Universität Karlsruhe (TH)

# Entwicklung einer generischen Instant Messenger Anwendung für Peer-to-Peer Netze

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

von

cand. inform.
**Andreas Schuster**

Betreuer:
Prof. Dr. habil. Martina Zitterbart
Dipl.-Ing. Kendy Kutzner

Tag der Anmeldung: 15. Februar 2004
Tag der Abgabe: 15. April 2004

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

Karlsruhe, den 15. April 2004

# Inhaltsverzeichnis

# 1. Einleitung

Seit einigen Jahren steigt die Popularität sogenannter Peer-to-Peer Netze nahezu ungebremst. Hauptanwendungszweck der meisten Peer-to-Peer Anwendungen stellt nach wie vor das File-Sharing dar. Doch das ist bei weitem nicht die einzig denkbare Anwendung.
Im Rahmen dieser Arbeit soll die Eignung von Peer-to-Peer Netzen für weitere Zwecke anhand der Beispielanwendung des Instant Messaging ausgelotet werden.

## 1.1 Zielsetzung der Arbeit

Ziel der Arbeit ist es, Peer-to-Peer Netze um eine generische Instant Messaging Funktionalität zu erweitern.

Wo möglich soll dabei auf schon bestehende Lösungen zurückgegriffen werden, sowohl bei der Nutzung von bestehenden Peer-to-Peer Clients als auch soweit sinnvoll bei der Oberfläche für einen Instant Messenger.
Zu Beginn steht daher eine Analyse verschiedener Peer-to-Peer Anwendungen sowie der Fähigkeiten von verbereiteten Instant Messengern.
Darauf aufbauend soll bei Entwurf und Implementierung Wert auf ein größtmöglich modulares Vorgehen gelegt werden, um einen späteren einfachen Austausch der Komponenten Peer-to-Peer Netz sowie der Benutzeroberfläche zu ermöglichen, ohne dabei in größerem Umfang Veränderungen an der zu erstellende Zwischenschicht vornehmen zu müssen.
Abschließend soll die Implementierung mit variierten Entwurfsparametern getestet werden und eine Bewertung der Designentscheidungen in Hinblick auf das zugrunde liegende Peer-to-Peer Netz vorgenommen werden.

## 1.2 Gliederung der Arbeit

Die Arbeit beginnt mit einer Analyse des Ist-Zustandes in Kapitel 2. Dabei werden zunächst verbreitete Instant Messaging Anwendungen vorgestellt und danach die technischen Hintergründe zur Realisierung von Instant Messaging Systemen erläutert.

Dann werden die grundlegenden Merkmale von Peer-to-Peer Anwendungen erklärt und für verschiedene Gattungen von Peer-to-Peer Netzen anhand einer existierenden Anwendung deren Funktionsweise beschrieben. Das Kapitel schließt letztlich mit einer Betrachtung der Vor- und Nachteile der jeweiligen Gattungen für bestimmte Einssatzszenarien.

In Kapitel 3 wird ein Entwurf für die zu erstellende generische Instant Messaging Anwendung für Peer-to-Peer Netze erstellt.
Das beginnt mit der Suche nach geeigneten existierenden Anwendungen für die Peer-to-Peer- und Oberflächen-Komponente. Daraufhin wird mit der Definition einer generischen Schnittstelle für diese Komponenten dem Ziel der Modularität mit einfach austauschbaren Komponenten nachgekommen.
Der letzte Teil des Kapitels beschreibt dann den Entwurf der eigentlichen Anwendung, die die generische Instant Messaging Funktionalität für Peer-to-Peer Netze erbringt.

Die Implementierung der zu erstellenden Anwendung, die den Namen „IM4P2P“ tragen wird, ist in Kapitel 4 dokumentiert.
Zunächst liegt dabei das Augenmerk auf der modularen Integration der Peer-to-Peer Anwendung sowie der grafischen Oberfläche in ein eigens zu diesem Zwecke entworfenes Framework.
Danach wird die Implementierung der generischen Zwischenschicht betrachtet, die den Kern von IM4P2P bildet.

In Kapitel 5 wird die erstellte Instant Messaging Lösung für Peer-to-Peer Netze schließlich in Betrieb genommen.
Nach einer kurzen Einführung in die notwendigen Konfigurationseinstellungen, wird die Anwendung mit verschiedenen Parametern getestet und die dabei erzielten Resultate bewertet.

