine Anfrage von diesem Pseudonym, so ist eine Attacke doch noch möglich, und zwar durch die Änderung des Anfrage-Strings, des Hop-Counters und der Trefferanzahl. Um dieses zu verhindern, ist es empfehlenswert, dass der Anfragende den Inhalt dieser drei Felder signiert. Der Schlüssel des Anfragenden ist in der Vertrauenskette enthalten (und kann nicht ausgetauscht werden, ohne dass dieses erkannt wird, siehe hierzu Abschnitt „Vertrauenskette“, Seite 50), so dass die Signatur vor der Versendung einer Antwortnachricht überprüft und gegebenenfalls eine manipulierte Anfrage verworfen werden kann.

Sinnvoll ist eventuell auch ein Anfrageparameter, über den der Anfragende entscheiden kann, ob auch zwischengespeicherte Nachrichten bei der Beantwortung berücksichtigt werden, oder nur selbst erstellte. Letzteres würde einerseits dazu führen, dass nicht so viele Duplikate in den Antworten enthalten sind, jedoch würde man auf diese Art und Weise manche Empfehlungen möglicherweise gar nicht erreichen.

Zugriffskontrolle

Der Ersteller einer Empfehlung hat eventuell die Absicht, den Kreis derjenigen Pseudonyme einzuschränken, die die erstellte Empfehlung als Antwort auf Anfragen empfangen. Diese Aufgabe ist ansatzweise dadurch lösbar, indem zu der Empfehlung eine *Zugriffskontrollliste* (engl. *access control list*, *ACL*) hinzugefügt wird. Die Signatur der Empfehlung erstreckt sich auch über diese Listen, damit sie später nicht geändert werden können. Die Liste wird bei der Erstellung der Antwortnachrichten berücksichtigt, so dass unerwünschte Pseudonyme die Empfehlung nicht erhalten.

Je nach Empfehlung kann der Kreis der Pseudonyme, für die die Empfehlung bestimmt ist, unendlich, sehr groß oder sehr klein sein. Um die Auswahl dieses Kreises so flexibel wie möglich zu gestalten, enthält die Zugriffskontrollliste zwei Teile: die *Allow-ACL* und die *Deny-ACL*. Die Allow-ACL enthält Pseudonyme, die die Empfehlung erhalten dürfen, während den Pseudonymen in der Deny-ACL der Zugriff auf die Empfehlung verwehrt wird.

Wird keine Zugriffskontrolle durchgeführt, so sind sowohl die Allow-ACL als auch die Deny-ACL leer. Enthält die Allow-ACL Pseudonyme, so haben nur diese Zugriff, und die Deny-ACL wird ignoriert. Ist die Allow-ACL leer, so wird die Deny-ACL berücksichtigt – die hier enthaltenen Pseudonyme haben keinen Zugriff. Ist auch diese leer, so kann die Empfehlung an beliebige Pseudonyme versendet werden. Hierdurch kann aber auch die generelle Verwehrung des Zugriffs auf die Empfehlung erreicht werden (z.B. weil die Empfehlung noch nicht fertig gestellt wurde). Hierzu nimmt der Ersteller der Empfehlung sein eigenes Pseudonym in die Allow-ACL auf.

Das beschriebene Verfahren bietet natürlich keine absolute Sicherheit, da durch eine Änderung der Software die Weitergabe trotz verbietender Zugriffskontrollliste ermöglicht werden kann. Der Inhalt der Empfehlung kann natürlich auch kopiert werden, dieses Problem ist jedoch generell nicht lösbar.

Um dem Ersteller die Verwaltung der Zugriffskontrolllisten zu erleichtern, kann ein Gruppenmanagement-Subsystem entwickelt werden. Dieses wurde innerhalb des Identitätsmanagers realisiert. Details hierzu kann man [Widm03] entnehmen.

Empfang von Antwortnachrichten

Wie in Kapitel 3 beschrieben, hat der Benutzer des Systems die Möglichkeit, für jedes Pseudonym einen beliebigen Anonymitätsgrad zu wählen. Dieser beeinflusst jedoch die Latenz der Kommunikation in erheblichem Maße.

Je nach dem gewählten Anonymitätsgrad der betroffenen Kommunikationspartner und nach dem bei der Formulierung der Anfrage eingestellten Wert für den Hop-Counter variiert die Zeitspanne zwischen der Formulierung der Anfrage und dem Erhalt einer Antwort auf diese Anfrage sehr stark. Ferner hängt diese Zeitspanne auch von der Qualität der Netzwerkverbindung, der Zuverlässigkeit der verwendeten Mixe, und nicht zuletzt vom Umfang der Empfehlungen ab.

Der Anfragende möchte jedoch häufig nur eine begrenzte Zeit lang auf Antworten warten. Hat er eine Empfehlung erhalten, die seinen Erwartungen entspricht, so möchte er eventuell keine weiteren Antworten auf seine Anfrage erhalten, da diese zwischengespeichert werden und dadurch wertvollen Speicherplatz verbrauchen.

Sinnvoll wäre auch die Möglichkeit einer Zeitangabe, etwa in Sekunden, nach deren Ablauf weitere Antworten verworfen werden.

Hierzu kann ein Datencontainer verwendet werden, der zwei Zustände hat: offen und geschlossen. In offenem Zustand akzeptiert er Daten und speichert diese, geschlossen nimmt er hingegen keine neuen Daten mehr an. Pro Anfrage wird ein solcher Container generiert und geöffnet. Dieser bleibt für eine gewisse Zeit offen, danach wird er geschlossen. Er kann jedoch auch durch Benutzerinteraktion geschlossen werden, oder wenn eine bestimmte Datenmenge erreicht wurde. Nachdem er geschlossen wurde, werden die enthaltenen Daten im Empfehlungsmodell zwischengespeichert, können bewertet und zur Beantwortung eintreffender Anfragen verwendet werden.

Anhand der eintreffenden Antwortnachrichten und der darin enthaltenen Vertrauensketten werden die Statistikdaten des Expertisemodells aktualisiert. Basierend auf diesen Daten können die Authority- und Hub-Ratings der einzelnen Pseudonyme im Kontext der Anfragekategorie errechnet werden. Details zur Realisierung der Expertise-Aktualisierung werden in [Widm03] beschrieben.

Interessensgebiete

Manchmal kann es bei der Formulierung einer Anfrage schwierig sein, sich für die Auswahl einer Kategorie zu entscheiden, vor allem dann, wenn die Kategorien fein gegliedert sind. In diesem Fall könnte die Zusammenfassung von Kategorien in Interessensgebieten hilfreich bei der Formulierung von Anfragen sein.

Abbildung 5-2 zeigt eine mögliche Zuordnung von Kategorien zu Interessensgebieten. Hier erkennt man, dass verschiedene Interessensgebiete nicht disjunkt sein müssen: sie können verschachtelt sein, können sich aber auch überschneiden. Möglich ist die Definierung spezieller Gebiete wie „Data Storage“, aber auch generellere wie etwa „Computer Science“, die eine Reihe anderer Gebiete umfasst.

Eine Zuordnung von Kategorien zu Interessensgebieten kann vorgegeben werden, sinnvoll ist es jedoch auch, dass jeder Benutzer die Zuordnung nach Belieben ändern kann.

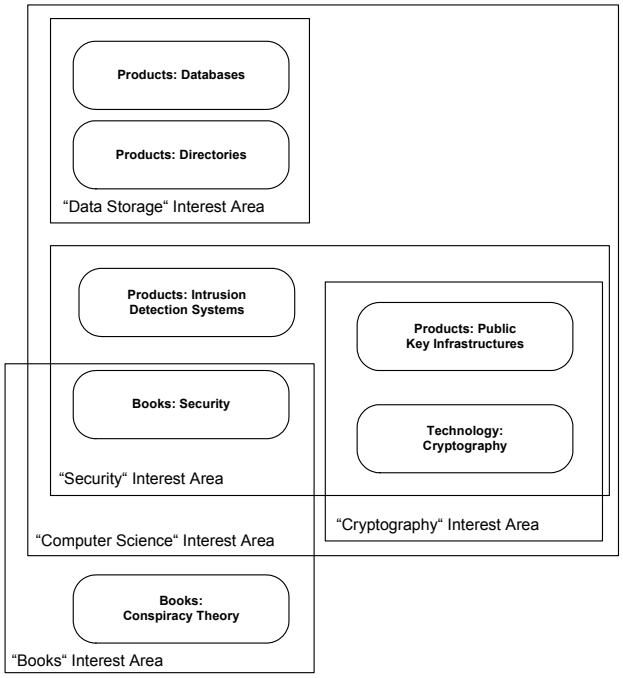


Abbildung 5-2: Interessensgebiete

Intern ändert sich durch die Einführung der Interessensgebiete nichts, da eine Anfrage auf einem Interessensgebiet mittels mehrerer Anfragen in jeweils den Kategorien realisiert wird, die das Interessensgebiet umfasst.

Bei der Verwendung von Interessensgebieten ist jedoch Vorsicht geboten. Da jede Anfrage auf mehrere konventionelle Anfragen abgebildet wird, kann allein schon durch das Versenden von Anfragen sehr viel Verkehr erzeugt werden. Werden zu

den Anfragen viele Treffer gefunden, so kann das bei einem generellen Interessensgebiet, das viele Kategorien umfasst, leicht zur Überlastung des Netzes führen. Somit ist die Einführung von Interessensgebieten einerseits von Nutzen bei der Formulierung von Anfragen, andererseits liefert es Angreifern ein leicht zu bedienendes Werkzeug, das unter Umständen zur Durchführung einer Denial of Service-Attacke auf das Kommunikationssubsystem verwendet werden kann.

Zudem besteht durch die Formulierung einer Anfrage in einem Interessensgebiet die Gefahr, dass dadurch zur gleichen Zeit Anfragen im Kontext verschiedener Pseudonyme des anfragenden Benutzers versendet werden. Dieses tritt auf, falls ein Interessensgebiet Kategorien enthält, für die verschiedene Pseudonyme eines Benutzers zuständig sind. Führt ein Benutzer eine statistische Analyse anhand eintreffender Anfragen, des Zeitpunkts, zu dem diese eintreffen und der Pseudonyme, in deren Kontext sie formuliert wurden, so kann der Angreifer unter Umständen verschiedene Pseudonyme eines einzigen Benutzers einander zuordnen. Diese Analyse kann erschwert werden, indem die Anfragen in zufälliger Reihenfolge und zufälligem zeitlichem Abstand versendet werden. Diese Maßnahmen erhöhen die Sicherheit, jedoch auch die Wartezeit. Tiefere Einblicke in diese Thematik können [Widm03] entnommen werden.

Kapitel 6

Realisierung

Dieses Kapitel beschreibt zum einen die Technologien, die bei der Realisierung eingesetzt wurden, es geht aber auch auf einige Implementierungsdetails ein. Die genaue Klassenstruktur des entstandenen Prototyps wird im Anhang beschrieben.

6.1 Verwendete Technologien

Die Entwicklung des Prototyps basiert auf einigen grundlegenden Technologien. Dieser Abschnitt beschreibt kurz deren wichtigste Eigenschaften und geht auf die Softwarekomponenten ein, mit deren Hilfe diese Technologien eingesetzt wurden.

XML

Sowohl bei der Speicherung von Daten als auch bei der Kommunikation wurde die standardisierte, plattformunabhängige Datenbeschreibungssprache XML (Extensible Markup Language) verwendet. Insbesondere in Java-Umgebungen können mit Hilfe von XML Daten beliebiger Struktur mit geringem Aufwand so aufbereitet werden, dass sie plattformübergreifend ausgetauscht und gespeichert werden können (siehe [Haro02]).

XML-Serialisierung

Um Java-Objekte aus dem Objektmodell eines Systems auf XML-Strukturen abzubilden, und umgekehrt, um XML-Dokumente in das Objektmodell einzulesen, kann man eine der Implementierungen des standardisierten Document Object Model (DOM) verwenden. DOM ist zum einen plattform- und programmiersprachenneutral, zum anderen ergibt sich jedoch daraus dessen wesentlicher Nachteil: da die DOM-API nicht speziell auf Java ausgelegt ist, ist die Nutzung von DOM mit viel Aufwand verbunden. Abhilfe schaffen teilweise Java-spezifische Erweiterungen von DOM (etwa JDOM oder dom4j) (siehe hierzu [Haro02]).

Eine andere Möglichkeit, die die Abbildung von Objekten auf XML mit wesentlich geringerem Aufwand realisiert, ist die *XML-Serialisierung*. Hierbei wird das betreffende Objekt analysiert und anhand dieser Analyse wird ein XML-Dokument erstellt, dessen Struktur und Inhalt denen des Objekts entspricht. Nutzt man hingegen für die abzubildenden Objekte das JavaBeans-Komponentenmodell (spezifiziert in [Hami97]), und verwendet man ein JavaBeans-Serialisierungsframework wie etwa

JOX[6], so kann man durch die Definition der Eigenschaften der JavaBean entscheiden, welche Daten auf die XML-Struktur abgebildet werden. Zusätzlich kann bei JOX durch die Angabe einer DTD (Document Type Definition) das XML-Dokument angepasst werden – etwa die Namen der entstehenden XML-Tags, die standardmäßig den Namen der Eigenschaften der zu serialisierenden JavaBean entsprechen.

XML-Datenbanken

XML-Dokumente können nach deren Generierung persistent gespeichert werden, etwa in einer Datei, oder als String bzw. BLOB (Binary Large Object) in einer relationalen Datenbank. Beide Möglichkeiten haben einen entscheidenden Nachteil: bei der Speicherung wird die Struktur des Dokuments nicht berücksichtigt. Eine Suche nach einem bestimmten Element muss sequentiell durchgeführt werden und ist in einem großen Datenbestand sehr aufwendig.

Der Suchaufwand kann verringert werden, wenn bei der Speicherung eine Indizierung durchgeführt wird. Hierbei werden die zu speichernden Daten aufbereitet und in einer Datenstruktur abgelegt, die eine effiziente Suche ermöglicht – dem Index.

XML-Datenbanken wie Apache Xindice[7] übernehmen die Speicherung von XML-Dokumenten. Die meisten implementieren die XML:DB-API-Spezifikation[8] und ermöglichen eine effiziente Suche über den gespeicherten Datenbestand mittels der standardisierten Abfragesprache *XPath*. Eine detaillierte Einführung in diese Sprache ist in [Haro02] enthalten.

Volltextsuche

Der Empfehlungsdienst setzt einen Mechanismus voraus, der die Formulierung flexibler, intuitiver Anfragen ermöglicht, mit deren Hilfe Empfehlungen beliebiger Struktur im Empfehlungsmodell ausfindig gemacht werden können. XPath-Anfragen bieten hierzu nicht die erwünschte Mächtigkeit, da sie die Kenntnis der Struktur der gespeicherten Dokumente voraussetzen (vgl. [Haro02]).

Ein Mechanismus hingegen, der eine Volltextsuche über den Datenbestand ermöglicht, führt durch einfache Anfragen schnell zum erwünschten Ergebnis, ohne die Dokumentenstruktur bei der Anfrage berücksichtigen zu müssen. Diese Art von Mechanismus wird von bekannten Internet-Suchmaschinen wie etwa Google[9] verwendet. Sie setzt die Aufbereitung der Inhalte und die Erstellung von Volltextindizes voraus.

Lucene[10] ist ein Open Source-Framework, das im Rahmen des Apache Jakarta-Projekts[11] entwickelt wurde, eines der ambitioniertesten Java-Projekte im Open Source-Bereich, dessen Ziel die Entwicklung verschiedener flexibler Java-Komponenten ist. Diese können in eigenen Java-Projekten ohne Lizenzkosten eingesetzt werden.

---

[6] http://www.wutka.com/jox.html
[7] http://xml.apache.org/xindice
[8] http://www.xmldb.org/xapi
[9] http://www.google.de
[10] http://jakarta.apache.org/lucene
[11] http://jakarta.apache.org

Das Lucene-Framework ermöglicht die Indizierung von Dokumenten und eine anschließende Volltextsuche über die indizierten Inhalte. Die Anfragen können zum einen ohne Berücksichtigung der Dokumentenstruktur formuliert werden, sie können jedoch durch die Angabe von Dokumentenfeldern verfeinert werden, falls die Struktur bekannt ist. Hierdurch können bei der Suche bessere Ergebnisse erzielt werden.

Das Framework bietet eine hohe Flexibilität bei den Indizierungsoptionen. Es können verschiedene *Analyzer* verwendet werden, mit deren Hilfe der in den indizierten Dokumententen enthaltene Text analysiert, in *Tokens* (Wörtern) aufgespaltet und von *Stopwörtern* – Wörtern mit wenig Bedeutung für die Anfragen wie etwa „und", „aber", „doch" – befreit wird. *Stemmer* werden zur Ermittlung der Stammform von Wörtern verwendet, um sowohl bei der Indizierung, als auch bei der Suche zwischen verschiedenen Flexionsformen eines Wortes (z.B. „kaufen", „Käufer") nicht unterscheiden zu müssen. Lucene bietet außerdem eine sehr flexible Formulierung von Anfragen, deren Syntax im Anhang (Abschnitt B) beschrieben wird.

Bei Bedarf können neue Analyzer entwickelt oder vorhandene erweitert werden, die zum Beispiel eine eigene Liste von Stopwörtern berücksichtigen oder komplexeres Stemming beherrschen. Diese Flexibilität ist vor allem bei der Verwendung von verschiedenen Sprachen notwendig, da sich deren Eigenschaften meist sehr stark unterscheiden.

Eine detaillierte Evaluation des Lucene-Frameworks sowohl in Hinsicht auf die Anwendung, als auch dessen Architektur und Erweiterungsmöglichkeiten betreffend, ist in [Weni03] enthalten.

Remote Procedure Calls

Bei einem Remote Procedure Call (kurz: *RPC*) werden Methoden auf entfernten Knoten aufgerufen. Zusammen mit dem Aufruf können Parameter an die entfernten Methoden übergeben werden, und der Aufruf kann ein Ergebnis zurückliefern.

Die Implementierung der Chord-DHT (siehe Kapitel 3, Abschnitt 3.3) wird durch die Verwendung eines RPC-Mechanismus wesentlich erleichtert (siehe [SMK+01]). Bei der Auswahl eines geeigneten RPC-Mechanismus wurden CORBA, RMI, SOAP und XML-RPC evaluiert. Die wichtigsten Kriterien hierbei waren leichte, jedoch flexible Handhabung, geringer Einarbeitungsaufwand, begrenzter Kommunikations-Overhead und reduzierter Ressourcenverbrauch. Während der Evaluierung erwies sich XML-RPC und speziell die Apache XML-RPC-Implementierung[12] als leichtgewichtiges, leicht einzusetzendes RPC-Protokoll, das zudem sowohl synchrone als auch asynchrone Aufrufe ermöglicht.