Abschließend wird in Kapitel 6 eine Zusammenfassung der erarbeiteten Erkenntnisse und ein Ausblick auf zukünftige Möglichkeiten und Entwicklungen gegeben.

# 2. Analyse

Im Kapitel Analyse sollen die allgemeinen Konzepte der der Studienarbeit zugrunde liegenden Technologien erläutert werden und im Rahmen einer Bestandsaufnahme der aktuelle Stand der Entwicklung in Sachen Instant Messaging und Peer-to-Peer Technologie dokumentiert werden.

## 2.1 Was ist Instant Messaging?

Instant Messaging (IM) ist eine recht junge Anwendung im Internet, der erste öffentlich verfügbare IM Client tauchte Ende 1996 auf.

Auf Deutsch übersetzt heißt Instant Messaging soviel wie „augenblicklicher Nachrichtenaustausch“. Die Hauptanwendung von Instant Messaging besteht dann auch in nahezu in Echtzeit ablaufendem Text-Chat zwischen jeweils zwei Endnutzern.
Zusätzlich dazu erlauben nahezu alle IM Applikationen den Anwenderen einen sogenannten Onlinestatus zu setzen, den andere IM Clients abfragen können. So lässt sich z. B. erkennen ob andere Anwender zur Zeit ebenfalls im IM Netz eingebucht sind (Status: Online) oder nicht (Status: Offline) oder seit längerem nicht mehr am Rechner sitzen was die IM Clients anhand fehlender Mausbewegungen oder Tastatureingaben erkennen (Status: Away).

Ein entscheidender Nutzen von IM ergibt sich aber aus der Tatsache, dass heutzutage die meisten Endnutzer per Einwahlverbindung ohne feste IP-Adresse ans Internet angeschlossen sind. Daher ist es für entfernte Rechner nicht einfach festzustellen ob eine bestimmte Person online ist und wenn ja hinter welcher Adresse sich ihr Rechner aktuell denn verbirgt. Für dieses Problem eine Lösung zu bieten, ist wohl einer der Hauptgründe für den Erfolg von Instant Messaging.

### 2.1.1 Allgemeines

Die meisten Instant Messenger besitzen eine grafische Oberfläche die sich in zwei Bestandteile gliedern lässt (siehe Abbildung 2.1): Eine Kontaktliste (auch Buddy Liste genannt) in der alle einem bekannten Personen aufgelistet sind und außerdem deren aktueller Onlinestatus angezeigt wird. Darüber hinaus werden bei Bedarf ein


Abbildung 2.1: Nachrichtenfenster und Kontaktliste am Beispiel des Gaim IM Client [Gaim]

oder mehrere Nachrichtenfenster dargestellt, in dem der bisherige Gesprächsverlauf mit einzelnen Kontakten aufgeführt wird.

In [GrPa02] wurden unter Jugendlichen zwei grundsätzlich unterschiedliche Nutzungsprofile identifiziert: Personen die sich den PC mit anderen Menschen teilen oder keine ständige Anbindung ans Internet besitzen nutzen IM sehr intensiv und zielstrebig und mit meist weniger als 3 Stunden am Tag Onlinezeit. Währenddessen lassen Nutzer mit der Möglichkeit einer ständigen Internetverbindung und eigenem PC häufig die IM Applikation auch während anderer Aktivitäten am Computer und auch in Abwesenheit weiterlaufen und antworten daher nur sporadisch über den Tag verteilt auf Benachrichtigungen.

Im Gegensatz zu Internet Relay Chat (IRC) in dem sich die Nutzer oft nicht im echten Leben begegnen, kennen sich IM Nutzer meistens auch persönlich. Das mag daran liegen das über IRC meist themenbezogene Nutzergruppen zusammenfinden, während sich persönliche 1:1 Chats per IM eher bei Personen ergeben die sich auch privat kennen.

Instant Messaging hat sich mittlerweile nicht nur für Privatkommunikation etabliert, es findet auch in firmeninternen Abläufen häufig Verwendung und beschleunigt durch die Möglichkeit zu direkten und informellen Rückfragen den Arbeitsablauf [NaWB00].
Allerdings sollten die schädlichen Auswirkungen durch die ständig möglichen Unterbrechungen von eingehenden Nachrichten bei einer Entscheidung für oder gegen

IM in Arbeitsprozessen und auch beim Design Groupware-orientierter IM Systeme nicht vergessen werden [CzCH00].

Häufig bieten IM Clients bzw. IM Netze neben Text-Chat und Onlinestatusanzeige auch weitere Dienste, darunter die Möglichkeit sogenannte Offline Messages an derzeit nicht eingebuchte IM Clients zu senden oder Dateien an Kontakte zu veschicken. Oft wird auch eine Kopplung zu externen Programmen geboten, die auf die Kontaktinformationen aus dem IM Netz zurückgreifen um beispielsweise Voice Over IP Anwendungen, Video Chat Programme oder Multiplayer Spiele zu starten.

Zu den wohl populärsten proprietären Instant Messaging Anwendungen gehören derzeit ICQ [ICQ], der MSN Messenger [MSNM] und der AOL Instant Messenger (AIM) [AIM]. Neben den Originalclients der jeweiligen Instant Messaging Netzbetreiber existieren zahlreiche weitere häufig quelloffene Clients die teilweise auch mehrere Protokolle nebeneinander bedienen können.
Als Beispiel einer auf einem vollständig offenenem Protokoll basierenden Instant Messaging Anwendung wird später Jabber [Jabb] näher betrachtet werden.

### 2.1.2 ICQ

Der Durchbruch für die weite Verbreitung von Instant Messaging ist eng mit der Erfolgsgeschichte der israelischen Start-Up Firma Mirabilis verknüpft [ICQS]. Im November 1996 veröffentlichte Mirabilis die erste Version von ICQ (sprich „I seek you“) zum freien Download im Internet. Innerhalb nur eines halben Jahres registrierten sich bei Mirabilis bereits 850000 Nutzer.
Im Juni 1998 wurde Mirabilis dann von AOL aufgekauft und später wurde das ICQ Protokoll mit dem des AOL eigenen Konkurrenzprodukts AIM verschmolzen.
Die ICQ Homepage spricht heute von über 160 Millionen registrierten Benutzern weltweit.

Um einen eigenen Adressraum für die IM Clients zu schaffen, der von der dynamischen Natur der IP-Adressen der Nutzer abstrahiert, nutzt ICQ eine zentrale Server-Infrastruktur.
Alle ICQ Protokolle unterscheiden daher grundsätzlich die Kommunikation zwischen Client und Server von der Kommunikation zwischen Clients.

Ältere ICQ Protokolle bis einschließich Version 5 [Isak01] nutzen UDP Nachrichten für die Kommunikation zwischen Server und Client. Das neuere OSCAR („**O**pen **S**ystems for **C**ommunic**a**tion in **R**ealtime“) ist als Ergebnis einer Zusammenführung der Protokolle der beiden im Besitz von AOL befindlichen IM Clients (ICQ und AIM) entstanden. OSCAR verwendet Datagramm-ähnliche in ein eigenes Protokoll namens FLAP gekapselte Nachrichten über eine TCP Verbindung [Shut],[Kuhl].
Allen Protokollversionen gemeinsam sind folgende Funktionsmerkmale: Durch eine Authentifzierung via Passwort beim Server melden sich die Clients im IM Netz als erreichbar an. Außerdem gleichen sie mit dem Server die Kontaktliste ab, die je nach Version Server-seitig oder Client-seitig abgespeichert werden kann. Danach holen sie eventuell in Abwesenheit aufgelaufene Offline Messages ab.
Um zu erkennen wenn ein Client vom Netz getrennt wurde ohne sich beim Server abzumelden, meldet der Client sich regelmäßig mit Keep-Alive Nachrichten beim Server. Bleiben diese Nachrichten aus, nimmt der Server nach einer gewissen Timeout Zeitspanne an, dass der betreffende Client vom Netz getrennt wurde und

setzt seinen Status auf Offline.
Darüber hinaus sind die Server in der Lage Nachrichten zwischen Clients weiterzuleiten zwischen denen keine direkte Verbindung etabliert werden kann, weil z. B. restriktive Firewallregeln keine eingehenden Verbindungen auf Ports zulassen. Nachrichten an Kontakte die derzeit nicht im IM Netz eingebucht sind werden als sogenannte Offline Messages auf dem Server zwischengespeichert und beim nächsten Einbuchen von den entsprechenden Addressaten am IM Server abgeholt.