XML-RPC verwendet HTTP als Transportprotokoll, als Datenformat wird XML eingesetzt. Die Apache-Implementierung schreibt Handler-Klassen vor, die die aufzurufenden Methoden enthalten. Instanzen dieser Klassen registriert man beim lokalen Webserver, der in der Implementierung enthalten ist, wenig Ressourcen benötigt und keiner Konfiguration bedarf. Nach der Registrierung können die Methoden der

[12] http://ws.apache.org/xmlrpc

Handler-Klassen von entfernten Rechnern sowohl synchron als auch asynchron aufgerufen werden.

Kryptographie

Die Realisierung des Kommunikationssubsystems setzt den Einsatz kryptographischer Mittel voraus (siehe hierzu Kapitel 4, zu kryptographischen Grundlagen siehe Anhang, Abschnitt A). Zur Verschlüsselung wurde ein Hybridverfahren eingesetzt, das auf einer Kombination von RSA und 3DES (siehe Anhang, Abschnitt A) basiert. Hierbei wird ein zufälliger 3DES-Schlüssel generiert, mit dem die Daten verschlüsselt werden. Dieser Schlüssel wird anschließend mit einem öffentlichen RSA-Schlüssel verschlüsselt und zu den Daten hinzugefügt. Zur Berechnung von Hashfunktionen wurde SHA-1 (siehe [Schn96]) verwendet, die Generierung von Signaturen basiert auf einer Kombination von SHA-1 und RSA. Die eingesetzten Algorithmen können jedoch ohne Aufwand durch andere ausgewechselt werden, etwa 3DES durch AES oder SHA-1 durch MD5.

Das aktuelle Java Software Development Kit (Version 1.4.2) beinhaltet nicht alle benötigten kryptographischen Funktionen, es bietet jedoch über die *Java Cryptography Extension*-Schnittstelle (*JCE*) die Erweiterung dieser Funktionen durch einen *JCE-Provider*. Bei diesem handelt es sich um eine signierte JAR-Datei, die kryptographische Methoden enthält. Durch die Registrierung des Providers werden diese Methoden verfügbar und können über die JCE-Schnittstelle aufgerufen werden.

Die *Bouncy Castle Crypto API*[13] bietet einen geeigneten, frei erhältlichen JCE-Provider, der alle benötigten Funktionen bereitstellt. Der Quellcode ist außerdem einsehbar, so dass er auf Sicherheitslücken überprüft werden kann.

Logging

Während der Entwicklung werden Ausgaben benötigt, die hauptsächlich zur Fehlersuche verwendet werden können. Nach Abschluss der Entwicklung, während des Betriebs, können diese Ausgaben jedoch auch zur Protokollierung verwendet werden. Die entstehenden Protokolle können zur Überprüfung der Korrektheit des Systems, zum Debugging, oder aber zur Aufzeichnung von Details zum Systemzustand, deren Ausgabe über eine Benutzeroberfläche nicht sinnvoll wäre.

Die Protokollierung kann einerseits selbst implementiert werden, etwa mittels einfacher Methoden zur Bildschirm- oder Dateiausgabe. Der Einsatz von Logging-Frameworks ist aber meist sinnvoller. Das Framework *log4j*[14], das wie Lucene auch im Rahmen des Apache Jakarta-Projekts entwickelt wurde, bietet eine hohe Flexibilität.

Die Konfiguration der log4j-Protokollierung kann statisch im Programm festgelegt werden, kann aber auch mittels einer Konfigurationsdatei erfolgen. Zu den einstellbaren Optionen zählen die Prioritätsstufe, das Ziel und das Format der Ausgabe. log4j unterscheidet zwischen fünf verschiedenen Prioritätsstufen – von Debug-Meldungen über Warnungen bis hin zu fatalen Fehlermeldungen.Die Prioritätsstufe legt fest, welche Ausgaben vorgenommen werden; wählt man etwa die WARN-Stufe,

[13] http://www.bouncycastle.org
[14] http://jakarta.apache.org/log4j

so werden alle Ausgaben ignoriert, die eine niedrigere Prioritätsstufe haben. Zu den Ausgabezielen zählen die Konsole, Protokolldateien einstellbarer Größe und grafische Dialoge, aber auch entfernte Logging-Server. Auch das Ausgabeformat kann beeinflusst werden; zu den Komponenten der Ausgabe zählen unter anderem die Meldung, die aufrufende Java-Klasse, die Zeile innerhalb dieser Klasse, Datum und Zeit, und die Prioritätsstufe.

Ferner kann mit Hilfe von log4j eine hierarchische Logging-Struktur ähnlich der Java-Package-Struktur aufgebaut werden, wodurch die Konfiguration für die Protokollierung innerhalb bestimmter Teile des Systems unabhängig von dem Gesamtsystem vorgenommen werden kann – hierdurch kann man zum Beispiel erreichen, dass Fehlermeldungen zur Datenhaltung in einer Datei protokolliert werden, während Warnungen, die die Systemsicherheit betreffen, sowohl in einer Datei aufgezeichnet, als auch an der Konsole angezeigt werden, und das gesamte System fatale Fehlermeldungen an einen Administrator sendet, etwa per Mail oder SMS.

6.2 Datenhaltung

Die Datenmodelle des UniTEC-Systems zeichnen sich vor allem dadurch aus, dass ein Teil der zu speichernden Daten – vor allem die Empfehlungen – eine beliebige Struktur haben. Diese Daten müssen anschließend durch intuitive Anfragen im Datenbestand auf effiziente Weise ausfindig gemacht werden können.

Innerhalb dieser Arbeit wurde ein Framework entwickelt, das diese Anforderungen erfüllt. Es basiert auf *Datencontainern*, also Konstrukten, die beliebig strukturierte Daten sowohl im persistenten als auch im flüchtigen Zustand vorhalten können.

Im Folgenden werden einige wichtige Aspekte des Frameworks beschrieben. Details zur Implementierung und dazu gehörende Klassendiagramme sind im Anhang, Abschnitt 0 enthalten.

XML-Persistenz

Basierend auf den Erkenntnissen, die in Abschnitt 6.1 beschrieben wurden, wurde die persistente Speicherung der Daten auf der Grundlage der XML-Serialisierung von JavaBeans mit Hilfe des JOX-Frameworks realisiert. Dieses ermöglicht die transparente Generierung von XML-Dokumenten, in denen die zu speichernden Daten enthalten sind. Durch den Einsatz von JOX ergeben sich einige Einschränkungen bezüglich der Typen der JavaBean-Eigenschaften, die im Anhang, Seite 85 beschrieben werden.

Die dadurch entstehenden XML-Dokumente werden von der XML-Datenbank Apache XIndice gespeichert. Diese ermöglicht, wie bereits im letzten Abschnitt beschrieben, das Auffinden gespeicherter Daten mit Hilfe der Abfragesprache XPath.

Volltextsuche

Die Daten können vor der Speicherung in Datencontainern mit Hilfe von Lucene indiziert werden. Auf die vielfältigen Möglichkeiten von Lucene wurde im letzten Abschnitt auf Seite 58 eingegangen.

Je nachdem, ob die Datencontainer die Daten persistent speichern, wird der Volltext-Index, der bei der Indizierung mit Hilfe von Lucene entsteht, entweder persistent gespeichert oder im Hauptspeicher vorgehalten.

War die Volltext-Indizierung bei der Speicherung eines Eintrags aktiviert, so kann dieser mit Hilfe einer Volltextanfrage aufgefunden werden. Die Syntax von Volltextanfragen und die Suchmöglichkeiten, die sich durch die Formulierung dieser Anfragen ergeben, werden im Anhang, Abschnitt B beschrieben.

Cache-Container

Cache-Container sind spezielle Datencontainer durch deren Einsatz erreicht werden kann, dass für die Speicherung von Daten nur eine begrenzte Menge an Speicherplatz verwendet wird, indem man die maximale Größe des Containers durch Angabe einer bestimmten Anzahl an Einträgen definiert. Ferner ist es möglich, für einzelne Einträge eine *maximale Lebensdauer* (*TTL*, engl.: *time to live*) einzustellen, nach deren Ablauf der Eintrag gelöscht werden kann.

Wird die maximale Größe erreicht, so müssen zur Speicherung neuer Einträge ältere, bereits gespeicherte ersetzt werden. Die Kriterien, die hierbei berücksichtigt werden, werden als *Ersetzungsstrategie* bezeichnet.

Implementiert wurde die LRU-Ersetzungsstrategie (engl.: least recently used), gemäß der diejenigen Einträge ersetzt werden, auf die in letzter Zeit nicht zugegriffen wurde, und zwar die Einträge, deren Zeitpunkt, zu dem auf sie zugegriffen wurde, am weitesten in der Vergangenheit zurückliegt. Durch die Verwendung dieser Cache-Container kann erreicht werden, dass trotz begrenztem zur Verfügung stehenden Speicherplatzes relevante Daten vorgehalten werden können, indem Einträge ersetzt werden, die nicht benötigt werden.

Storage Manager

Zur Erstellung neuer und zur Verwaltung existierender Datencontainer können Storage Manager verwendet werden. Bei der Erstellung neuer Container kann angegeben werden, ob die Daten im persistenten oder flüchtigen Speicher vorgehalten werden; ferner kann bei der Generierung von Cache-Containern deren Größe, die Standard-Lebensdauer der Einträge und die Ersatzstrategie des Caches angegeben werden.

Manager, die Cache-Container verwalten, prüfen in regelmäßigen Abständen das Alter der Einträge und bereinigen die Container, indem sie die abgelaufenen Einträge löschen.

Durch die Möglichkeit zur Erstellung und Verwaltung von Containern zur Laufzeit wird ein hohes Maß an Flexibilität erreicht.

6.3 Kommunikationsschicht

Das Kommunikationssubsystem, deren Anforderungen bereits in Kapitel 4 beschrieben wurde, wurde auf der Grundlage der im Anschluss an die Anforderungen vorgeschlagenen Lösung entwickelt.

Implementierung

Im Folgenden wird auf die Implementierung des Kommunikationssubsystems eingegangen.

Verteilte Hashtabelle

Zur Realisierung eines verteilten Verzeichnisdiensts zur Speicherung der Routing-Informationen und der öffentlichen Schlüssel aktiver Pseudonyme wurde eine verteilte Hashtabelle (DHT) auf der Grundlage von Chord implementiert. Die Prinzipien von Chord wurden in Kapitel 3 beschrieben, einschließlich der Möglichkeit zur Erreichung einer redundanten Datenspeicherung.

Bei der Chord-Implementierung waren die Anleitung und die Pseudocode-Ausschnitte, die in [SMK+01] enthalten sind, besonders hilfreich. Der Aufwand der Implementierung kann durch den Einsatz eines RPC-Mechanismus deutlich reduziert werden (siehe hierzu [SMK+01]). Wie bereits in Abschnitt 6.1 beschrieben, wurde während einer Evaluation bestehender RPC-Systeme die Entscheidung für Apache XML-RPC getroffen, das anschließend bei der Implementierung der Kommunikation verwendet wurde.

Chord wurde im Hinblick auf die Datensicherheit erweitert, so dass die Daten vor der Speicherung auf Authentizität und Aktualität überprüft werden. Die Daten – nämlich Routing-Informationen und öffentlicher Schlüssel eines aktiven Pseudonyms – werden vom Agenten des Pseudonyms mittels eines zufällig ausgewählten Mixes, der als Update-Proxy fungiert (siehe hierzu Abschnitt 4.4), in einer signierten Nachricht (`TrustedDataMessage`, siehe Anhang, Seite 92) an den zuständigen Knoten der DHT übergeben und auf dessen Nachfolger repliziert.

Anonyme Kommunikation

Zur anonymen Kommunikation wurde das Extended Destination Routing (siehe Abschnitt 4.5) implementiert. Es wurde eine Hop-to-Hop-Verschlüsselung mit zufälligem Padding realisiert, um einen Pattern Matching-Angriff zu erschweren. Die Kaskadenerweiterung mit Hilfe von Onion Routing wurde erst im Anschluss der Implementierung entwickelt, so dass sie nicht implementiert wurde.

Einige der Angriffsmöglichkeiten wurden während oder nach der Implementierung entdeckt, wie etwa der Last-Mix-Angriff, durch den ein böswilliger Sender, der zugleich letzter Mix der Kaskade ist, erkennen kann, dass der nächste Hop der Empfängeragent ist, falls dieser den Mix-Dienst nicht aktiviert hat (siehe Abschnitt 4.5).

Der Agent wurde so implementiert, dass bei dessen Start die lokale Adresse und eine Bootstrap-Adresse, d.h. die Adresse eines Agenten, der bereits aktiv ist, angegeben werden. Ferner kann hierbei entschieden werden, ob der Mix-Dienst gestartet wird oder nicht.

Pseudonyme werden bei der Aktivierung beim Agenten registriert, hierbei teilt man diesem die Daten des Pseudonyms mittels einer Instanz der Klasse `PseudonymInfo` (siehe Anhang, Seite 92) mit. Zu diesen zählen unter anderem das RSA-Schlüsselpaar des Pseudonyms, einige Parameter zur Einstellung des Anonymitätsgrads und Parameter zur Zusammensetzung der Nachbarschaft des Pseudonyms.

Der Anonymitätsgrad eines Pseudonyms kann durch zwei Parameter beeinflusst werden: die Länge der Mix-Kaskade, die für die Erstellung der Routing-Header benötigt wird, und die erwünschte Anzahl an verschiedenen Mixen, die bei der Erstellung dieser Kaskade verwendet werden soll. Ferner kann durch einen Parameter angegeben werden, ob auch eine geringere Anzahl an Mixen als die angegebene für die Kaskade verwendet werden darf, falls die MI-Liste (Mix-Informationsliste) die erwünschte Anzahl von Einträgen noch nicht erreicht hat.

Der Agent enthält einige aktive Komponenten, die als Java-Threads implementiert sind und in einstellbaren, regelmäßigen Abständen verschiedene Aktionen durchführen. Zu diesen zählen der Mix Discoverer, der Mix Information Updater und der Routing Information Updater.

Der *Mix Discoverer* (siehe `MixDiscoverer`-Klasse, Seite 97) prüft, ob die Mixe in der MI-Liste aktiv sind und anschließend, ob die Liste die gewünschte Länge hat. Ist dieses nicht der Fall, so wird eine Mix Discovery durchgeführt (siehe Abschnitt 4.6). Der *Mix Information Updater* (siehe `MixInfoUpdater`-Klasse, Seite 97) gibt hingegen Informationen über den eigenen Mix-Dienst an die Nachfolger des Agenten im Chord-Ring weiter, falls der Mix-Dienst aktiviert wurde.

Der *Routing Information Updater* (siehe `RoutingInfoUpdater`-Klasse, Seite 95) aktualisiert in regelmäßigen Abständen die Routing-Informationen, die im Verzeichnisdienst gespeichert sind, anhand aktueller Mix-Informationen, so dass sich Mix-Ausfälle nur beschränkt auf die Erreichbarkeit der Pseudonyme auswirken. Der Erreichbarkeitsgrad kann durch die Anzahl der Routing-Header, die in der Routing-Information enthalten sind, beeinflusst werden. Dieser Parameter wird bei der Registrierung des Pseudonyms dem Agenten mitgeteilt.

Die Parameter des Agenten, wie etwa das Aktualisierungsintervall für den Mix Discoverer, aber auch Standardwerte für die Parameter der Pseudonyme wie etwa solche zur Einstellung des Anonymitätsgrads, können in einer Properties-Datei angegeben werden (siehe hierzu Anhang, Abschnitt C).

Die Nachrichten, die bei der Kommunikation verwendet werden, entstehen durch XML-Serialisierung von JavaBeans mit Hilfe des JOX-Frameworks, das auch bei der Implementierung der Datenhaltung verwendet wurde (siehe Abschnitt 6.2). Hierdurch ergeben sich die gleichen Einschränkungen bezüglich der Datentypen für die Eigenschaften der verwendeten JavaBeans (siehe Anhang, Seite 85). Es können jedoch beliebige Daten (z.B. Bilder) serialisiert werden, wenn diese vorhin als String repräsentiert wurden. Hierzu bietet sich der *Base64*-Algorithmus an, der binäre Daten auf Strings abbilden kann, durch dessen Verwendung jedoch ein konstanter Overhead von etwa 33 % entsteht.

Zur Reaktion auf eintreffende Nachrichten werden *Message Listener* verwendet, die für jeweils einen bestimmten Nachrichtentyp beim Agenten pro Pseudonym registriert werden. Beim Eintreffen einer Nachricht dieses Typs wird der registrierte Listener aktiviert. Diese Listener müssen das Interface `MessageListener` (siehe Anhang, Seite 95) implementieren. Für die Aktivierung der Listener ist der `MessageDispatcher` (siehe Seite 91) des Pseudonyms zuständig, für das die Nachricht bestimmt war.

Performanzanalyse

Nach der Implementierung der Kommunikationsschicht wurden verschiedene Tests durchgeführt, zum einen um dessen Leistungsfähigkeit zu prüfen, jedoch auch mit dem sekundären Ziel, zu untersuchen, inwiefern ein höherer Anonymitätsgrad die Performanz der entwickelten Kommunikationslösung beeinflusst. Hierzu wurden diverse Messwerte gesammelt und, basierend auf diesen, Erkenntnisse gewonnen, die im Folgenden präsentiert werden.

Lokale Messung

Zunächst wurden zwei Agenten lokal auf einem Rechner installiert und der Mix-Dienst auf beiden aktiviert. Die Eigenschaften des Testrechners können Tabelle 6-1 entnommen werden. Zur Kommunikation wurde die *Loopback-Schnittstelle* verwendet. Hierbei handelt es sich um eine virtuelle Netzwerkkarte, über die ein Rechner in der Lage ist, intern mit sich selbst zu Testzwecken zu kommunizieren.

Prozessor:	AMD Athlon XP
Taktung:	1800 MHz
Hauptspeicher:	512 MB
Betriebssystem:	Windows XP
Java-Version:	1.4.1_02-b06

Tabelle 6-1: Umgebung für lokale Messung

Im Rahmen der Messung wurden je zwei Pseudonyme auf jedem Agenten registriert. Es wurden anschließend Messungen bezüglich der Latenz der Kommunikation zwischen je zwei Pseudonymen durchgeführt. Hierbei wurde die *Roundtrip-Zeit (round-trip time, RTT)*, also die Zeit zwischen dem Versenden einer Nachricht und dem Empfang der zu der jeweiligen Nachricht gehörenden Bestätigung gemessen.

Während der Messungen wurde zum einen die Länge der Kaskade variiert. Der Einfachheit halber wurden symmetrische Mix-Kaskaden verwendet, bei denen die Sender- und Empfänger-Teilkaskaden die gleiche Länge haben. Da in diesem Versuch nur zwei Agenten und hiermit nur zwei Mixe zur Verfügung standen, war die Bildung von Schleifen in Kaskaden, deren Länge mehr als zwei beträgt, unvermeidlich, so dass etwa bei einer Länge von 40 eine Nachricht jeden Mix im Durchschnitt zwanzigmal passieren wird. Zum anderen wurde aber auch die Größe der versendeten Nachricht zwischen 1024 Byte und 65536 Byte variiert. Jede Messung wurde zehnmal durchgeführt und der Durchschnitt dieser Messungen wurde nach der Berechnung festgehalten.