Für die Kommunikation zwischen zwei ICQ Clients wird eine direkte TCP Verbindung zwischen diesen etabliert. Die jeweils aktuelle Adresse des Kommunikationspartners erfährt der den Kommunikationssvorgang initiierende IM Client vom Server. Beim Aufbau dieser Verbindung wird die maximal von beiden unterstützte Protokollversion ausgehandelt, um auch zu älteren ICQ Clients Kompatibilität zu gewährleisten.
Über diese Verbindung lassen sich dann Nachrichten austauschen oder Dateien verschicken. Falls keine direkte Verbindung zustandekommt werden die Nachrichten über den Server vermittelt. Dateien zu versenden ist bei ICQ auf diesem Wege nicht möglich.

Um den Onlinestatus anderer ICQ Anwender einzusehen ist eigentlich eine einmalige „User Authentication“ durch den betreffenden Nutzer nötig, allerdings wird diese Einschränkung wohl nur in wohlwollenden IM Client Implementierungen durchgesetzt und nicht auf Protokollebene, denn es existieren einige Clients die keine Einwilligung der betroffenen Nutzer erzwingen.

### 2.1.3 Jabber

Jabber [Jabb] ist ein Satz offener Protokolle, die zum Austausch von Nachrichten und anderen strukturierten Informationen zwischen je zwei Kommunikationspartnern verwendet werden können. Die erste Anwendung von Jabber stellt ein eigener Instant Messaging Dienst dar.
Jabber hat seinen Ursrpung in der Open-Source Community wurde aber dann von der XMPP Working Group der IETF aufgegriffen und als Extensible Messaging and Presence Protocol formalisiert [XMPP].

Das Extensible Messaging and Presence Protocol (XMPP) dient als Grundlage für Instant Messaging mit Jabber. XMPP definiert XML Datenströme in denen über kurze XML-konforme Abschnitte, sogenannte XML Stanzas, Nachrichten ausgetauscht werden können.

Um sich im Jabber-Netz einzubuchen, verbindet sich ein Jabber IM Client über TCP mit einem Jabber Server und es wird in beide Richtungen ein XMPP Datenstrom initialisiert über den dann alle folgenden XML Stanzas verschickt werden. Teil der Loginprozedur auf dem Server ist eine Authentifizierung via SASL (Simple Authentication and Security Layer) und das Binden einer Ressource die die Verbindungssitzung kennzeichnet. Jeder IM Client lässt sich eindeutig über seinen JID (Jabber Identifier) der Form `user@domain[/resource]` identifizieren, wobei `domain` den Server repräsentiert und `resource` eine optionale Ressourcenangabe, die nur für aktuell verbundene Clients gilt darstellt.

Nachrichten und Informationen über den Onlinestatus werden beide als XML Stanzas kodiert, dem Server mitgeteilt. Den Onlinestatus eines Jabber Nutzers lesen, dürfen nur Nutzer die eine „Subscription“ dafür erworben haben, die nur der Anwender

selbst erteilen kann. Diese Subscripiton kann jederzeit widerrufen werden. Die Verantwortung den Onlinestatus nur berechtigten Nutzern weiterzuleiten obliegt dem Server des betroffenen Anwenders.

Im Gegensatz zu ICQ werden bei Jabber keine direkten Verbindungen zwischen den einzelnen Clients aufgebaut. Jabber Server verhalten sich mehr wie Mail Server, die untereinander kommunizieren und Nachrichten an den jeweils für den Addressaten zuständigen Server weiterleiten. Sind Absender und Empfänger am selben Jabber Server angemeldet kann dieser die Nachricht einfach weitergeben, falls nicht muss er dazu erst eine Verbindung mit dem Jabber Server des Empfängers herstellen.
Dabei lassen sich Jabber Server so konfigurieren, dass sie nur mit bestimmten oder gar keinen anderen Servern Verbindungen aufnehmen. Dies ermöglicht den Aufbau autonomer IM Netze..

## 2.2 Was ist Peer-to-Peer?

Seit dem rasanten Aufstieg und Fall von Napster [Mori] ist der Begriff Peer-to-Peer (P2P) auch außerhalb der technischen Fachwelt weit bekannt. Doch was ist Peer-to-Peer eigentlich?
Dieser Fragestellung soll dieser Abschnitt nachgehen. Dazu wird zunächst ein Überblick über die grundlegenden Begriffe gegeben, typische Verwendungszwecke werden diskutiert um schließlich auf die technischen Details verbreiteter Peer-to-Peer Anwendungen einzugehen.

Clay Shirky gibt folgende Definition für Peer-to-Peer [Shir00]:

> P2P is a class of applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.
> That's it. That's what makes P2P distinctive.

Peer-to-Peer Anwendungen sind also Netzwerkanwendungen die Ressourcen am Rande des Internets nutzen. Sie verbinden typischerweise Endsysteme miteinander deren Nutzer meist über keine feste IP Adresse verfügen und keine ständige Verbindung ins Internet haben, daher die Notwendigkeit auf DNS verzichten zu müssen und ein eigenes Adressierungsschema zu entwerfen. Trotz dieser Hindernisse soll es den Anwendern ermöglicht werden ihre Ressourcen, sei es Speicherplatz, Rechenzeit oder Dateiinhalte, zu teilen.

In Shirkys Definition finden sich interessanterweise auch Instant Messaging Anwendungen als Peer-to-Peer Anwendungen wieder.
Striktere Definitionen lassen keine solche sogenannten hyriden Systeme zu. Als hybride Peer-to-Peer Netze werden Systeme bezeichnet, die nicht völlig ohne zentrale Server auskommen aber deren Clients dennoch gewisse Peer-to-Peer Kommunikationsformen nutzen. Durch eine solche verschärfte Einschränkung wird auch Systemen wie Napster, die als Wegbereiter von Peer-to-Peer gesehen werden können, ihr Status

als Peer-to-Peer Anwendung entzogen. Daher wird diese Form der Definition häufig umgangen.

Oftmals wird bei der Definition von Peer-to-Peer Anwendungen gefordert, dass sie nicht nur Client- sondern auch Serverfunktionalität übernehmen. Dies wird häufig auch historisch begründet, als Abkehr vom durch den WWW-Boom induzierten Client/Server Paradigma, zurück zu einer ursprünglicheren Idee des Internets in der alle Rechner sowohl als Client agieren wie auch Serverdienste bieten.
Im folgenden soll als Anspruch an Anwendungen zumindest eine gewisse Autonomie des Handelns, die über reine Client-Fähigkeiten hinausgeht genügen um sie als Peer-to-Peer Anwendungen zu klassifizieren.

In neueren Peer-to-Peer Systemen ist häufig auch von Overlay Netzen die Rede. Overlays stehen wieder für eine ganz eigene Klasse von Systemen, die nicht notwendigerweise Peer-to-Peer Systeme sein müssen. Die Idee hinter Overlay-Netzen ist, eine von der physikalische Topologie der zugrunde liegenden (meist IP-basierten) Netze unabhängige logische Topologie zu definieren. Diese Overlaystruktur ist nicht fest vorgegeben, sondern kann sich dynamisch an Anwendungen und Lastituationen anpassen. Dadurch lassen sich beispielsweise optimierte Routingverfahren innerhalb der Overlaystruktur implementieren.

Die am weitesten verbreitete Anwendung von Peer-to-Peer Netzen ist das sogenannte File-Sharing. In File-Sharing Anwendungen, die auch unter dem Begriff Tauschbörsen bekannt sind, tauschen die Nutzer untereinander Dateien aus. Da es dabei häufig zu Urheberrechtsverletzungen kommt, wenn Anwender z. B. MP3 Dateien ohne Einwilligung der Künstler und deren Rechteverwerter zum Download anbieten, sorgen Peer-to-Peer Netze im Zusammenhang mit File-Sharing für regelmäßige negative Schlagzeilen.
File-Sharing ist, wenn auch die zur Zeit verbreitetste, bei weitem nicht die einzig denkbare Anwendung von Peer-to-Peer Systemen. Das Spektrum reicht ausschnittsweise erwähnt von Groupware und Instant Messaging Anwendungen über Verteiltes Rechnen bis hin zu Application Level Multicast Systemen.

Nach diesem allgemeinen Überblick über Peer-to-Peer Technologie, sollen nun einige verbreitete Peer-to-Peer Netze näher betrachtet werden.
Im folgenden werden die Begriffe Peer, Client, Knoten und Anwender häufig austauschbar benutzt, um den einen oder anderen Aspekt zu betonen. Sie beziehen sich jedoch meist auf denselben Teil eines Peer-to-Peer Netzes, nämlich auf die auf dem Endsystem laufende Applikation.