Nachrichten-	Länge der Mix-Kaskade								
Größe (Byte)	0	2	4	6	8	10	20	30	40
1024	207	401	653	846	1225	1496	3223	4859	6493
2048	206	445	654	906	1281	1634	3228	5268	6906
4096	207	485	773	1020	1256	1571	3357	5461	7759
8192	226	573	803	1114	1603	1889	3798	5860	8465
16384	248	759	1086	1543	1956	2257	4832	6925	9407
32768	320	1056	1582	2039	2689	3059	6064	9418	12673
65536	401	1634	2171	3087	3798	4192	8698	13034	18443

Tabelle 6-2: Werte für RTT (in ms) bei lokaler Messung ohne Wartezeit

Der gesamte beschriebene Messvorgang wurde zweimal durchgeführt: beim ersten Mal wurde die Warteschlange der Mixe ohne Wartezeit konfiguriert, so dass die Nachrichten sofort weitergeleitet wurden, und anschließend wurde ein Warteschlangenintervall von 100 Millisekunden eingestellt. Die Ergebnisse sind jeweils in Tabelle 6-2 und Tabelle 6-3 enthalten.

Nachrichten-	Länge der Mix-Kaskade								
Größe (Byte)	0	2	4	6	8	10	20	30	40
1024	204	643	1043	1473	1862	2328	4745	6837	9360
2048	193	679	996	1426	1903	2407	4611	7084	10526
4096	198	665	1111	1426	1940	2326	4904	7567	11265
8192	209	714	1195	1701	2262	2756	5418	8359	12457
16384	240	868	1448	2062	2664	3334	6126	9612	13675
32768	303	1142	2028	2676	3582	4182	8129	12175	16957
65536	460	1535	2837	3764	4978	5715	11075	16089	21870

Tabelle 6-3: Werte für RTT (in ms) bei lokaler Messung mit Wartezeit

Die Diagramme in Abbildung 6-1 und Abbildung 6-2 enthalten die ermittelten Messreihen als Kurvendarstellungen. Diese sind einander sehr ähnlich: bei kürzeren Mix-Kaskaden weist der RTT-Wert geringe Schwankungen auf, mit zunehmenden Kaskadenlängen steigt er jedoch linear. Die Steigung der Kurven wird zum einen von der Größe der Nachricht bestimmt, wobei die Unterschiede bei Nachrichten geringen Umfangs vernachlässigbar sind und sich nur bei der Verwendung längerer Mix-Kaskaden bemerkbar machen.

Zusätzlich ist die Steigung aller Kurven abhängig von der durchschnittlichen Zeit, die eine Nachricht in der Warteschlange der Mixe verbringt. Vergleicht man die Werte der Tabelle 6-2 mit denen, die in Tabelle 6-3 enthalten sind, so wird ein deutlicher Zuwachs von bis zu 50 % bemerkbar.

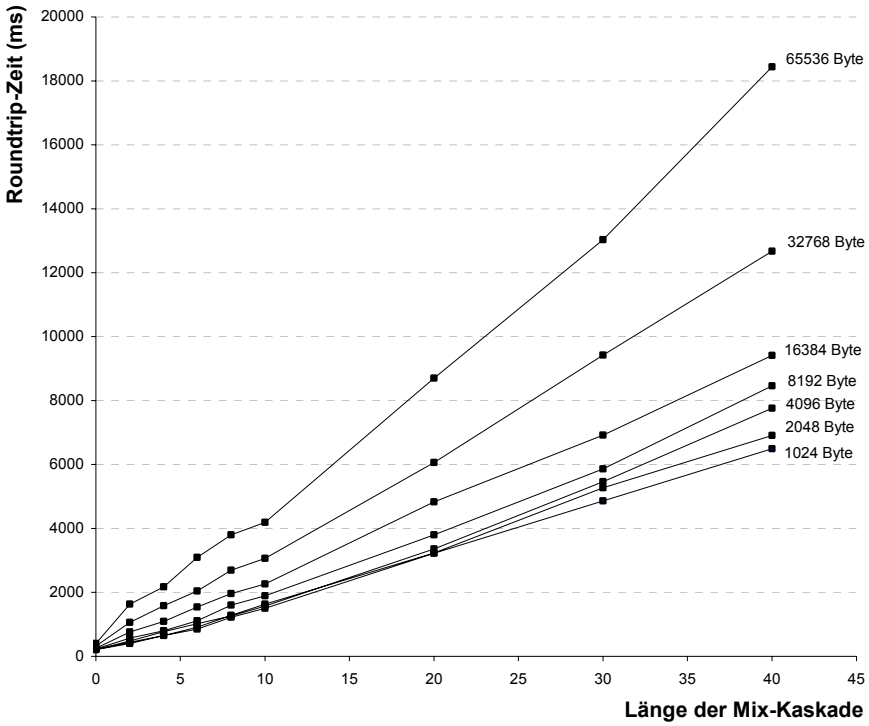


Abbildung 6-1: Werte für RTT bei lokaler Messung ohne Wartezeit

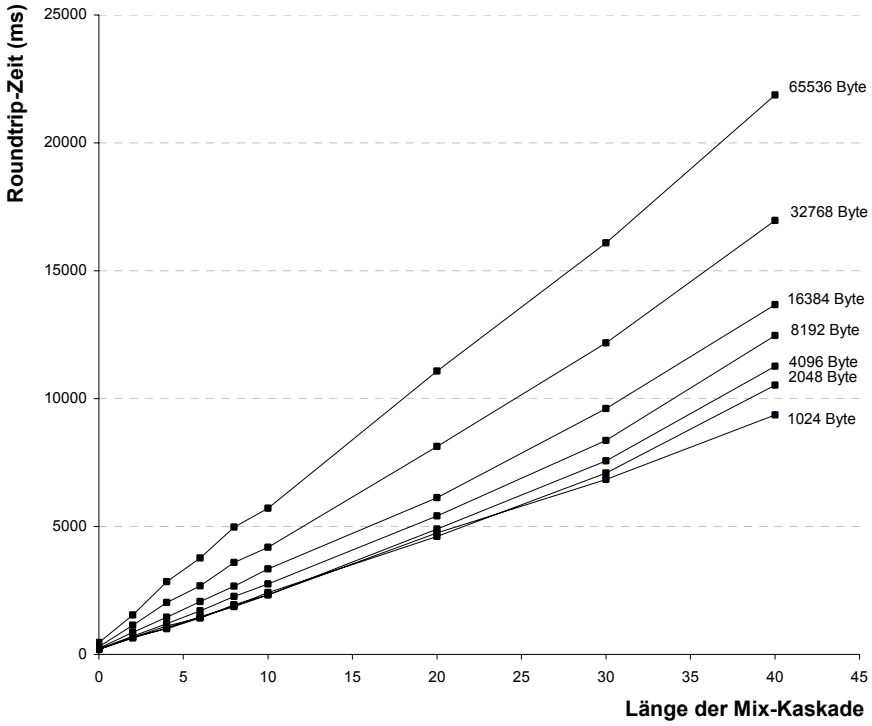


Abbildung 6-2: Werte für RTT bei lokaler Messung mit Wartezeit

Cluster-Messung

Weiterhin wurde das Kommunikationssubsystem in einer Cluster-Umgebung getestet (siehe Tabelle 6-4). Diese entspricht – im Gegensatz zu einer lokalen Messumgebung – weitgehend den realen Einsatzbedingungen, da sie mehrere unabhängige, vernetzte Knoten enthält.

Anzahl verwendeter Knoten	31
Prozessoren:	Intel Pentium IV
Taktung:	2400 MHz
Hauptspeicher pro Knoten:	512 MB
Netzwerk:	1 Gbit/s
Betriebssystem:	Linux (2.4.21)
Java-Version:	1.4.1-b21

Tabelle 6-4: Umgebung für Cluster-Messung

Der Test basiert, ähnlich dem lokalen Messverfahren, auf der Messung der Roundtrip-Zeit von Nachrichten, die zwischen Pseudonymen versendet werden. Auf jedem Knoten des Clusters wird ein Agent gestartet, an dem sich drei Pseudonyme registrieren. Der Mix-Dienst wird auf der Hälfte der Agenten (16) gestartet. Um ein Peer-to-Peer-Netz mit aktiven Agenten zu simulieren, wird Datenverkehr generiert, indem jeder Agent alle 5 Sekunden eine Nachricht verschickt, deren Größe 2048 Bytes beträgt. Hierbei wird eine Mix-Kaskade der Länge 10 verwendet.

Nachrichten-	Länge der Mix-Kaskade								
Größe (Byte)	0	2	4	6	8	10	20	30	40
1024	205	1027	1388	2055	3172	3458	6932	11050	15268
2048	205	735	1422	2435	3454	4008	7436	11279	15687
4096	281	1089	1454	2272	3228	3752	7378	11574	15995
8192	230	986	1799	2766	3796	4290	8367	12762	17169
16384	357	1123	2067	2994	4083	5195	9582	15461	19578
32768	319	1572	2445	4300	5444	6317	12166	18058	24177
65536	471	2382	4053	5641	7409	8957	17289	25085	33043

Tabelle 6-5: Werte für RTT (in ms) bei Cluster-Messung ohne Wartezeit

Wie bei den lokalen Messungen wurden die Mix-Warteschlangen erst so konfiguriert, dass sie die Nachrichten sofort weitergeben. Dabei wurde erneut sowohl die Kaskadenlänge als auch die Nachrichtengröße variiert. Anschließend wurde der Versuch wiederholt, dieses Mal jedoch mit einem Warteschlangenintervall von 100 ms. Um vorzubeugen, dass Schwankungen des gemessenen RTT-Werts, die im Vergleich zur lokalen Messumgebung ausgeprägter sind, sich auf die Messergebnisse stark auswirken, wird jeder RTT-Wert als Durchschnittswert von zwanzig einzelnen Messungen berechnet. Die Ergebnisse sind in jeweils in Tabelle 6-5 und Tabelle 6-6 enthalten.

Nachrichten-Größe (Byte)	Länge der Mix-Kaskade								
	0	2	4	6	8	10	20	30	40
1024	217	1384	1751	2586	3053	4924	9347	14182	19176
2048	223	1431	1815	2753	3678	4766	9507	15019	19565
4096	222	1564	2041	2678	3740	4983	9613	14952	19784
8192	288	1612	2335	3323	4436	5292	10124	15614	20785
16384	281	1598	2590	3644	5115	6094	11857	17297	23645
32768	497	1982	3337	4344	5613	7194	14179	20807	27946
65536	421	2829	4267	6427	7874	9849	18883	27553	35719

Tabelle 6-6: Werte für RTT (in ms) bei Cluster-Messung mit Wartezeit

Eine Untersuchung der zu den enthaltenen Messreihen gehörenden Kurvendarstellungen in Abbildung 6-1 und Abbildung 6-2 bestätigt erneut die während der lokalen Messung festgestellte Linearität.

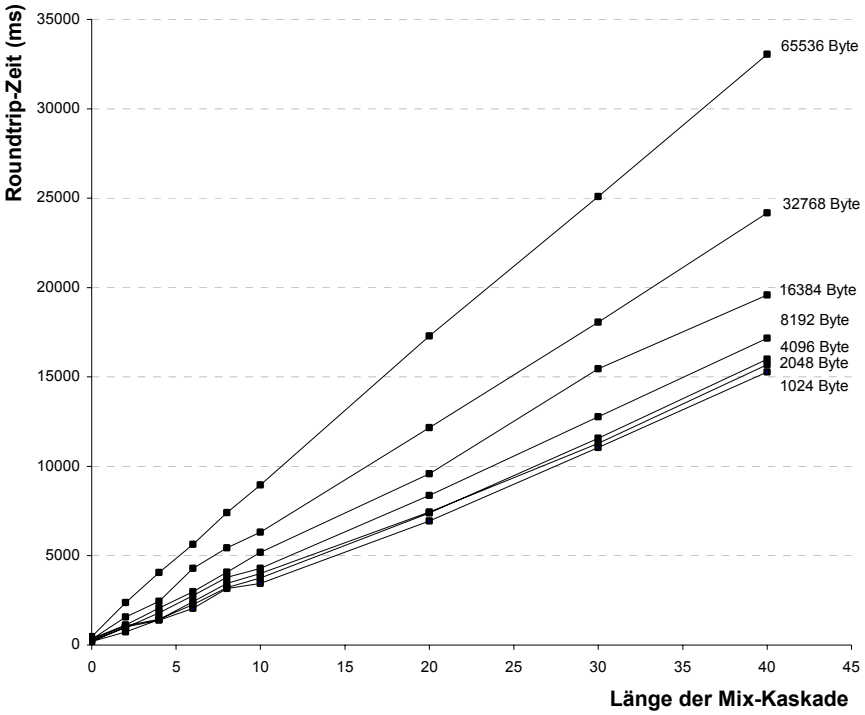


Abbildung 6-3: Werte für RTT bei Cluster-Messung ohne Wartezeit

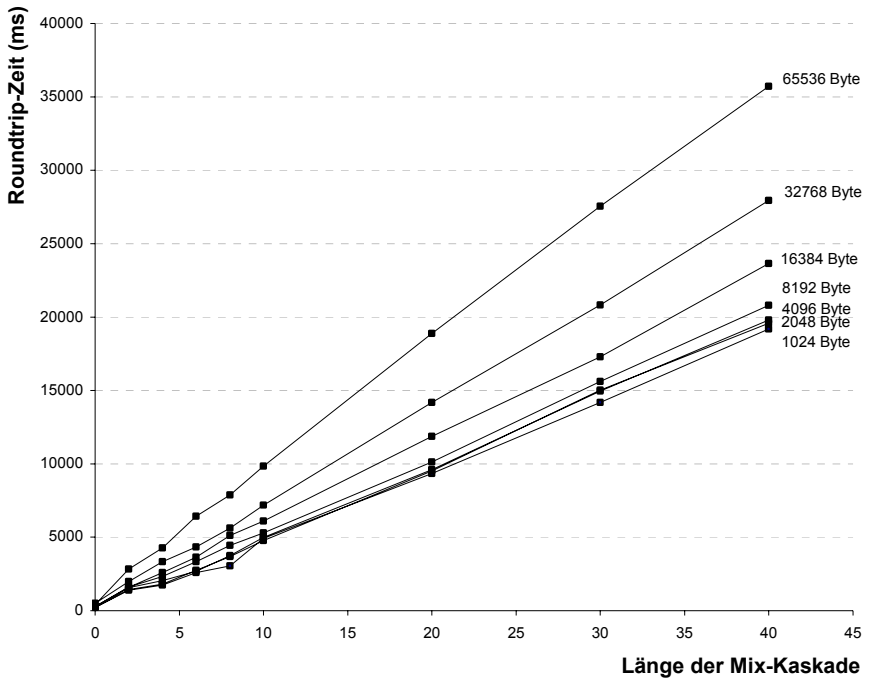


Abbildung 6-4: Werte für RTT bei Cluster-Messung mit Wartezeit

Bewertung

Die Versuche haben die nahe liegende Vermutung bestätigt, dass ein höherer Anonymitätsgrad basierend auf einer längeren Mix-Kaskade nur durch Performanzeinbußen zu erkaufen ist. Ein deutlicher linearer Zusammenhang zwischen der Kaskadenlänge und der Latenz der Kommunikation ist erkennbar. Ferner wird die Latenz auch von der Verweildauer der Nachrichten in den Warteschlangen der Mixe und von der Größe der Nachrichten direkt beeinflusst.

Die festgestellte Linearität ist dadurch zu erklären, dass alle Mixe der Mix-Kaskade gleich konfiguriert sind und die identische Hardware verwenden. Die Zeit, die pro Mix in Kauf genommen werden muss, setzt sich zum einen aus der Gesamtdauer der Ver- und Entschlüsselungsvorgänge und der Verweildauer der bearbeiteten Nachricht in der Warteschlange des Mixes. Bei der Implementierung wurde eine Hop-to-Hop-Verschlüsselung verwendet, so dass an jedem Mix die eintreffende Nachricht zunächst entschlüsselt wird, um anschließend, nach der Entfernung der ersten Schicht des Routing-Headers, wieder verschlüsselt zu werden. Die durchschnittliche Zeit, die für diese Operationen benötigt wird, ist für identisch ausgestattete Mixe näherungsweise gleich, da sich die geringe Differenz in der Nachrichtengröße, die durch die Entfernung der Schichten des Routing-Headers entsteht, praktisch nicht auf die Dauer der erwähnten Operationen auswirkt. Auch die durchschnittliche Verweildauer in der Warteschlange des Mixes weist bei identisch eingestellten Werten für das Warteschlangenintervall geringe Schwankungen auf.

In der Praxis variiert jedoch sowohl die Hardwareausstattung der Mixe als auch deren Konfiguration. Dadurch unterliegt in einer realen Umgebung die Latenz pro Mix mit hoher Wahrscheinlichkeit stärkeren Schwankungen als im Test.

6.4 Empfehlungsdienst

Die in Kapitel 5 entworfene Lösung wurde basierend auf dem entwickelten Framework zur Datenhaltung und der anonymen Kommunikationsschicht bis auf die Implementierung von Interessensgebieten komplett umgesetzt.

Durch den Einsatz der flexiblen Datenhaltung können Empfehlungen beliebig strukturiert werden. Sie können aus einer beliebigen Anzahl an Feldern bestehen. Als Feldtypen können folgende Java-Datentypen verwendet werden

- für boolesche Felder: `bool`
- für Zahlen: `int` und `float`
- für Zeichenketten: `String`
- für binäre Daten: `String` generiert durch Base64-Codierung

Bei der Erstellung eines Feldes wird dessen Inhalt zusammen mit dem Namen des Feldes der Methode übergeben, die für den jeweiligen Feldtypen zuständig ist. Der angegebene Name des Feldes muss innerhalb des Feldtyps eindeutig sein.

Zur Speicherung lokal erstellter Empfehlungen wird ein persistenter Datencontainer, eine Instanz der Klasse `XMLDBStorage` (siehe Anhang, Seite 86) verwendet. Vor der Speicherung werden die in der Empfehlung enthaltenen Daten indiziert, so dass anschließend eine Volltextsuche durchgeführt werden kann. Die Syntax von Volltextanfragen wird im Anhang, Abschnitt B beschrieben. Zur Zwischenspeicherung von Empfehlungen wird ein persistenter Cache-Container – eine Instanz der Klasse `XMLDBCache` (siehe Seite 87) – verwendet, dessen Größe vom Benutzer eingestellt werden kann. Auch dieser Datencontainer kann mit Hilfe einer Volltextsuche durchsucht werden. Diese wird bei der Beantwortung eintreffender Anfragen verwendet, wie bereits in Kapitel 5 beschrieben.

Anfragen, die sich über ein Interessensgebiet erstrecken, wurden im Rahmen dieser Arbeit nicht implementiert; da deren Auswirkungen auf die Anonymität der Pseudonyme noch erforscht werden muss.

Kapitel 7

Fazit

Dieses Kapitel fasst die Ergebnisse zusammen, die in der vorliegenden Arbeit erzielt wurden. Anschließend werden einige Vorschläge zur Erweiterung der entwickelten Lösungen gemacht und einige Forschungsaspekte beschrieben, die einer näheren Untersuchung bedürfen.