### 2.2.1 eDonkey2000

Als Vertreter des klassischen Filesharing mit Index-Server soll hier eDonkey2000 [eDon] betrachtet werden. Auch wenn Napster der eigentliche Urahn dieser Gattung ist, fiel die Wahl hier aus Gründen der Aktualität auf das eDonkey2000 Netz. Allein die Downloadstatistiken des größten Open Source Projekt Hosters SourceForge [SoFo] sprechen Bände für die Verbreitung dieses Netzes: Seit Jahren unangefochtene Nummer eins ist dort der alternative eDonkey2000 Client eMule [eMul].

Die ursrpüngliche Implementierung von eDonkey2000 war nicht quelloffen, allerdings entstanden durch Reverse Engineering recht schnell offene Implementierungen, die das Protokoll bekannt werden ließen [Klim03].

eDonkey2000 arbeitet, wie bereits angeklungen, nach dem hybriden Peer-to-Peer Muster mit Index-Servern, die für die Suche nach Dateien und Quellen die Vermittlerrolle zwischen den Peers übernehmen.

Jeder Peer muss sich beim Start der eDonkey2000 Anwendung zunächst bei einem eDonkey-Server anmelden. Dabei teilt er dem Server mit, welche Dateien er selbst freigeben möchte und erhält von seinem Server eine aktuelle Serverliste mit derzeit aktiven Servern.
Jeder Peer kann recht komplexe Suchanfragen an seinen oder auch an andere aktive Server schicken, in denen er das gewünschte Resultat neben dem Dateinamen auch nach Dateigröße oder Metadaten wie Bitrate bei Audio- oder Videodateien einschränken kann. Der Server schickt dann nach einem Vergleich mit den ihm bekannten Dateien passende Ergebnisse zurück.
Ebenso kann jeder Server nach zusätzlichen Downloadquellen für eine Datei befragt werden. Zu diesem Zweck werden im eDonkey2000 Netz alle Dateien eindeutig durch ihren MD4 Hashwert [Rive92] identifiziert.