7.1 Zusammenfassung

Im Rahmen der vorliegenden Diplomarbeit wurde ein universell einsetzbarer Empfehlungsdienst entwickelt, der auf einer Peer-to-Peer-Architektur basiert und dabei ohne zentrale Komponenten wie etwa zentralisierte Verzeichnisdienste auskommt. Hierbei wurde ein Ansatz verfolgt, der die Kommunikation zwischen Benutzern unter Verwendung von Pseudonymen ermöglicht, die Aufdeckung der Identität der Pseudonyme verhindert und dadurch die persönlichen Daten der Benutzer schützt.

Zu diesem Zweck wurden verschiedene bestehende Ansätze zur anonymen Kommunikation in Peer-to-Peer-Umgebungen analysiert und deren Vor- und Nachteile abgewogen. Anschließend wurde das Onion Routing-Verfahren dahingehend erweitert, dass es die Anforderungen des zu entwickelnden Empfehlungsdiensts an die Anonymität erfüllt. Hierbei wurde ein neues, skalierbares Verfahren entwickelt, dessen Ziel zum einen die Wahrung der Anonymität sowohl des Senders als auch des Empfängers ist, zum anderen aber auch die Kommunikationsbeziehung zwischen diesen verheimlicht und die versendeten Daten vor der Einsicht durch Dritte schützt.: jeder Teilnehmer im Peer-to-Peer-Netzwerk veröffentlicht die Daten, über die er erreicht werden kann, in einem verteilten Verzeichnisdienst. Anhand dieser Daten kann er schließlich Nachrichten empfangen, unter Wahrung der Anonymität aller Teilnehmer.

Anschließend wurden verschiedene Angriffsszenarien erkannt, die beim Einsatz des entwickelten Verfahrens die Anonymität der Benutzer gefährden können. Diese Szenarien wurden untersucht und es wurden Lösungen vorgeschlagen, durch die die entsprechenden Angriffe erschwert oder – soweit möglich – verhindert werden können.

Anhand des entwickelten Verfahrens wurde das Kommunikationssubsystem implementiert und auf verschiedenen Rechnern erfolgreich getestet; hierbei wurde bestätigt, dass die Latenz linear mit dem gewählten Anonymitätsgrad steigt. Darauf aufbauend wurde anschließend der Empfehlungsdienst entwickelt. Für die Speicherung von Daten im System wurde eine flexible Lösung zur Datenhaltung entwickelt, die

Daten beliebiger Struktur speichern und diese mit Hilfe intuitiver Volltextanfragen auffinden kann.

7.2 Ausblick

Innerhalb der vorliegenden Arbeit wurden mehrere Angriffsszenarien untersucht, die sich auf die Kommunikation zwischen zwei Pseudonymen beziehen, mit dem primären Ziel, die Anonymität der Kommunikationsteilnehmer zu kompromittieren. Weiterhin wäre es sinnvoll, Angriffsmöglichkeiten in Erwägung zu ziehen, die auf die Beeinträchtigung der Erreichbarkeit der Teilnehmer abzielen. So sollte zum Beispiel untersucht werden, ob durch gezielte Attacken die verteilte Hashtabelle zum Erliegen gebracht werden kann.

Die Anzahl der Nachfolger im Chord-Ring beeinflusst die Robustheit und den Redundanzgrad des verteilten Verzeichnisdienstes. Andererseits sind sowohl die Replikation der zu speichernden Daten als auch die Verwaltung und Aktualisierung der Nachfolgerliste aufwendig. Die Wahl eines geeigneten Werts sollte näher untersucht werden.

Die Versendung von bedeutungslosem Verkehr (Dummy-Verkehr) wurde im Rahmen der Arbeit nicht implementiert. Es ist sinnvoll zu untersuchen, inwieweit die Skalierbarkeit des Verfahrens unter Verwendung von Dummy-Verkehr leidet.

Der Empfehlungsdienst sollte in Zukunft durch eine geeignete, idealerweise grafische Benutzeroberfläche ergänzt werden, um Empfehlungen ohne Aufwand erstellen zu können. Ferner wäre die Erweiterung des Dienstes um eine Möglichkeit zur Pflege verschiedener Versionen sinnvoll.

Für die Auswahl der Teilnehmer, an die eine Anfrage gesendet wird, ist die Verfügbarkeit der Expertisedaten von besonderer Wichtigkeit. Daher ist die Speicherung dieser Daten in einem Verzeichnisdienst in Betracht zu ziehen, um somit ein Äquivalent zu den bekannten „Gelben Seiten“ zu realisieren. Eine nahe liegende Möglichkeit hierfür wäre die Speicherung der Expertisedaten in der verteilten Hashtabelle. Beispielsweise könnte jedes Pseudonym bei der Aktivierung einen Teil seiner Expertisedaten, etwa die Anzahl der gespeicherten Empfehlungen pro Kategorie, im Verzeichnisdienst veröffentlichen. Da jedoch zu erwarten ist, dass die Anzahl der Kategorien wesentlich geringer als die der aktiven Pseudonyme, würde bei einer Speicherung im Chord-Ring, bei der der zuständige Knoten etwa durch Hashing des Kategorienamens ermittelt wird, zur Folge haben, dass die Expertisedaten nicht gleichmäßig verteilt, sondern auf wenigen Agenten angehäuft werden. Hierzu müssen geeignete Speicherstrukturen entwickelt werden, die eine gleichmäßige Verteilung und eine hohe Effizienz bieten.

Durch Anpassungen am Framework, das zur Datenhaltung entwickelt wurde, kann eine höhere Flexibilität erreicht werden. Beispielsweise sollten auch andere Ersetzungsstrategien für Cache-Container in Erwägung gezogen und eventuell auch implementiert werden. Außerdem ist die Entwicklung von Mitteln zweckmäßig, durch die die Flexibilität des Lucene-Frameworks, das zur Volltextsuche eingesetzt wird, besser genutzt werden kann. Sinnvoll ist etwa die Reduzierung des Aufwands zur

Erweiterung der Liste bedeutungsloser Wörter oder eine Möglichkeit zur Vereinfachung des Wechsels zwischen verschiedenen Sprachen.

Anhang

A. Kryptographische Grundlagen

Bei der Datenübertragung zwischen Sender und Empfänger werden kryptographische Mittel primär zur Wahrung der Vertraulichkeit der übertragenen Daten eingesetzt. Diese Mittel können zusätzlich auch anderen Zwecken dienen, wie etwa der Sicherung der Integrität der Daten oder der Authentifizierung. In den meisten Systemen, die Anonymität implementieren, sind sie daher unentbehrlich.

Dieser Abschnitt geht kurz auf die wichtigsten kryptographischen Verfahren ein.

Symmetrische Verfahren

Bei symmetrischen Verfahren wird sowohl bei der Verschlüsselung als auch bei der Entschlüsselung der Daten derselbe Schlüssel verwendet. Damit der Empfänger der verschlüsselten Daten diese entschlüsseln kann, muss der Schlüssel mit dem Sender ausgetauscht werden. Der Schlüsselaustausch muss über einen sicheren Kanal erfolgen. Dieses stellt den wichtigsten Nachteil symmetrischer Verfahren dar.

Etablierte Verfahren wie etwa *DES* (Data Encryption Standard) basieren auf einfachen Operationen wie Permutationen, Substitutionen und XOR-Verknüpfungen (siehe [Schn96]). Diese Operationen können sehr effizient implementiert werden, daher ist die Performanz sehr hoch.

Hardware-Implementierungen, die Entwicklung immer schnellerer Rechner und deren paralleler Einsatz zum gemeinsamen Lösen von Aufgaben haben dazu geführt, dass das beliebte DES-Verfahren in seiner ursprünglichen Form nicht mehr sicher ist. Detaillierte Anleitungen zum Kompromittieren des Verfahrens sind verfügbar, etwa in [Elec98], daher wurde das Verfahren erweitert.

Der *3DES*-Algorithmus verwendet drei Stufen, die auf dem klassischen DES-Verfahren basieren, dieses entspricht der Verdreifachung der DES-Schlüssellänge. Ein anderes Verfahren, das nach diversen Einschätzungen 3DES in Kürze ablösen wird, ist *AES* (*Advanced Encryption Standard*)[1].

Asymmetrische Verfahren

Der Nachteil symmetrischer Verfahren besteht hauptsächlich darin, dass für den Schlüsselaustausch ein sicherer Kanal verwendet werden muss. Asymmetrische Verfahren verwenden hingegen zwei Schlüssel: einen öffentlichen Schlüssel, der über beliebige Kanäle verfügbar gemacht werden kann, und einen privaten (oder geheimen) Schlüssel, der vor fremder Einsicht geschützt werden muss.

[1] Informationen unter http://csrc.nist.gov/CryptoToolkit/aes verfügbar

Wird ein Datenblock mit einem der Schlüssel verschlüsselt, so kann der andere der beiden zur Entschlüsselung verwendet werden. Werden also Daten mit einem öffentlichen Schlüssel chiffriert, so kann nur der Besitzer des dazu gehörenden privaten Schlüssels die Daten entschlüsseln. Asymmetrische Verfahren können aber auch zur Erstellung von digitalen Signaturen verwendet werden (Siehe Abschnitt „Hashfunktionen").

Das bekannteste asymmetrische Verschlüsselungsverfahren ist *RSA*, das nach dessen Entwicklern Rivest, Shamir und Adleman benannt wurde (siehe [RSA78]). Das Verfahren basiert auf zahlentheoretischen Überlegungen, die ergeben haben, dass große Zahlen nur mit sehr hohem Aufwand in Primzahlen zerlegbar sind (Faktorisierungsproblem, siehe [MOV96]). Obwohl es Ende der siebziger Jahre entwickelt wurde, gilt es auch heute noch als sehr sicher und wird sehr häufig eingesetzt, sowohl in Chiffrier- als auch in Signiersystemen.

Hashfunktionen

Hashfunktionen generieren aus einem Datenblock beliebiger, jedoch finiter Länge eine Bitfolge fester Länge. Gängige Längen für solche Bitfolgen sind 160 Bit bei der weit verbreiteten *SHA-1*-Funktion, oder 128 Bit bei MD5.

Hashfunktionen sind nur dann in der Kryptographie brauchbar, wenn sie nicht umkehrbar sind, also wenn aus dem Hashwert die Eingangsdaten nicht rekonstruiert werden können. Da die Größe des Eingangsdatenblocks meistens die Länge des daraus berechneten Hashwerts um ein Vielfaches übertrifft, sind Kollisionen unvermeidlich. Durch eine Gleichverteilung der berechneten Werte wird das Kollisionsrisiko jedoch reduziert.

Hashfunktionen können bei der Prüfung der Datenintegrität verwendet werden. Der zu sichernde Datenblock wird zusammen mit dem berechneten Hashwert versendet. Der Empfänger kann den Hashwert neu berechnen und diesen mit dem empfangenen vergleichen. Sind die zwei Werte identisch, so kann der Empfänger mit sehr hoher Wahrscheinlichkeit davon ausgehen, dass die Integrität der Daten gegeben ist.

Verschlüsselt der Sender den berechneten Hashwert vor dem Versenden mit seinem privaten Schlüssel, so kann dieser vom Empfänger mit dem öffentlichen Schlüssel des Senders entschlüsselt werden. Ist nun der dadurch erhaltene Wert identisch mit dem selbst berechneten Hashwert, so ist nicht nur die Integrität, sondern auch die Authentizität der Daten gesichert. Diese Methode wird häufig bei der Erstellung *digitaler Signaturen* verwendet (z.B. bei OpenPGP[1]).

[1] Format-Spezifikation verfügbar unter http://www.ietf.org/rfc/rfc2440.txt

B. Syntax von Volltextanfragen

Dieser Abschnitt beschreibt die Syntax, die bei der Formulierung von Volltextanfragen verwendet werden kann, und deren Bedeutung. Die Syntax ist durch die Verwendung von Lucene (Version 1.2) gegeben. Weitere Details können der Lucene-Website[1] entnommen werden.

Terme

Eine Anfrage besteht aus Termen. Terme sind einzelne Wörter oder Phrasen. Phrasen sind Wortmengen, die in Anführungszeichen angegeben werden. Beispiele hierfür sind `Digitalkamera` oder `"optischer Sucher"`.

Boolesche Operatoren

Terme können durch boolesche Operatoren verbunden Werden. Zu diesen zählen `AND`, `OR`, `NOT`, `-` und `+`. Die Anfrage

```
Bank AND Park -"Geld"
```

findet diejenigen Dokumente, die sowohl die Terme `Bank` und `Park`, jedoch nicht `Geld` enthalten

Felder

Möchte man nach dem Inhalt eines bestimmten Feldes im Dokument suchen, so kann man vor dem Term den Feldnamen angeben, mit `:` als Trennzeichen:

```
Titel:"Krieg und Frieden"
```

Wird kein Feldname angegeben, so sucht Lucene nach dem angegebenen Term im festgelegten Standardfeld. Um eine erfolgreiche Suche ohne Kenntnis der Struktur durchführen zu können, wurde Lucene so integriert, dass das Standardfeld den gesamten Inhalt des Dokuments enthält – hierzu werden bei der Indizierung alle Felder des Dokuments konkateniert. Dieses hat zur Folge, dass ohne Angabe eines Feldnamen der gesamte Inhalt durchsucht wird.

Gruppierung

Mittels Klammern kann man eine Gruppierung erreichen, sowohl im Zusammenhang mit booleschen Operatoren:

```
(Resin OR Tomcat) AND Server
```

als auch bei der Angabe von Feldnamen:

```
Titel:("Krieg und Frieden" OR Faust)
```

[1] http://jakarta.apache.org/lucene

Term-Modifikatoren

Einige Optionen können über Modifikatoren angegeben werden. Hierdurch werden die Suchmöglichkeiten deutlich erweitert.

Wildcard-Suche

Ein Wildcard ist ein Platzhalter; erlaubt sind `*` und `?`. `?` steht für ein einzelnes Zeichen, während `*` eine beliebige Anzahl Zeichen ersetzt. `Mau?` findet zum Beispiel Dokumente, die `Maus`, `Maut` oder `Maul` enthalten, während `Mau*` auch `Maulwurf` findet.

Unscharfe Suche

Die unscharfe Suche kann verwendet werden, wenn die genaue Schreibweise eines Terms nicht genau bekannt ist. `Maus~` findet zum Beispiel sowohl `Maul` als auch `Haus`.

Abstandssuche

Die Entfernung zwischen Termen im Dokument kann auch berücksichtigt werden. Zum Beispiel findet

```
"Suchmaschine Internet"~10
```

diejenigen Dokumente, in denen `Suchmaschine` und `Internet` im Abstand von höchstens zehn Wörtern voneinander vorkommen.

Intervallsuche

Man kann nach Dokumenten suchen, deren Felder Werte aus einem bestimmten Bereich beinhalten, etwa folgendermaßen:

```
Alter:[18 TO 40]
```

Escaping

Folgende Zeichen werden bei der Formulierung von Anfragen als Sonderzeichen behandelt:

```
+ - && || ! ( ) { } [ ] ^ " ~ * ? : \
```

Möchte man trotzdem nach einem oder mehreren dieser Zeichen suchen, so kann dieses durch Escaping mit Hilfe von `\` erreicht werden. Zum Beispiel findet die Anfrage

```
\(1\+2\)\*3
```

Dokumente, die `(1+2)*3` enthalten.

C. Parameter der Konfigurationsdatei `unitec.properties`

Im Folgenden werden die Konfigurationsparameter beschrieben, die Einfluss auf die in dieser Arbeit entwickelten Softwarekomponenten haben.

`com.hp.unitec.p2p.messageExpiryTime`

Gibt das maximale Alter einer Routing Update-Nachricht in Sekunden an (hierdurch werden Replay-Attacken verhindert).

`com.hp.unitec.p2p.chord.stabilizationInterval`

Intervall zwischen zwei Aufrufen der Chord-Stabilisierung.

`com.hp.unitec.p2p.mix.keySize`

Länge des öffentlichen Schlüssels, der für den lokalen Mix-Dienst beim Start erzeugt wird.

`com.hp.unitec.p2p.mix.knownMixes`

Anzahl Einträge in der Mix-Informationsliste, die während der Mix Discovery angestrebt werden.

`com.hp.unitec.p2p.mix.queueProcessorInterval`

Intervall zwischen zwei Leerungen der Mix-Warteschlange.

`com.hp.unitec.p2p.mix.updaterInterval`

Intervall zwischen zwei Aufrufen des Mix Information Updates

`com.hp.unitec.p2p.mix.discovererInterval`

Intervall zwischen zwei Mix Discovery-Vorgängen.

`com.hp.unitec.p2p.mix.maxPaddingLength`

Maximale Größe des zufälligen Paddings bei der Hop-to-Hop-Verschlüsselung

`com.hp.unitec.p2p.cascadeLength`

Standardwert für die Länge der Empfänger-Teilkaskade eines Pseudonyms.

`com.hp.unitec.p2p.preferredMixAmount`

Standardwert für die Anzahl der verschiedenen erwünschten Mixe in der Empfänger-Teilkaskade.eines Pseudonyms.

`com.hp.unitec.p2p.headerAmount`

Standardwert für die Anzahl der Routing-Header, die während eines Routing Updates in dem Verzeichnisdienst gespeichert werden.

com.hp.unitec.p2p.mandatory

Gibt an, ob die Anzahl der verschiedenen erwünschten Mixe der Teilkaskade unbedingt verwendet werden muss, oder ob bei Bedarf auch weniger Mixe bei der Erstellung von Routing-Headern verwendet werden können (etwa bei wenigen Einträgen in der Mix-Informationsliste).

com.hp.unitec.p2p.messageTimeout

Zeit in Sekunden, nach deren Ablauf ohne Eintreffen eines Bestätigungspakets davon ausgegangen wird, dass der Empfänger nicht erreicht wurde (zum Beispiel weil ein Mix ausgefallen ist). Danach wird ein anderer Routing-Header verwendet, falls verfügbar.

com.hp.unitec.p2p.routing.routingCacheSize

Maximale Anzahl von Routing-Informationen, die zwischengespeichert werden können.

com.hp.unitec.p2p.routing.routingCacheTTL

Maximale Lebensdauer der zwischengespeicherten Routing-Informationen.

com.hp.unitec.p2p.routing.keyCacheSize

Maximale Anzahl an öffentlichen Schlüsseln, die zwischengespeichert werden können.

com.hp.unitec.p2p.routing.keyCacheTTL

Maximale Lebensdauer der zwischengespeicherten öffentlichen Schlüssel.

com.hp.unitec.p2p.routing.updaterInterval

Intervall zwischen zwei Routing Update-Vorgängen.

com.hp.unitec.recmgr.hopCounter

Standardwert für den Hop-Counter eines Pseudonyms. Der Hop-Counter gibt an, wie oft eine Anfrage weitergereicht wird.

com.hp.unitec.recmgr.neighbourhoodSize

Standardwert für die Größe der Nachbarschaft eines Pseudonyms.

com.hp.unitec.recmgr.trustedRatio

Standardwert für den Anteil der Anzahl vertrauenswürdiger Pseudonyme an der gesamten Nachbarschaft.

com.hp.unitec.recmgr.hubRatio=25

Standardwert für den Anteil der Anzahl von Hubs an der gesamten Nachbarschaft.

`com.hp.unitec.recmgr.authorityRatio`

Standardwert für den Anteil der Anzahl von Authorities an der gesamten Nachbarschaft.

`com.hp.unitec.neighbourhood.reachabilityThreshold`

Schranke für den Erreichbarkeitswert eines Pseudonyms.

`com.hp.unitec.rpc.maxRetries`

Anzahl der wiederholten gescheiterten Versuche, eine entfernte Methode aufzurufen, bevor davon ausgegangen wird, dass der Rechner, der die Methode zur Verfügung stellt, nicht erreichbar ist.

`com.hp.unitec.rpc.retryPause`

Wartezeit zwischen einem gescheiterten Versuch, eine entfernte Methode aufzurufen, und einem erneuten Versuch.

`com.hp.unitec.storage.managers.VolatileCacheManager.purgeInterval`

Standardwert für das Intervall zwischen zwei Prüfungen des Alters der Einträge, die in einem flüchtigen Cache gespeichert sind. Überschreitet das Alter eines Eintrags seine maximale Lebensdauer, so wird der Eintrag entfernt.

`com.hp.unitec.storage.managers.PrevalentCacheManager.purgeInterval`

Standardwert für das Intervall zwischen zwei Prüfungen des Alters der Einträge, die in einem prävalenten Cache gespeichert sind. Überschreitet das Alter eines Eintrags seine maximale Lebensdauer, so wird der Eintrag entfernt.

`com.hp.unitec.storage.managers.XMLDBCacheManager.purgeInterval=5`

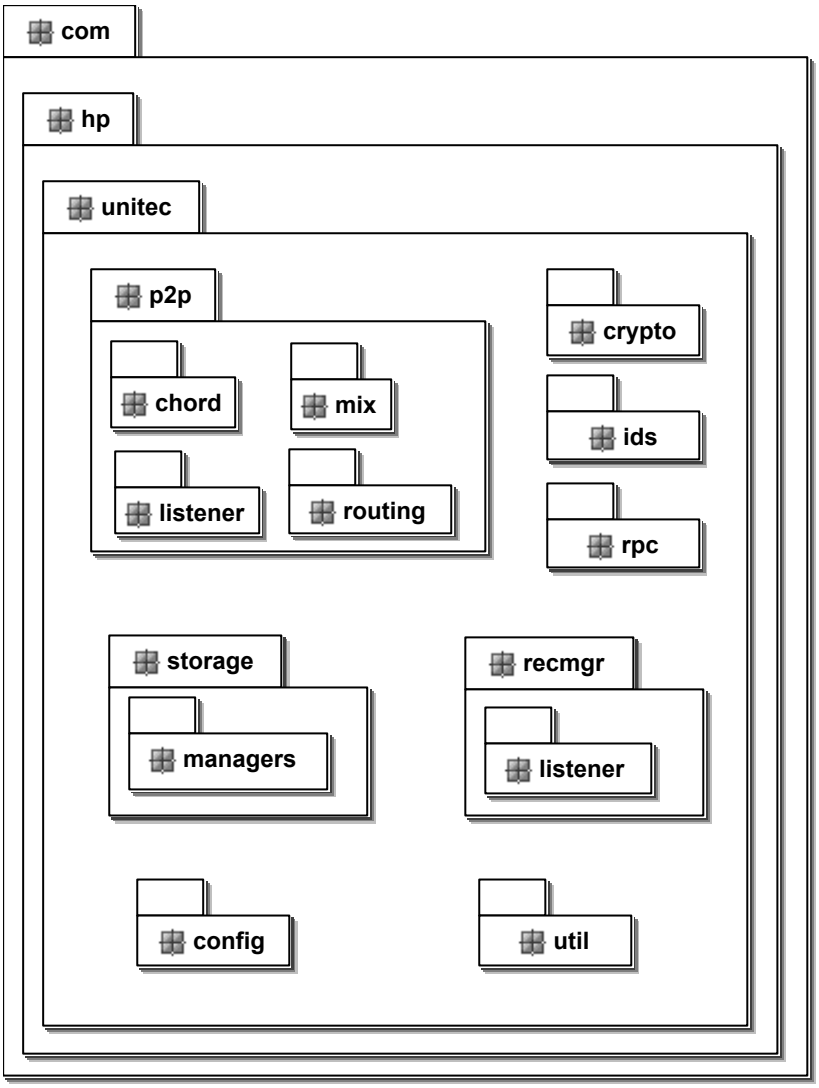
Standardwert für das Intervall zwischen zwei Prüfungen des Alters der Einträge, die in einem persistenten Cache gespeichert sind. Überschreitet das Alter eines Eintrags seine maximale Lebensdauer, so wird der Eintrag entfernt.

D. Implementierung

Dieser Abschnitt geht auf einige Implementierungsdetails ein, unter anderem auf die Package- und die Klassenstruktur der entstandenen Softwarekomponenten beschrieben.

Package-Hierarchie

Folgendes Bild zeigt eine Übersicht der Hierarchie der während der Implementierung entstandenen Java-Packages.



Klassen des Storage-Frameworks

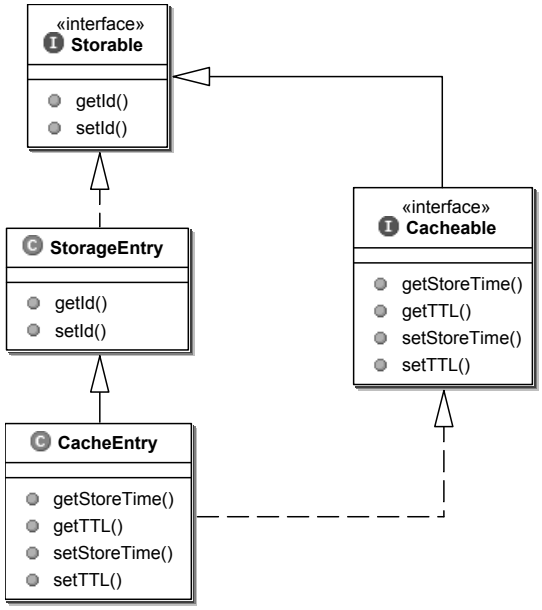
Im Folgenden werden diejenigen Klassen dargestellt, die Teil des Frameworks zur Datenhaltung sind. Der Übersichtlichkeit halber wurde hier auf eine Darstellung der Parameter und Rückgabewerte verzichtet.

*Storable-Interface (Package **com.hp.unitec.p2p.storage**)*

Das `Storable`-Interface muss von Klassen implementiert werden, deren Instanzen in `Storage`-Containern gespeichert werden sollen. Entsprechend implementieren Klassen, deren Instanzen in `Cache`-Containern gespeichert werden, das `Cacheable`-Interface.

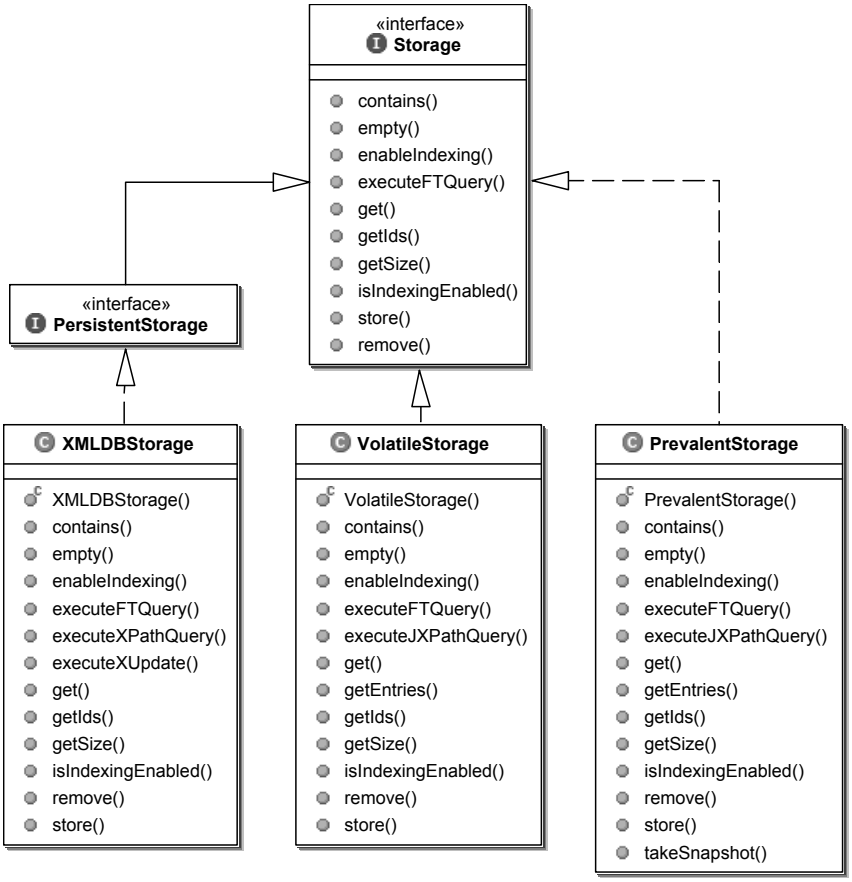
Damit deren Instanzen persistent gespeichert werden können, müssen beide Klassenarten als JavaBeans implementiert werden. Nur JavaBean-Eigenschaften werden persistiert, also die Attribute, auf die über Accessor- und Mutator-Methoden zugegriffen werden kann. Da die Instanzen dieser Klassen mit dem JOX-Framework zu XML serialisiert werden, können für die Eigenschaften folgende Typen verwendet werden:

- Java-Basistypen (`byte`, `char`, `short`, `int`, `long`, `float`, `double` und `boolean`)
- Wrapper-Klassen von Basistypen (`Byte`, `Char`, etc.)
- Instanzen der Klassen `java.util.Date` und `String`
- Andere JavaBeans, die diese Bedingungen erfüllen
- Arrays, die auf einem dieser Typen basieren



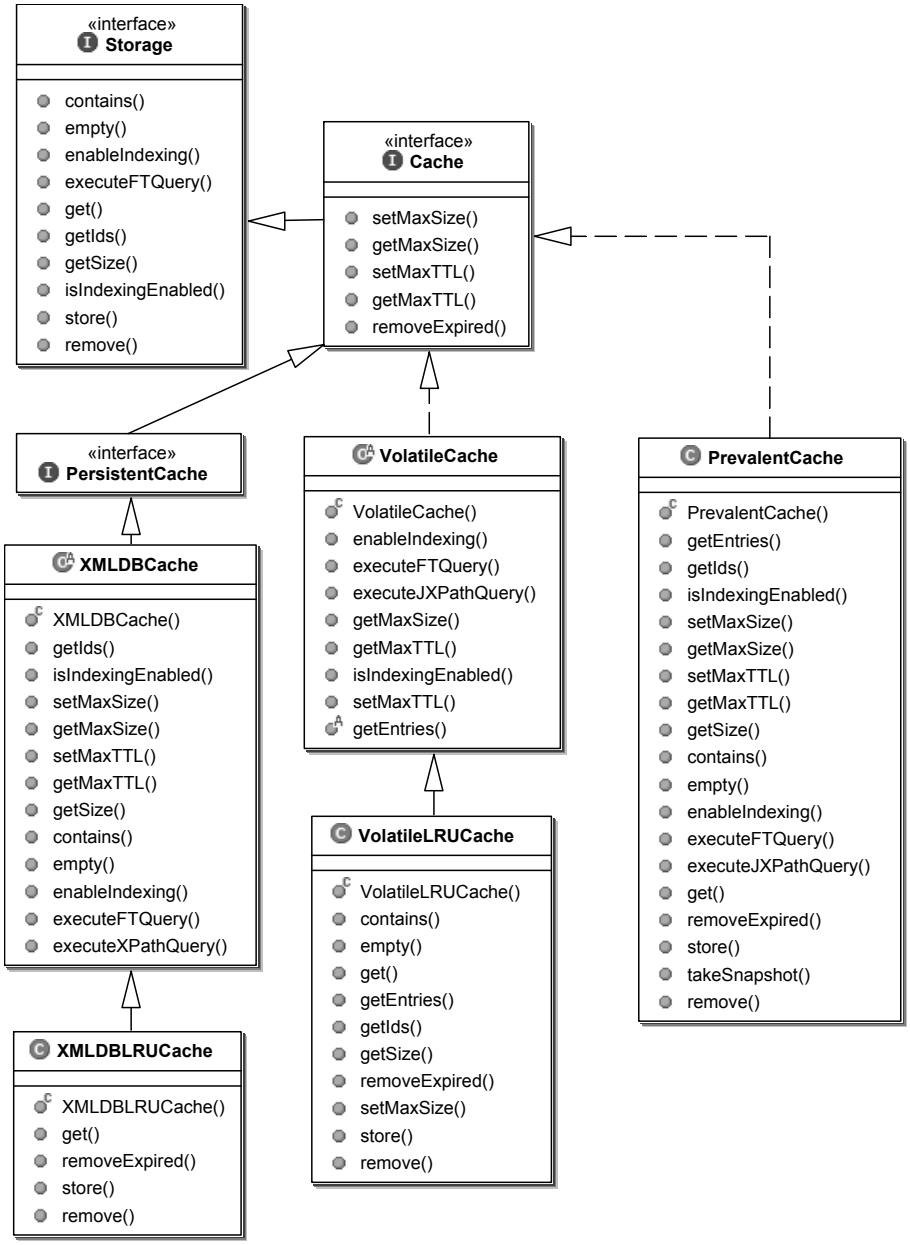
Storage-Klassen (Package ***com.hp.unitec.p2p.storage****)*

Die Storage-Container implementieren das Interface `Storage`. Implementiert wurden drei Klassen, entsprechend der Art und Weise, wie die Daten in den Containern vorgehalten werden: *persistent*, *volatil* und *prävalent*. Die *Persistenz* wird mit Hilfe der XML-Serialisierung von JavaBeans mit Hilfe des JOX-Frameworks realisiert. Bei der *volatilen* Speicherung werden die Daten lediglich im flüchtigen Speicher gehalten. Die *prävalente* Speicherung ist identisch mit der volatilen, zusätzlich werden jedoch alle Änderungen an den Daten persistent protokolliert, so dass der erreichte Zustand anhand des entstandenen Protokolls nach einem Absturz oder bei Neustart rekonstruiert werden kann; um die Dauer dieses Vorgangs zu verkürzen, können regelmäßig Schnappschüsse der Daten erstellt werden.



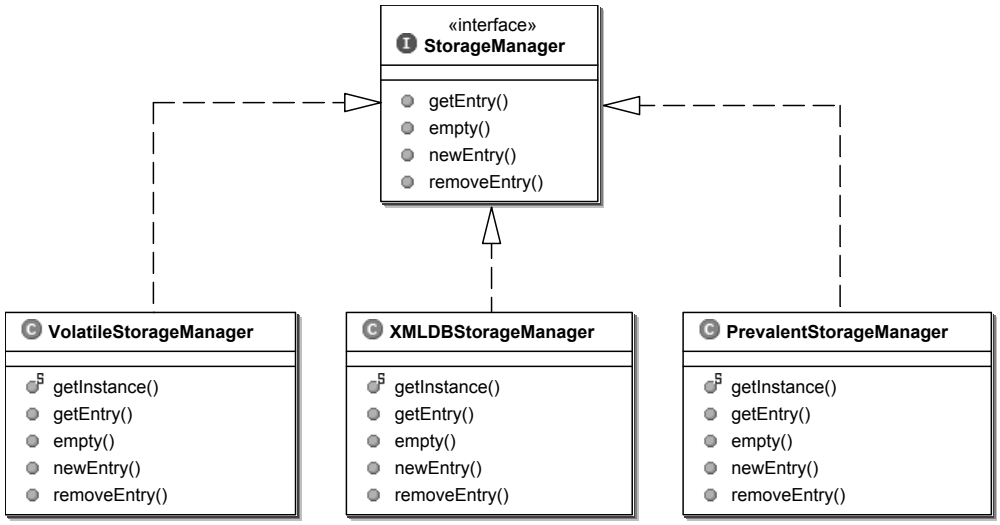
*Cache-Klassen (Package **com.hp.unitec.p2p.storage**)*

Die Cache-Container sind den Storage-Containern (siehe Seite 86) sehr ähnlich, sie erlauben jedoch die Beschränkung der Containergröße und die Festlegung einer maximalen Lebensdauer pro Eintrag (TTL, engl.: time to live). Als Ersetzungsstrategie wurde LRU (engl.: least recently used) implementiert – hierbei werden bei Erreichen der Maximalgröße des Containers diejenigen Einträge gelöscht, deren Zeitpunkt, zu dem auf sie zugegriffen wurde, am weitesten zurückliegt.



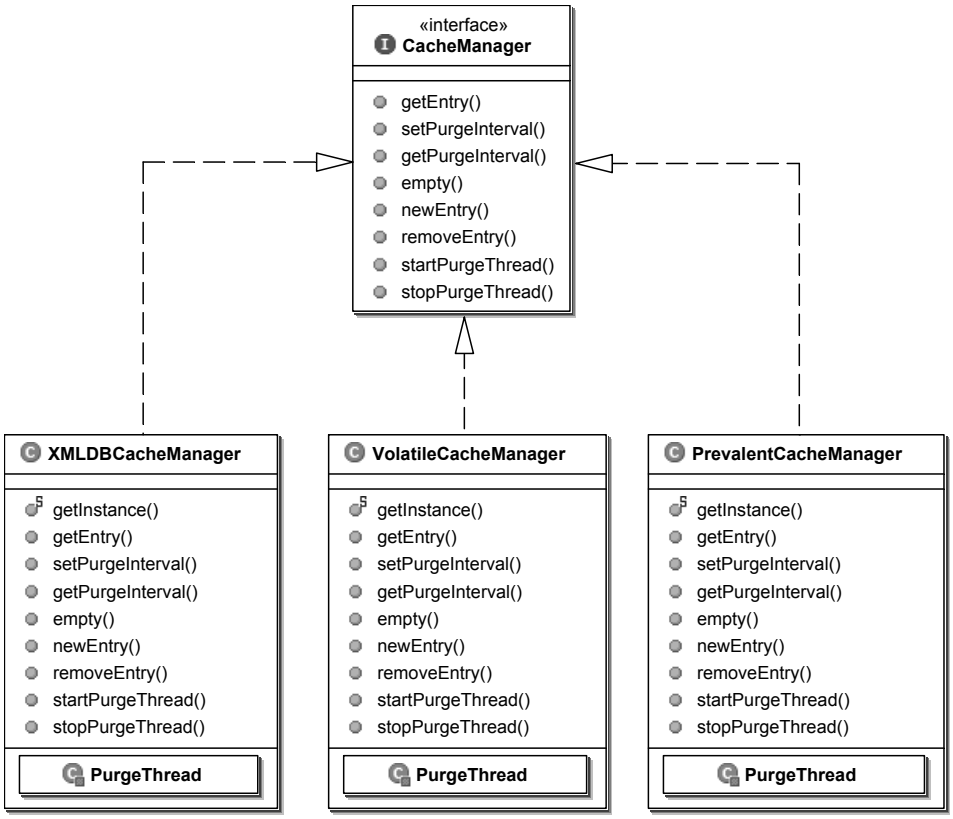
*Storage Manager-Klassen (Package **com.hp.unitec.p2p.storage.managers**)*

Über diese Klassen können die Storage-Container verwaltet werden.