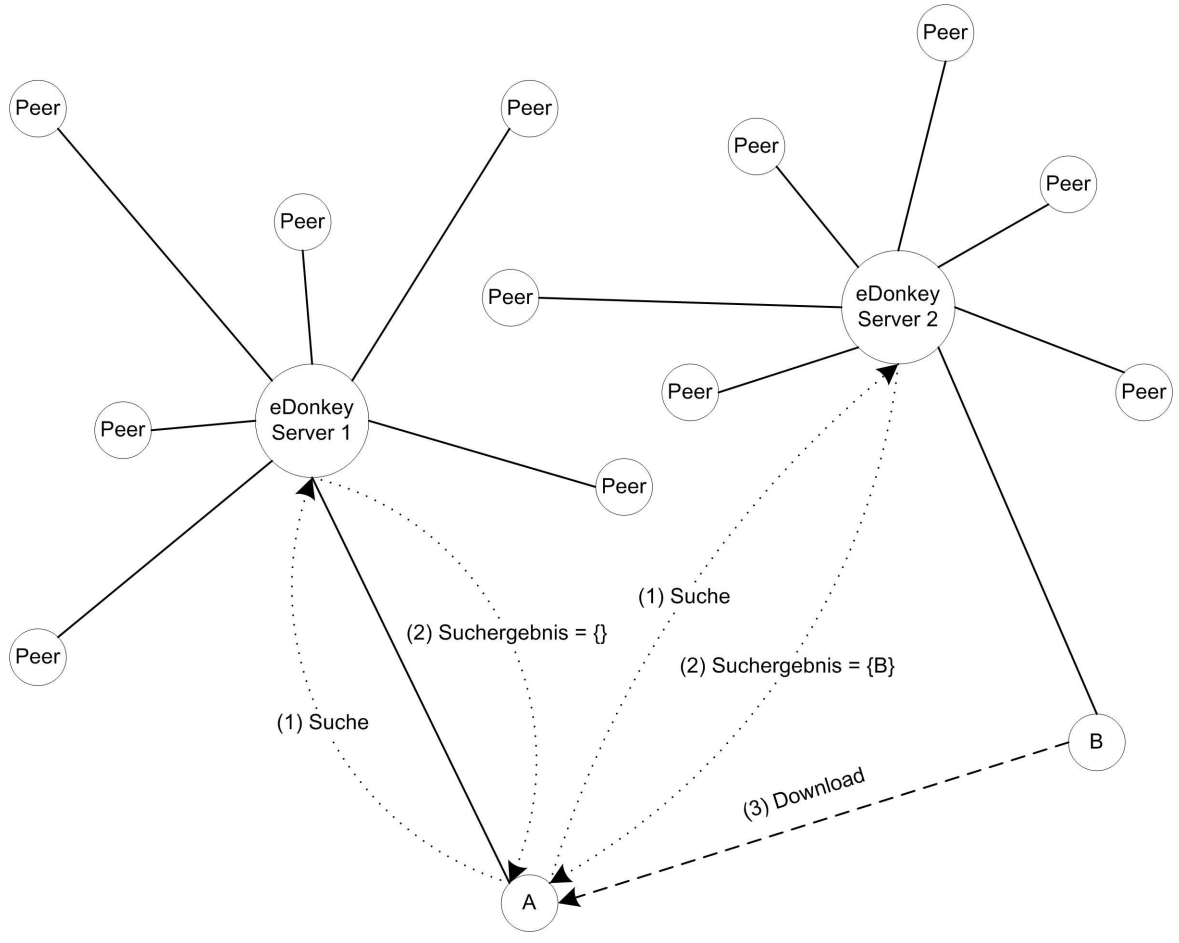


Abbildung 2.2: Suche im eDonkey2000 Netz: Peer A fragt den Server 1 mit dem er verbunden ist und einen anderen Server 2 nach einer bestimmten Datei, bekommt vom Server 2 eine potentielle Quelle, nämlich Peer B und initiiert den Download von diesem.

Der Dateitransfer selbst läuft dann zwischen den Peers ohne Zutun der Index-Server ab. Derjenige Peer der eine Datei herunterladen möchte, initiiert eine Verbindung zu einer möglichen Quelle und bekommt daraufhin von dieser mitgeteilt, welche Teile

der Datei der Quellen-Peer besitzt. Im eDonkey2000 Netz werden große Dateien in kleinere Abschnitte von etwas über 9 MB unterteilt, um es Peers zu ermöglichen auch Dateien weiterzuverbreiten, die sie selbst noch nicht komplett besitzen. Falls der Quellen-Peer noch Upload Kapazitäten frei hat, kann der initiierende Peer dann mit dem Download eines benötigten Dateiabschnittes beginnen, andernfalls landet er in einer Warteschlange.
Außerdem ist es möglich, von mehreren Quellen gleichzeitig Teile der selben Datei zu beziehen, was das eDonkey2000 Netz vor allem für den Tausch großer Dateien prädestiniert. Denn durch den parallelen Download einer Datei von mehreren Quellen addieren sich die einzelnen Upload-Bandbreiten der Peers.

#### eMule Erweiterungen

Seit MetaMachine die Entwicklung von eDonkey2000 zugunsten des designierten Nachfolgers Overnet (siehe Abschnitt 2.2.3.2) eingestellt hat, wird die Fortentwicklung des eDonkey Netzes hauptsächlich von den Entwicklern der derzeit am weitesten verbreiteten eDonkey2000 Anwendug namens eMule [eMul] angetrieben. So haben sich aus dieser Richtung einige sinnvolle Erweiterungen durchgesetzt, die hier kurz skizziert werden sollen.

Die Komprimierung des Dateitransferstroms zwischen Peers beschleunigt vor allem gut komprimierbare Downloads wie z. B. Textdateien.
Außerdem wurde ein Credit System eingeführt, das dafür sorgen soll im lokalen Warteschlangenmanagement Peers zu bevorzugen, die sich selbst durch Uploads als wertvoll erwiesen haben. Damit soll verhindert werden, dass Anwender das Netz durch Downloads belasten ohne selbst mit Upload-Bandbreite beizutragen.
Eine der wichtigsten Neuerungen ist aber der sogenannte Peer-Quellentausch, denn damit ist es Peers möglich untereinander potentielle Downloadquellen auszutauschen, ohne die Index-Server mit entsprechenden Anfragen zu belasten.

Die neueste und derzeit noch in der Testphase befindliche Idee der eMule Entwickler zielt darauf ab, die Index-Server auch noch weiter von Suchen zu entlasten, indem ein Kademlia-basiertes Netzwerk (siehe Abschnitt 2.2.3.1) in den eMule Client integriert werden soll. Damit könnte das ehemalige eDonkey2000 Netz vollständig von seiner ursprünglichen Index-Server zentrierten Architektur gelöst werden.