Cache Manager-Klassen (Package com.hp.unitec.p2p.storage.managers)

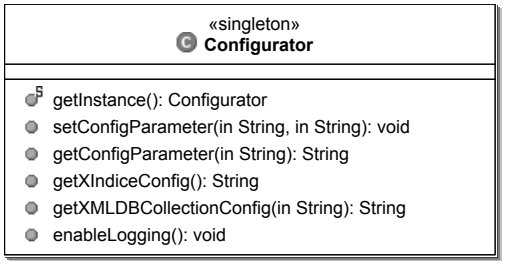
Dienen zur Verwaltung von Cache-Container. Zusätzlich sind sie für deren Säuberung zuständig. Ein Thread prüft periodisch die gespeicherten Einträge und löscht diejenigen, die abgelaufen sind.



Basisklassen

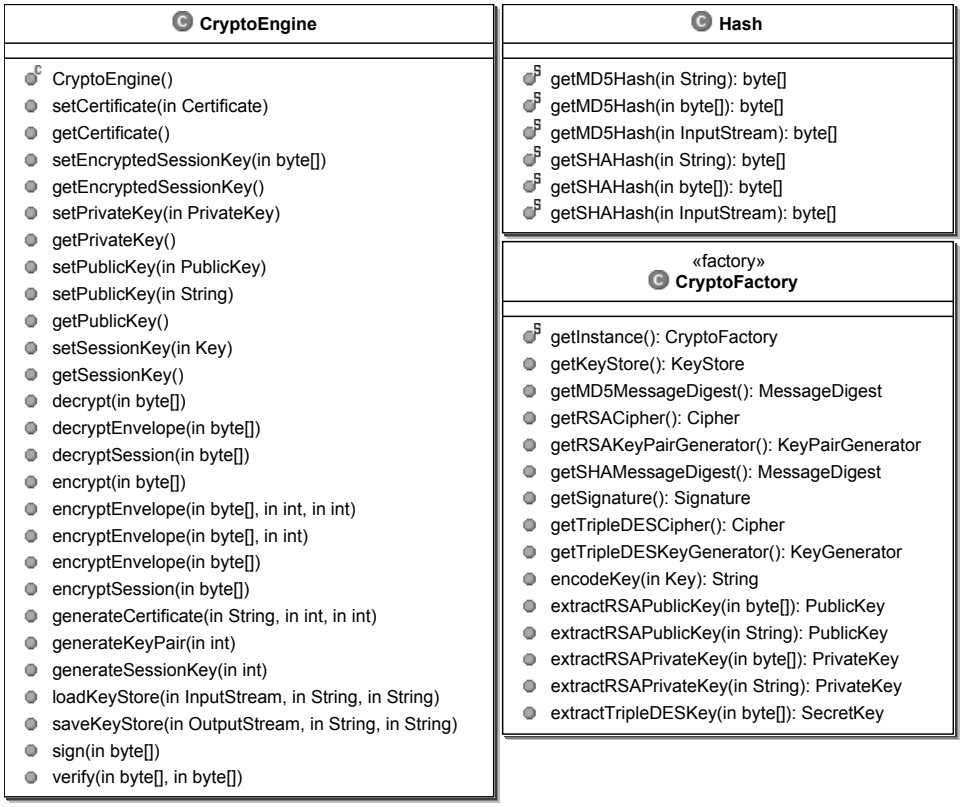
Package com.hp.unitec.config

Die `Configurator`-Klasse dient der Konfigurierung des UniTEC-Systems. Konfigurationen werden in der Datei `unitec.properties` persistent gespeichert.



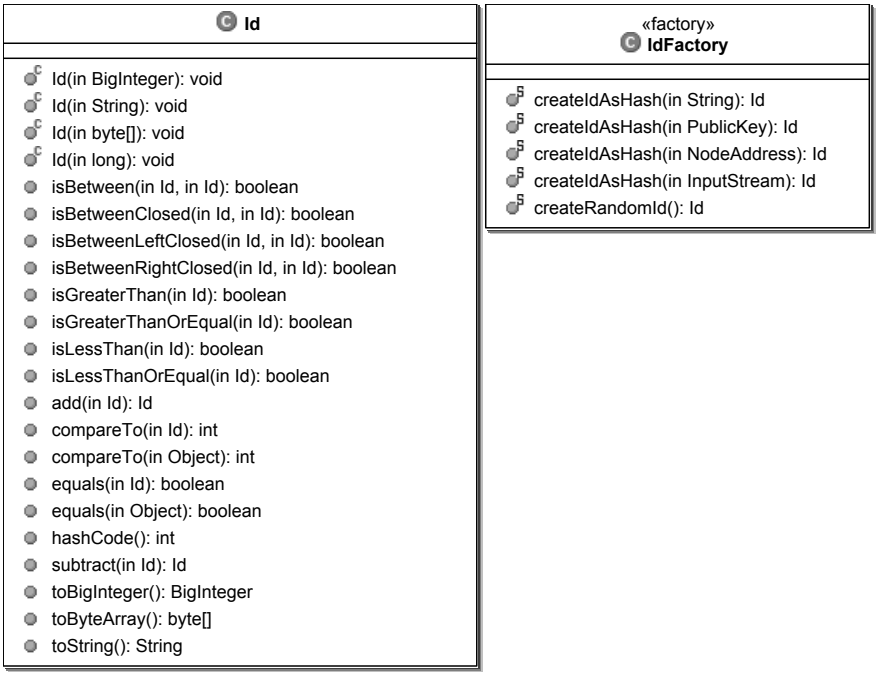
Package com.hp.unitec.crypto

Enthält alle kryptographischen Funktionen, die vom Kommunikationssubsystem und vom Empfehlungsmanager verwendet werden: Verschlüsselung mittels RSA und 3DES, Erstellung und Prüfung von Signaturen mittels SHA-1 und RSA, und Berechnung von SHA-1-Hashwerten.



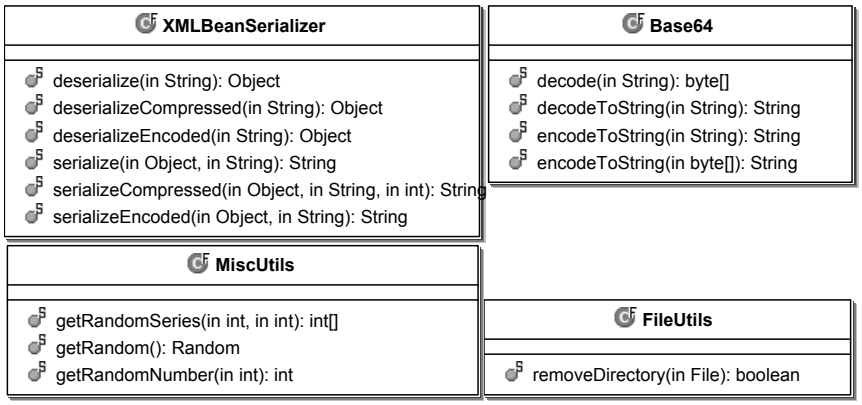
Package ***com.hp.unitec.ids***

Mit diesen Klassen können 160-Bit-IDs auf verschiedene Weise erstellt werden. Ferner bieten sie Rechen- und Vergleichsoperationen für diese IDs. Die IDs werden primär von der Chord-DHT benötigt, werden aber auch von den restlichen Subsystemen verwendet, etwa als Pseudonym-IDs.



Package ***com.hp.unitec.util***

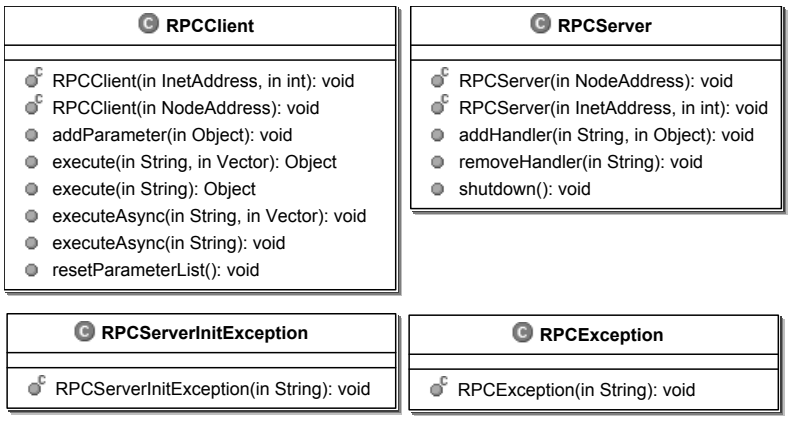
Dieses Package enthält einige Hilfsklassen, unter anderem auch die Klasse zur XML-Serialisierung.



Klassen des Kommunikationssubsystems

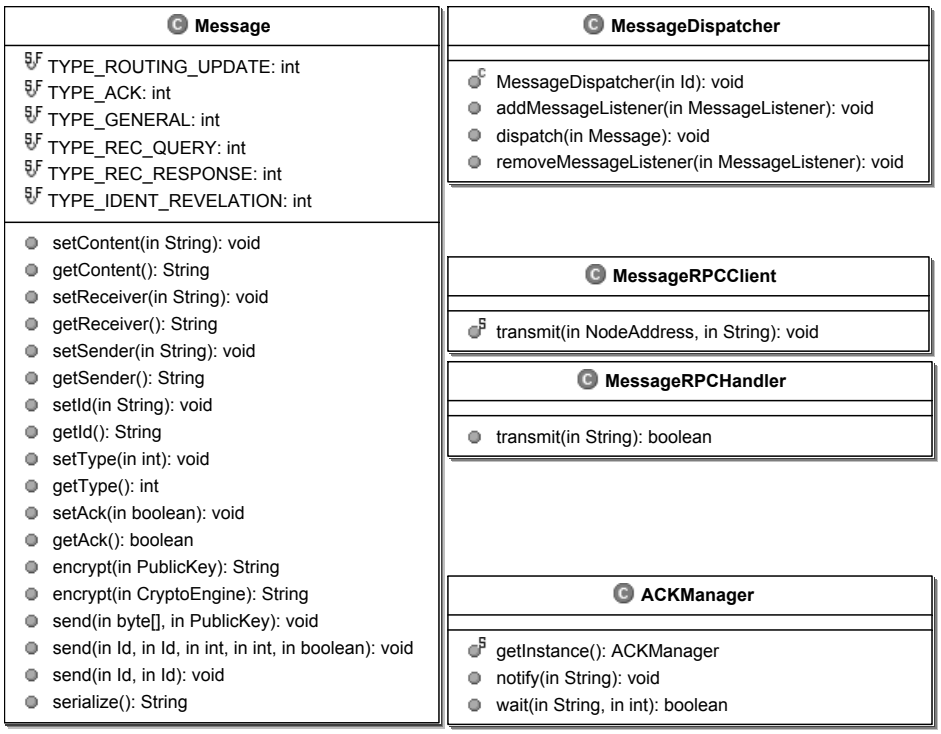
Package **com.hp.unitec.rpc**

Diese Klassen implementieren einen RPC-Mechanismus, der auf XML-RPC basiert.



Package **com.hp.unitec.p2p**

Folgende Klassen werden für die Erstellung, den Versand und den Empfang von Nachrichten im Peer-to-Peer-Netz verwendet. `Message` enthält die zu versendende Nachricht und stellt Methoden bereit, die zu deren Versendung benötigt werden. `MessageDispatcher` reagiert auf eintreffende Nachrichten, indem er den entsprechenden Listener aufruft. `AckManager` ist für die Verwaltung von Bestätigungsnachrichten verantwortlich.

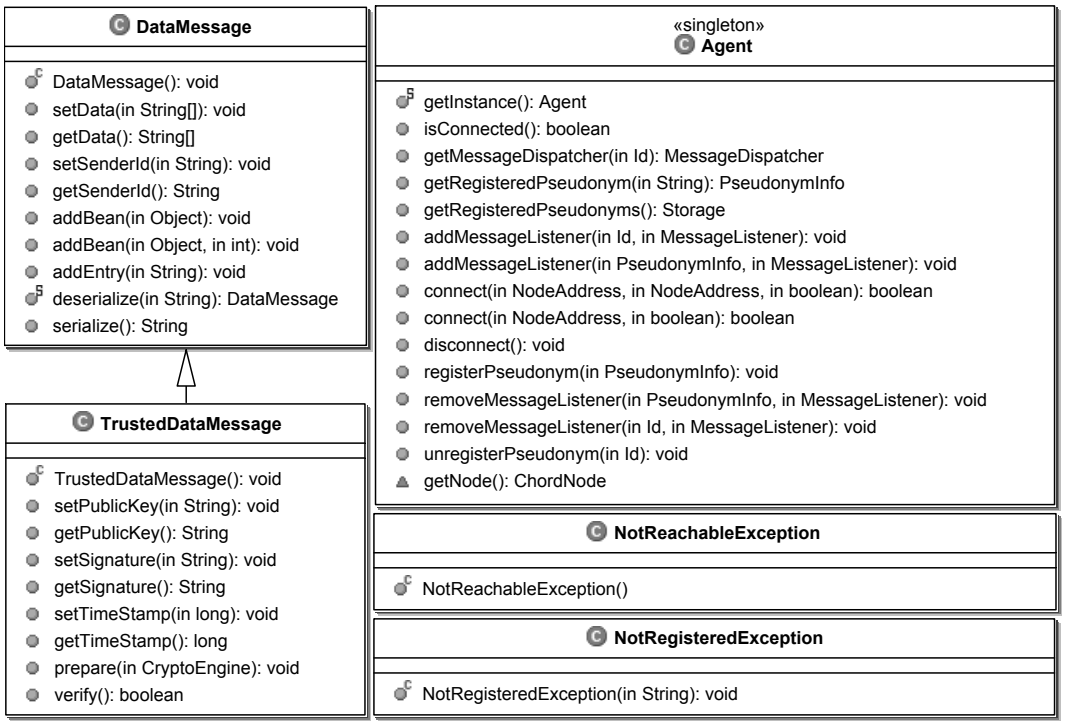


Package ***com.hp.unitec.p2p*** *(Fortsetzung)*

Die Klasse `TrustedDataMessage` ermöglicht sicheren Datentransport, indem sie Funktionalität zur Prüfung von Authentizität und Aktualität anbietet.

Die Klasse `Agent` realisiert einen Peer-to-Peer-Agenten und bietet folgende Funktionalität an:

- Aufbau und Trennung der Verbindung zum Peer-to-Peer-Netz
- Aktivierung des Mix-Diensts
- Registrieren und Deregistrieren von Pseudonymen
- Registrieren und Reregistrieren der Message Listener



Package ***com.hp.unitec.p2p*** *(Fortsetzung)*

Die Klasse `PseudonymInfo` beinhaltet Informationen zu Pseudonymen, die einerseits zur Kommunikation verwendet werden, aber auch vom Empfehlungsdienst genutzt werden:

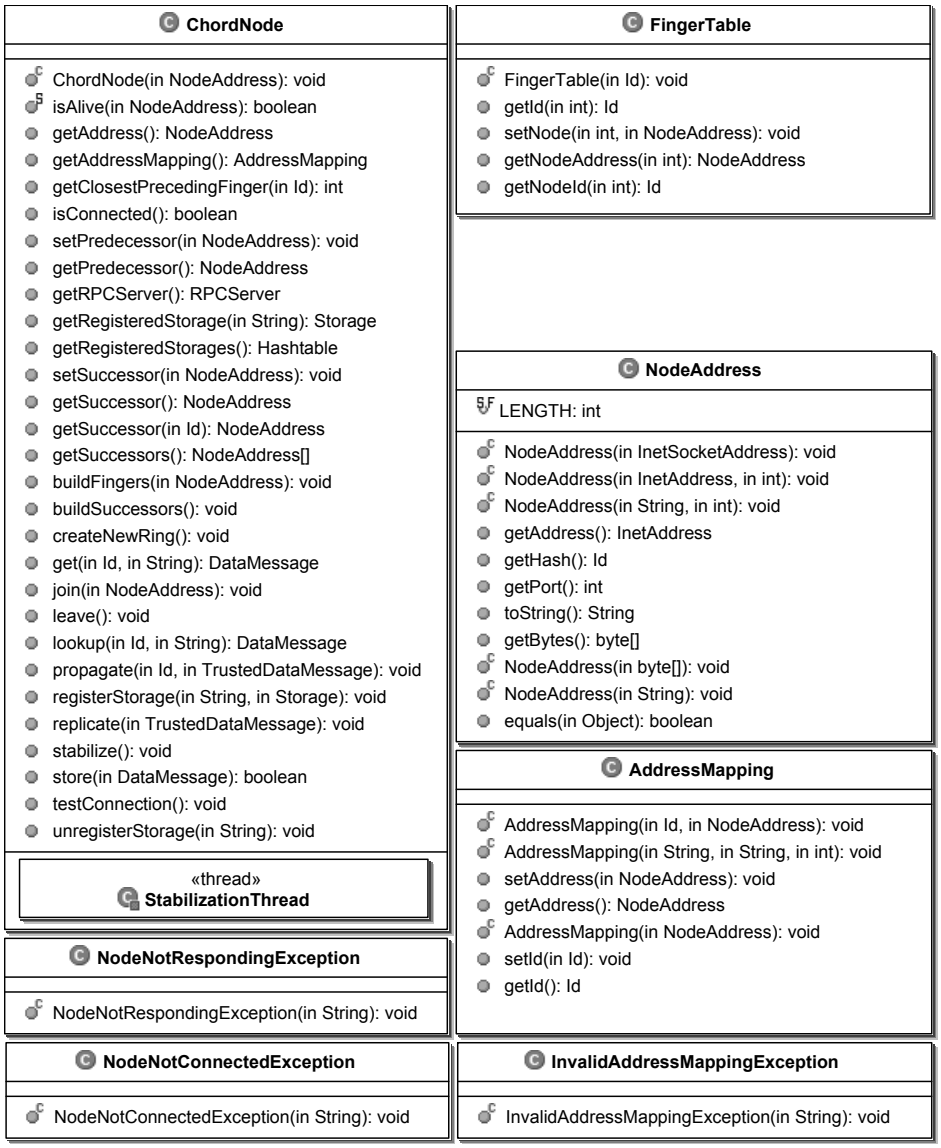
- RSA-Schlüsselpaar
- Anonymitätsparameter
- Parameter für die Zusammensetzung der Nachbarschaft

Package ***com.hp.unitec.p2p.chord***

Die Klassen dieses Packages implementieren eine redundante, verteilte Hashtabelle, die zur Speicherung der Routing-Informationen und öffentlichen Schlüssel der Pseudonyme verwendet wird. `ChordNode` enthält die Funktionalität, die zum Aufbau und zur Erhaltung des Chord-Rings notwendig ist, unter anderem den `StabilizationThread`, der für die periodische Stabilisierung des Rings zuständig ist.

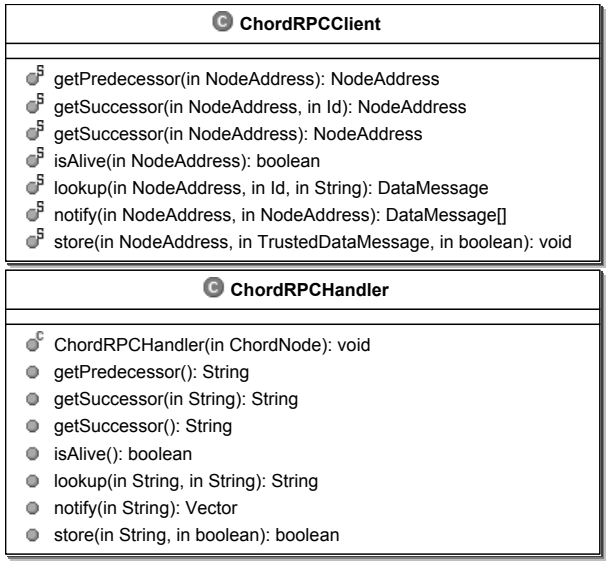
`FingerTable` enthält eine Tabelle, deren Einträge Sprünge innerhalb des Rings ermöglichen, die für die Effizienz des Verfahrens vorausgesetzt werden, da sie Lookup und Speicherung mit logarithmischem Aufwand ermöglichen.

`NodeAddress` und `AddressMapping` enthalten schließlich Adressdaten bzw. Zuordnungen zwischen Adressen und entsprechenden Knoten-IDs.



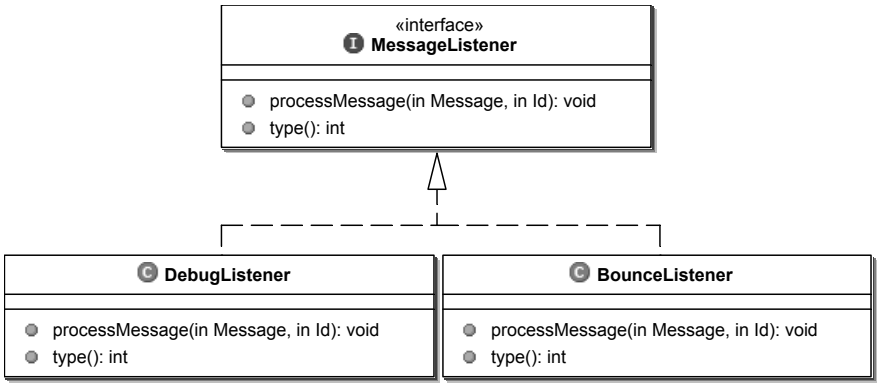
Package ***com.hp.unitec.p2p.chord*** *(Fortsetzung)*

Ferner enthält das Package Klassen, die entfernte Aufrufe von Chord-Funktionen mittels RPC ermöglichen.



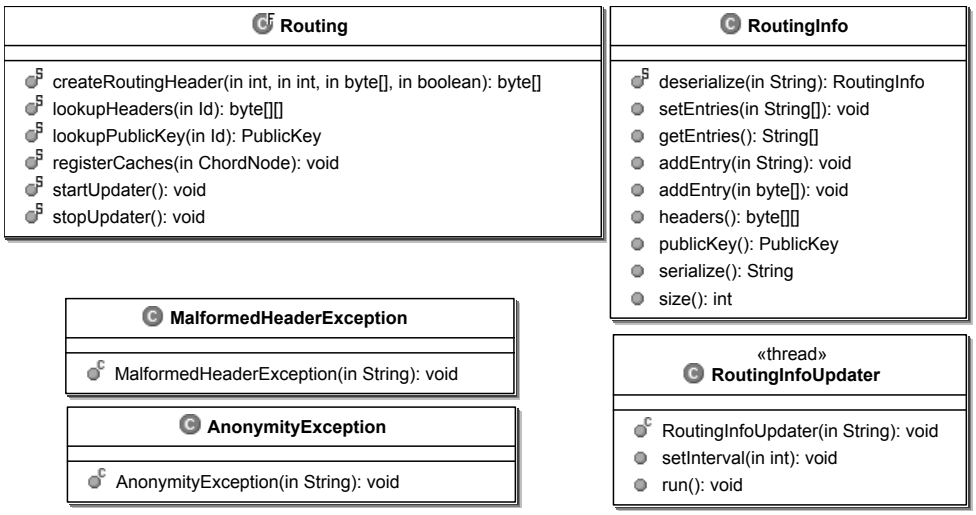
Package ***com.hp.unitec.p2p.listener***

Klassen, die das `MessageListener`-Interface implementieren, bestimmen die Reaktion auf eine eintreffende Nachricht. Dieses Package enthält zwei Beispielklassen. `DebugListener` gibt die Nachricht an der Konsole aus, `BounceListener` sendet sie zurück an den Sender.



Package ***com.hp.unitec.p2p.routing***

Enthält die Klassen, die zur Erstellung der Routing-Header und zur Durchführung des regelmäßigen Routing-Updates verantwortlich sind.



Package ***com.hp.unitec.p2p.mix***

Dieses Package beinhaltet Klassen, die mit der Mix-Funktionalität zusammenhängen – sowohl für den Mix-Dienst als auch für den Mix-Client. Die Kernfunktionalität ist in der `Mix`-Klasse enthalten.

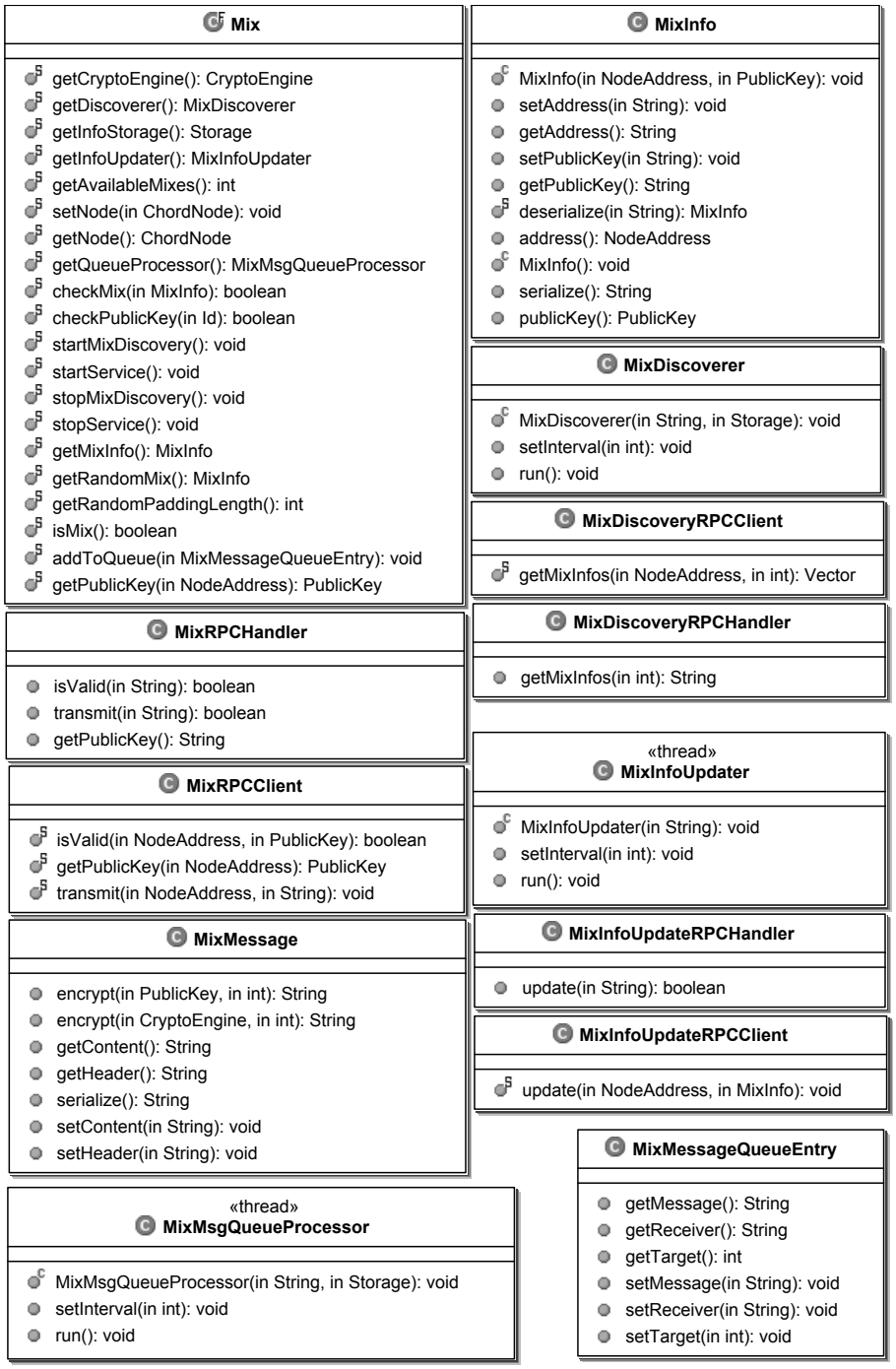
Um Angriffe zu erschweren, die auf Pattern Matching und statistischer Analyse basieren, wird eine Hop-to-Hop-Verschlüsselung mit zufälligem Padding durchgeführt (siehe Abschnitt 4.5). Diese wird in der Klasse `MixMessage` implementiert.

Ferner enthält das Package die Klasse `MixInfo`, die zur Speicherung von Informationen über bekannte Mixe benötigt werden, sowie aktive Komponenten wie den `MixInfoUpdater` und den `MixDiscoverer`, die für die Gewährleistung der Verfügbarkeit von Mix-Informationen im Netz verantwortlich sind.

Die Kommunikation wird mittels entsprechender RPC-Client- und RPC-Handler-Klassen realisiert.

Schließlich implementiert `MixMsgQueueProcessor` die regelmäßige Bearbeitung der Einträge, die in der Mix-Warteschlange auf das weitere Versenden warten. Die Einträge werden in zufälliger Reihenfolge versendet.

Package ***com.hp.unitec.p2p.mix*** *(Fortsetzung)*

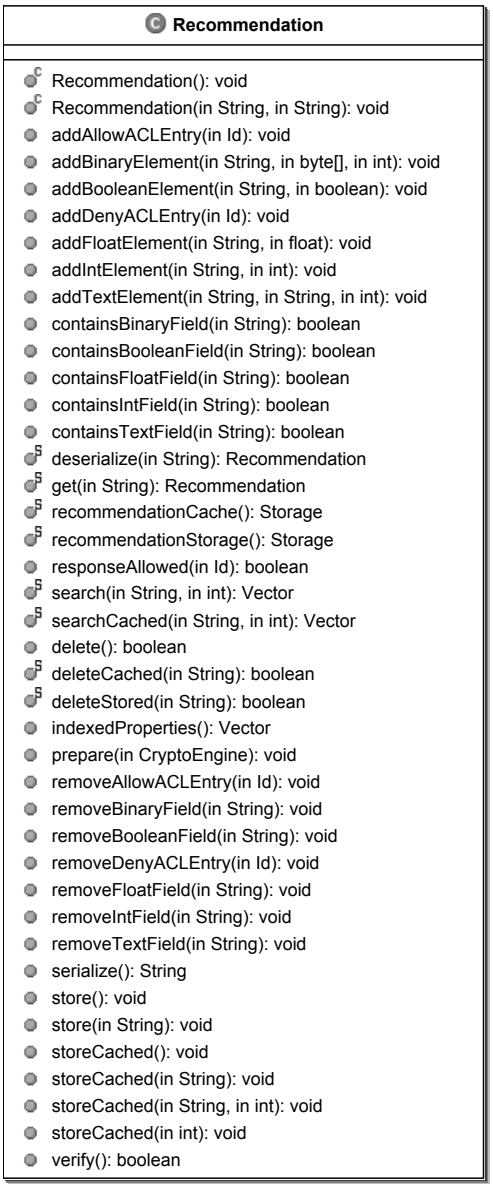


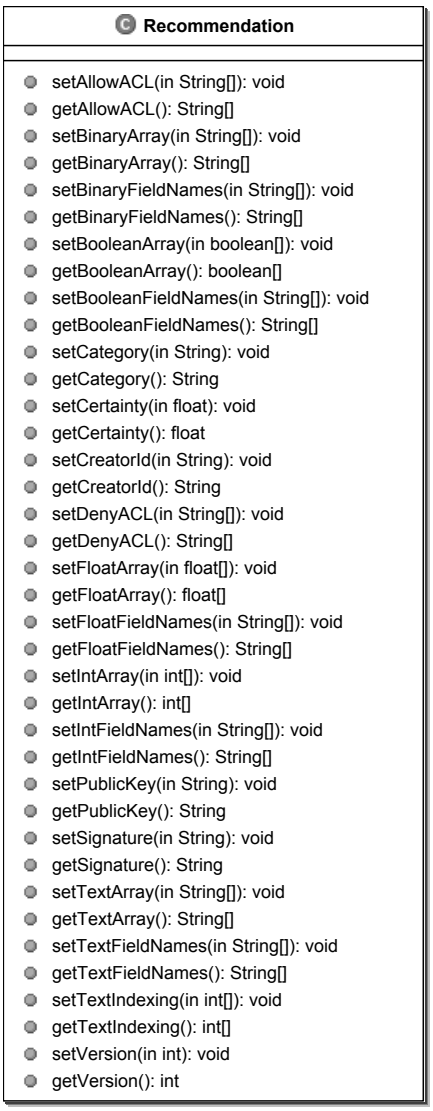
Klassen des Empfehlungsdiensts

Package ***com.hp.unitec.recmgr***

Die `Recommendation`-Klasse enthält zum einen die Daten einer Empfehlung und bietet Zugriffsmethoden für diese. Ferner stellt sie Methoden zur Erstellung von Empfehlungen, zur Berechnung und Überprüfung von Signaturen, aber auch zur Verwaltung des Empfehlungsmodells zur Verfügung.

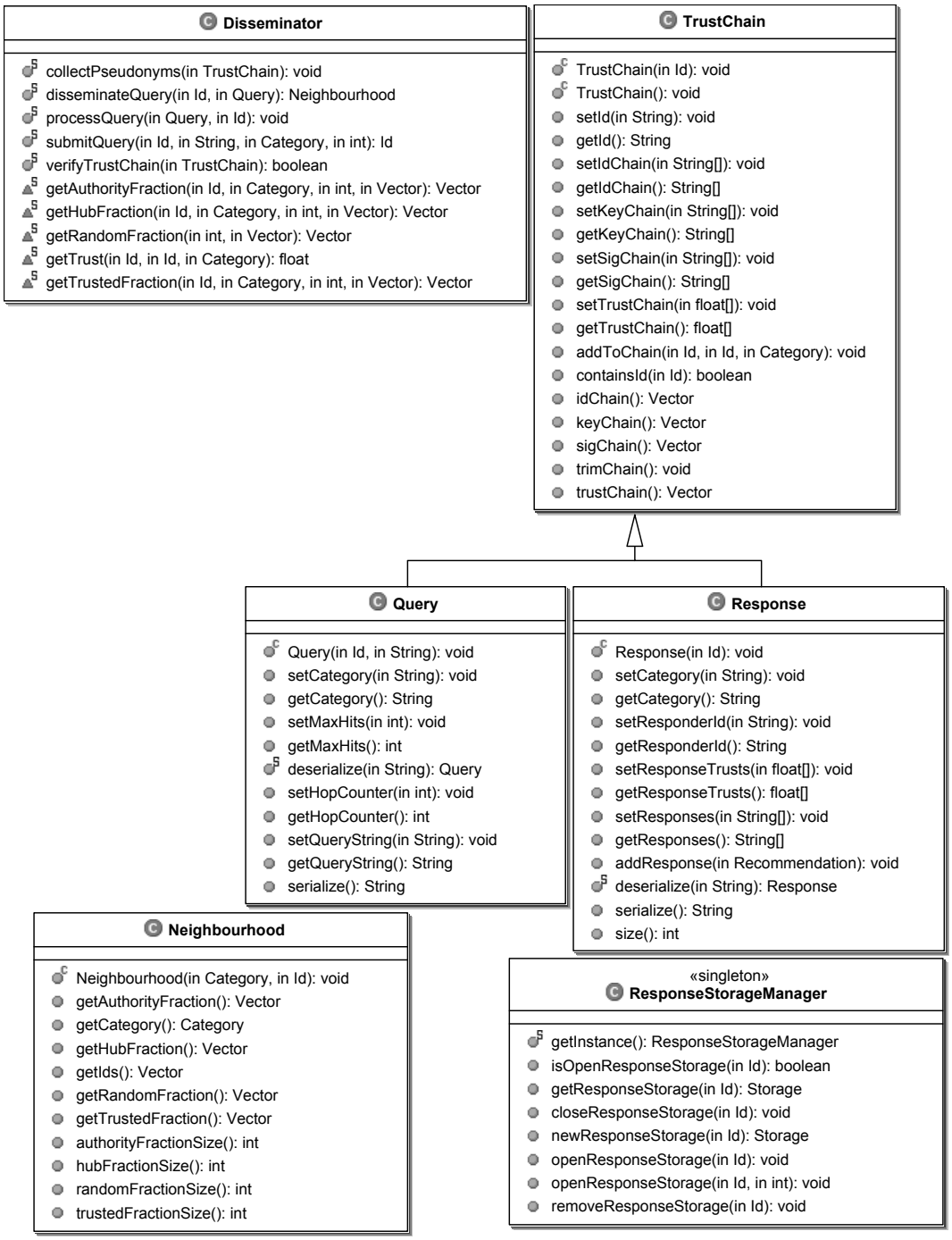
Der Übersichtlichkeit halber wurden in der folgenden Abbildung die Accessor- und Mutator-Methoden getrennt dargestellt.





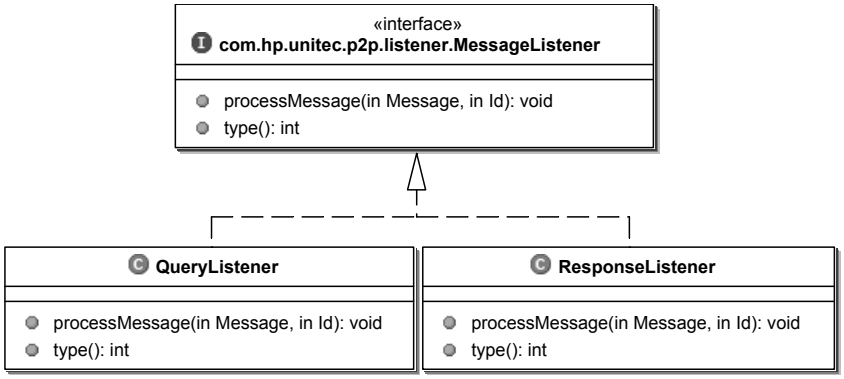
Package ***com.hp.unitec.recmgr*** *(Fortsetzung)*

In den folgenden Klassen ist die Funktionalität des Empfehlungsdiensts enthalten: die Formulierung und Weitergabe von Anfragen, die Berechnung einer Nachbarschaft, die Verwaltung von Vertrauensketten, die Beantwortung eintreffender Anfragen sowie die Speicherung eintreffender Antworten.



Package ***com.hp.unitec. recmgr.listener***

Das Package enthält Message-Listener, die auf das Eintreffen von Anfragen und Antworten reagieren.



Literaturverzeichnis

[Aich03] **Aichele, F.:** *Realisierung und Evaluierung von Trust-Update-Algorithmen für das UniTEC Reputationssystem*. Studienarbeit, Universität Stuttgart, Fakultät für Informatik, Elektrotechnik und Informationstechnik, IPVS (2003)

[Chau81] **Chaum, D.:** *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms.* In: *Communications of the ACM, Volume 24, Number 2, February 1981* (1981)

[Chau88] **Chaum, D.:** *The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability.* In: *Journal of Cryptology, Volume 1* (1988)

[CSWH01] **Clarke, I., Sandberg, O., Wiley, B., Hong, Th. W.:** Freenet: *A distributed anonymous information storage and retrieval system.* In: **Federrath, H. (ed.):** *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability.* LNCS 2009, Springer-Verlag (2001)

[DKK+01] **Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., Stoica, I.:** *Wide-area cooperative storage with CFS.* In: *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, *Chateau Lake Louise, Banff, Canada* (2001)

[Elec98] **Electronic Frontier Foundation, A.:** *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*. O'Reilly & Associates (1998)

[FrMo02] **Freedman, M. J., Morris, R.:** *Tarzan: A Peer-to-Peer Anonymizing Network Layer.* In: *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002), Washington, D.C., November 2002* (2002)

[Gern02] **von Gernler, A.:** *Chord und CFS: Ein kooperatives, verteiltes Dateisystem.* Hauptseminar *Moderne Konzepte für weitverteilte Systeme: Peer-to-Peer-Netzwerke und fehlertolerante Algorithmen*, Universität Erlangen, Institut für Informatik (2002)

[GRS99] **Goldschlag, D., Reed, M., Syverson, P.:** *Onion routing for anonymous and private internet connections.* In *Communications of the ACM, Volume 42* (1999)

[Hami97] **Hamilton, G. (ed.):** *JavaBeans™ Specification*. Sun Microsystems, Version 1.01-A (1997)

http://java.sun.com/products/javabeans/docs/spec.html (26.10.2003)

[Haro02] **Harold, E. R.:** *Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX*. Addison-Wesley, 1. Auflage (2002)

[KiPe03] **Kinateder, M.; Pearson, S.:** *A Privacy-Enhanced Peer-to-Peer Reputation System.* In: **Bauknecht, K. (ed.), Tjoa, A M. (ed.), Quirchmayr, G. (ed.):** *Proc. of the 4th International Conference on Electronic Commerce and Web Technologies: EC-Web 2003; Prague, Czech Republic, September 2003.* LNCS 2738, Springer-Verlag (2003)

[KiRo03] **Kinateder, M., Rothermel, K.:** *Architecture and Algorithms for a Distributed Reputation System.* In: **Nixon, P., Terzis, S. (eds.)**: *Proc. of the First International Conference on Trust Management, iTrust 2003, Heraklion, Crete, Greece, May 28-30, 2003*. LNCS 2692, Springer-Verlag (2003)

[Klei98] **Kleinberg, J. M.:** *Authoritative Sources in a Hyperlinked Environment*. In: *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 1998* (1998)
http://www.cs.cornell.edu/home/kleinber/auth.pdf (21.10.2003)

[MOV96] **Menezes, A. J., van Oorshot, P. C., Vanstone, S. A.:** *Handbook of Applied Cryptography*, CRC Press (1996)

[MMH02] **Mui, L., Mohtashemi, M., Halberstadt, A.:** *A Computational Model of Trust and Reputation*. In: *Proc. of the 35th Hawaii International Conference on System Sciences*, Big Island, Hawaii, IEEE (2002)

[Oram01] **Oram, A. (ed.):** *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 1. Auflage (2001)

[Pfit97] **Pfitzmann, A:** *Datensicherheit und Kryptographie*, Vorlesungsskript, TU Dresden (1997)
http://dud.inf.tu-dresden.de/lectures.shtml (21.10.2003)

[ReRu98] **Reiter, M. K., Rubin, A. D.:** *Crowds: Anonymity for Web Transactions.* In: *ACM Transactions on Information and System Security, Volume 1, Number 1, November 1998* (1998)

[RoDr01] **Rowstron, A., Druschel, P.:** *Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-peer Systems.* In: *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001). Heidelberg, Germany, November 2001.* LNCS 2218, Springer-Verlag (2001)

[RSA78] **Rivest, R. L., Shamir, A., Adleman, L.:** *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* In: *Communications of the ACM 2/1978* (1978)

[Schn96] **Schneier, B.:** *Applied Cryptography. Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, 2. Auflage (1996)

[Stam03] **Stamer, H.:** *Dining Cryptographers Networks. Praktikable Werkzeuge zur Wahrung der Anonymität?.* Diplomarbeit Nr. 2003-10, Universität Leipzig, Institut für Informatik (2003)

[SMK+01] **Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.:** *Chord: A scalable peer-to-peer lookup service for internet applications.* In: *Proceedings of the ACM SIGCOMM '01 Conference* (2001)

[SSL96] **Transport Layer Security Working Group:** *The SSL Protocol Version 3.0* (1996)

http://wp.netscape.com/eng/ssl3/draft302.txt (21.10.2003)

[Weni03] **Wenig, M.:** *Entwicklung eines Konzepts zur Volltextsuche in semantischen Netzen*. Diplomarbeit Nr. 2078, Universität Stuttgart, Fakultät für Informatik, Elektrotechnik und Informationstechnik, IPVS (2003)

[Widm03] **Widmaier, J.:** *Identitäts-Management in Peer-2-Peer-Umgebungen*. Diplomarbeit Nr. 2094, Universität Stuttgart, Fakultät für Informatik, Elektrotechnik und Informationstechnik, IPVS (2003)

[ZKJ01] **Zhao, B.Y., Kubiatowicz, J., Joseph, A. D.:** *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.* In: *Report No. UCB/CSD-01-1141*, U. C. Berkeley, Computer Science Division (2001)

Stichwortverzeichnis

A

ACL *Siehe* Zugriffskontrollliste
Advanced Encryption Standard 77
AES *Siehe* Advanced Encryption Standard
Agent **10**
Analyzer 59
Anfrage **11**, 46, 49
  Formulierung 49
  Reaktion 12
  Weiterleitung 51
Angriff
  Brute Force-Angriff **32**
  Denial of Service-Angriff 44, 53
  First Mix-Angriff **41**
  Last Mix-Angriff **40**
  Pattern Matching-Angriff **38**
  Replay-Angriff **36**
Anonymität 11
Anonymitätsgrad 64
Antwort 47, 52
  Empfang 54
Apache
  Jakarta-Projekt 58, 60
  log4j 60
  Lucene 58
  Xindice 58
  XML-RPC 59, 63
Authority **8**, 46, 50

B

Base64 64, 71
Bewertung **5**
Bootstrap-Adresse 63
Bouncy Castle Crypto API 60
Brute Force-Angriff **32**

C

Cache 62, 71
Chord **21**, 63
  Implementierung 63
Crowds **18**

D

Datencontainer 61, 62, 71
Datenhaltung 47, 61
Datenmodelle **6**
DC-Netze **16**
Denial of Service 44, 53, 56
Destination Routing **28**
DHT *Siehe* verteilte Hashtabelle
distributed hashtable *Siehe* verteilte Hashtabelle
Document Object Model 57
DOM *Siehe* Document Object Model
Dummy-Lookup **37**

E

Empfehlung **6**, 45
  Struktur 48
Empfehlungsdienst 45
Empfehlungsmodell **7**, 47
Erfahrung **5**
Erreichbarkeit **52**
Ersetzungsstrategie 62
Expertisemodell **8**
Extended Destination Routing 30, **38**

F

Feld 48
First Mix-Angriff **41**
Flooding **15**
Free Haven **20**
Freenet **19**

H

Hashfunktion **78**
Hashwert **78**
Hop-Counter **11**, 46, 49, 54, **108**
Hop-to-Hop-Verschlüsselung **40**
Hop-to-Hop-Verschlüsselung mit Padding **40**, 41
Hub **8**, 46, 50

I

ID 21, 31
  Erzeugung durch Hashing 31
  Kollision 32
  Pseudonym-ID **31**
Interessensgebiet 55

J

Java Cryptography Extension 60
JavaBeans 57, 64
JCE *Siehe* Java Cryptography Extension
JCE-Provider 60
Jondo 18
JOX-Framework 58, 64

K

Kategorie **6**
Kollision **32**
Kollisionsvermeidung **32**
Konfidenzwert 45

L

Least Frequently Used 48
least recently used 62
Least Recently Used 48
LFU *Siehe* Least Frequently Used
Listener 64
Load-Wert **43**
log4j 60
Logging **60**
Lookup **33**, 37
Lookup-Proxy **37**
LRU *Siehe* Least Recently Used
Lucene 58

M

maximale Lebensdauer 62
Message Listener 64
MI-Liste *Siehe* Mix-Informationsliste
Mix **16**
  Discoverer 64
  Discovery **43**
  Information Updater 64
  Mix-Kaskade **16**
Mix-Informationen **42**
  Verfügbarkeit **43**
Mix-Informationsliste **42**, 64
Mix-Netze **16**

N

Nachbarschaft **12**
neighborhood *Siehe* Nachbarschaft

O

Onion Routing **16**, 28, 41

P

Pattern Matching 41
Pattern Matching-Angriff **38**
Peer-to-Peer-Netz 10
Pseudonym **5**
  Anonymitätsgrad 64
  ID 31

R

reachability **52**
Remote Procedure Call **59**
Replay-Angriff **36**
Routing Information Updater 64
Routing Update **34**
Routing-Header 29
  Erweiterung **38**
RPC *Siehe* Remote Procedure Call
RSA 60, **78**

S

SHA-1 32, **78**
Signatur **78**
Source Routing 28
SSL 1
Stemmer 59
Stopwort 59
Storage Manager 62

T

Tarzan **20**
time to live 62
Token 59
trust chain *Siehe* Vertrauenskette
Trust Update **13**, 45
TTL *Siehe* time to live

U

UniTEC **2**
  Architectur 8
  Modell 10
  Systemmodell 5
Update-Nachricht **35**
Update-Proxy **34**

V

verteilte Hashtabelle **20**, 33, 63
Vertrauen **6**
Vertrauenskette **12**, 50
Vertrauensmodell **7**
Verzeichnisdienst 29, **33**
Volltextanfrage 48, 49
Volltextanfragen
  Syntax 79
Volltextsuche 61

X

XML **57**
  Datenbanken **58**
  Persistenz 61
  Serialisierung **57**, 61
XML-RPC 59
XPath 58

Z

zentralisiertes Modell 9
Zugriffskontrolle 47, **53**
Zugriffskontrolliste 53

Glossar

Agent

Instanz der Software, mit dessen Hilfe der Benutzer an der Kommunikation im UniTEC-System teilnimmt und dessen Funktionalität nutzt.

Anfrage

Nachricht, die ein Interessent im Kontext einer oder mehrerer Kategorien formuliert, mit dem Ziel, Empfehlungen zu finden, die den formulierten Kriterien entsprechen. Im UniTEC-System werden Anfragen als Volltextsuche formuliert.

Anonymität

Ziel der Anonymität ist es, jegliche persönlichen Daten des Benutzers vor der Einsicht anderer zu schützen. In der vorliegenden Arbeit wird sowohl Sender- als auch Empfängeranonymität angestrebt, um die unbeabsichtigte Aufdeckung der Identität eines Benutzers zu verhindern.

Authority

Pseudonym, dessen Expertise im Zusammenhang mit der Beantwortung einer Anfrage besonders relevant ist. Die Relevanz wird anhand der vom jeweiligen Pseudonym bereits beantworteten Anfragen in einer bestimmten Kategorie berechnet.

Bootstrap-Knoten

Knoten, der sich bereits im Peer-to-Peer-Netz befindet, dessen Adresse für den Beitritt eines neuen Agenten benötigt wird.

Brute Force-Angriff

Angriff, bei dem versucht wird, durch das Durchprobieren aller möglichen Kombinationen geheime Daten aufzudecken.

Denial of Service-Angriff

Gezielter Angriff auf ein Pseudonym oder auf einen Agenten, um dessen Erreichbarkeit bzw. fehlerfreien Betrieb zu beeinträchtigen.

Destination Routing

Verfahren, bei dem der Empfänger einer Nachricht deren Weg bestimmt, mit dem Ziel, die Empfängeranonymität zu gewährleisten.

Empfehlung

Subjektive Beschreibung einer Erfahrung, die mit einem bestimmten Gegenstand, etwa einem Produkt oder einer Dienstleistung, gemacht wurde.

Empfehlungsmodell

Datenmodell innerhalb des Agenten, das sowohl Empfehlungen enthält, die lokal vom Benutzer des Agenten erstellt wurden, als auch solche, die als Antworten auf Anfragen erhalten und anschließend zwischengespeichert wurden.

Expertisemodell

Datenmodell, das Statistiken zur Expertise bekannter Pseudonyme enthält. Anhand dieser Daten werden Hubs und Authorities ermittelt.

Extended Destination Routing

Verfahren, bei dem der Sender einer Nachricht die Mix-Kaskade erweitert, um einen gewünschten Anonymitätsgrad zu erreichen und um zu verhindern, dass böswillige Empfänger dessen Anonymität kompromittieren.

Hashfunktion

Nicht umkehrbare Funktion, die die Abbildung umfangreicher Mengen beliebig großer Elemente auf eine Menge von Elementen begrenzter Länge realisiert.

Hop-Counter

Numerischer Wert, mit dem Anfragen versehen werden, um die Anzahl der durch die Anfrage erreichten Pseudonyme zu begrenzen und zu verhindern, dass die Anfrage nicht unendlich oft weitergesendet wird. Hierzu wird dieser Wert vor jeder Weiterleitung der Anfrage dekrementiert. Bei Erreichen des Werts 0 wird die Anfrage verworfen.

Hub

Pseudonym, das Verbindungen zu zahlreichen Authorities hat. Diese Verbindungen können direkt oder indirekt (über andere Hubs) sein.

ID

Eindeutiges Merkmal, durch das eine Instanz, etwa ein Pseudonym oder ein Knoten im Netz, von anderen Instanzen der gleichen Klasse unterschieden werden kann.

Lookup

Nachschlagen der Routing-Informationen eines Pseudonyms im Verzeichnisdienst.

Mix

Zwischenstation, die durch Umcodieren mit Hilfe von Datenverschlüsselung und durch Ändern der Reihenfolge der bearbeiteten Nachrichten verhindert, dass die Route einer Nachricht nachvollzogen werden kann.

Mix-Informationsliste

Liste, in der aktuelle Informationen zu bekannten Mixen verwaltet werden. Zu diesen zählen die Adresse und der öffentliche Schlüssel des Mixes.

Mix-Dienst

Dienst, der auf einem Agenten gestartet wird, damit dieser im Netz als Mix fungiert.

Mix-Kaskade

Folge von Mixen, die zur Wahrung der Anonymität vom Sender bzw. vom Empfänger ausgewählt wird. Sie bestimmt die Route der Nachricht zwischen Sender und Empfänger.

Nachbarschaft

Menge von Pseudonymen, von denen erwartet wird, dass sie eine Anfrage auf zufrieden stellende Weise beantworten können bzw. an solche Pseudonyme weiterleiten können.

Peer-to-Peer

Dezentrales Kommunikationsmodell ohne Hierarchien, d.h. die Teilnehmer eines Peer-to-Peer-Netzes sind gleichberechtigt. Hierbei sind sie sowohl Nutzer als auch Anbieter verschiedener Dienste.

Pattern Matching-Angriff

Angriff auf die Anonymität eines Benutzers, mit dem Ziel, verschlüsselte Nachrichten anhand von Mustern wieder zu erkennen, deren Route zu verfolgen und dadurch die Identität des angegriffenen Benutzers aufzudecken.

Pseudonym

Fiktive Identität, in dessen Kontext der Benutzer das System verwendet, etwa durch die Verwendung eines zufälligen Namens, mit dem Ziel, die persönlichen Daten zu schützen. Benutzer können gleichzeitig mehrere Pseudonyme verwenden.

Routing-Header

Informationen, die zur Routenbestimmung verwendet werden mit dem Ziel, die Anonymität der Teilnehmer an der Kommunikation zu wahren.

Routing-Information

Beinhaltet einen oder mehrere Routing-Header.

Routing Update

Vorgang, bei dem Routing-Header eines Pseudonyms basierend auf aktuellen Mix-Informationen im Verzeichnisdienst gespeichert werden.

Trust Update

Aktualisierung der Vertrauensbeziehung zwischen Benutzern anhand positiver oder negativer Bewertung von Empfehlungen.

UniTEC

Abkürzung für *Universal Trust Architecture for Electronic Commerce*; Rahmenprojekt der vorliegenden Arbeit, das als primäres Ziel die Erhöhung der Akzeptanz des elektronischen Handels anstrebt.

Update-Proxy

Zufälliger Knoten, der als Stellvertreter beim Routing Update eines Pseudonyms verwendet wird, um die Anonymität des

Verteilte Hashtabelle

Verteilte Datenstruktur, die effiziente Speicherung und Zugriff auf Daten in einem Peer-to-Peer-Netz ermöglicht.

Vertrauenskette

Bevor eine Anfrage an ein Pseudonym aus der Nachbarschaft weitergeleitet wird, wird der zu diesem Pseudonym gehörende Vertrauenswert in der entsprechenden Kategorie zusammen mit der ID des Pseudonyms zur Anfrage hinzugefügt. Durch rekursive Weiterleitung der Nachricht entsteht eine Reihe von ID-Wert-Paaren, die *Vertrauenskette* genannt wird. Beantwortet ein Pseudonym eine Anfrage, so wird die Vertrauenskette aus der Anfrage in die Antwortnachricht kopiert, so dass sie vom Anfragenden ausgewertet werden kann.

Vertrauenswert

Wert, der das Vertrauen zwischen zwei Pseudonymen im Kontext einer bestimmten Kategorie quantifiziert.

Vertrauensmodell

Datenmodell, das zum einen die semantischen Beziehungen zwischen verschiedenen Kategorien, zum anderen aber auch das Vertrauen zwischen Pseudonymen enthält.

Verzeichnisdienst

Dienst, der für die Speicherung der Routing-Informationen und öffentlichen Schlüssel von Pseudonymen verantwortlich ist. In der vorliegenden Arbeit wird dieser Dienst mit Hilfe dezentraler Speicherung in einer verteilten Hashtabelle realisiert.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Stuttgart, den 31.10.2003

(Ralf Terdic)