### 2.2.2 Gnutella

Gnutella ist ein Peer-to-Peer Filesharing Protokoll, das ohne jegliche zentrale Infrastruktur wie Index-Server auskommt, sondern diese Aufgabe auf die Peers im Netz verteilt.
Es existieren zahlreiche Anwendungen die Gnutella implementieren, darunter LimeWire [Lime] für Java-fähige Systeme und Gnucleus [Gnuc] für die Win32 Plattform. Im folgenden soll das Gnutella Protokoll Version 0.4 [GnP4] genauer betrachtet werden.

Da Peers im Gnutella Netz sowohl Client- wie auch Server-Funktionalitäten übernehmen werden sie auch als Servents bezeichnet.

Das Gnutella Protokoll definiert die folgenden Nachrichtentypen:

**Ping:** Dient zur aktiven Erkundung der Umgebung. Der Empfänger der Ping Nachricht sollte mit einem Pong antworten und das Ping ins Netz weiterleiten.

**Pong:** Die Antwort auf ein Ping enthält die Adresse des Pong-Absenders und Informationen darüber wieviele Dateien der Servent frei gibt.

**Query:** Suchanfragen enthalten einen Suchstring. Falls es eine Übereinstimmung der Suchanfrage mit den lokalen Daten des Empfängers gibt, sollte dieser mit einem QueryHit antworten.

**QueryHit:** Die Antwort auf ein Query enthält eine Liste aller Dateien auf die das Suchkriterium zutrifft.

**Push:** Falls der Servent, welcher die gesuchte Datei anbietet mit einer privaten IP hinter einer NAT (Network Address Translation) steht oder seine Firewall keine eingehenden Verbindungen zulässt wird das in Gnutella „firewalled“ genannt. Der anbietende Servent kann dann durch ein Push den suchenden Servent dazu veranlassen selbst eine eingehende Verbindung zu zulassen, falls dieser nicht ebenfalls derart abgeschirmt ist.

Ein Peer der dem Gnutella Netz beitreten will, schickt ein Ping an einen aktiven Servent. Informationen über aktive Servents beziehen die meisten Gnutella Implementierungen aus sogenannten Web-Caches [GnWC], im Gnutella Protokoll selbst ist keine Vorgehensweise dafür festgelegt.
Der bereits im Gnutella Netz verbundene Servent flutet dieses Ping ins Netz weiter und leitet zurückkommende Pong Antworten an den neuen Peer zurück. Aus diesen zurückkommenden Pongs wählt der neue Peer dann einige aus, stellt mit diesen TCP-Verbindungen her und wird dadurch ein Teil des Gnutella Netzes.

Sämtliche Ping- und Query-Nachrichten die ein Servent erhält leitet dieser weiter an alle Servents mit denen er eine Verbindung aufgebaut hat, ausgenommen natürlich denjenigen von dem die Nachricht kommt.
Die Pong-, QueryHit- und Push-Nachrichten sind Antworten auf vorangegangene Ping- oder Query-Nachrichten und werden auf demselben Weg zurückgeroutet den diese auf dem Hinweg genommen haben.
Damit Nachrichten nicht ewig im Netz zirkulieren, haben alle Nachrichten ein TTL-Feld (Time To Live), das bei jeder Weiterleitung um eins dekrementiert wird. Erreicht das TTL-Feld Null wird das Paket nicht mehr weitergeleitet.
Um das Zirkulieren der Nachrichten in Schleifen zu verhindern, merken sich die Servents welche Nachrichten sie schon weitergeleitet haben anhand einer eindeutigen Deskriptor ID, die jede Nachricht tragen muss.

In Abbildung 2.3 ist gezeigt, wie eine Dateisuche und der anschließende Download typischerweise abläuft.
Servent A schickt ein Query an die mit ihm verbundenen Servents B und C (1a), die diese wiederum an die mit ihnen verbundenen Servents weiterleiten (1b). Zur Vereinfachung wurde hier eine TTL von 2 angenommen. Servent F erkennt, dass er eine Datei besitzt, die mit dem Suchkriterium übereinstimmt und schickt ein QueryHit zurück (2a, 2b).
Der eigentliche Dateitransfers findet jetzt nicht im Gnutella Netz statt, sondern wird über HTTP Aufrufe zwischen den beiden Servents A und F realisiert (3).

Seit den ersten Implementierungen wurde Gnutella, begünstigt durch die Verfügbarkeit offener Quellen die durch Reverse Engineering der ursrpünglichen Nullsoft

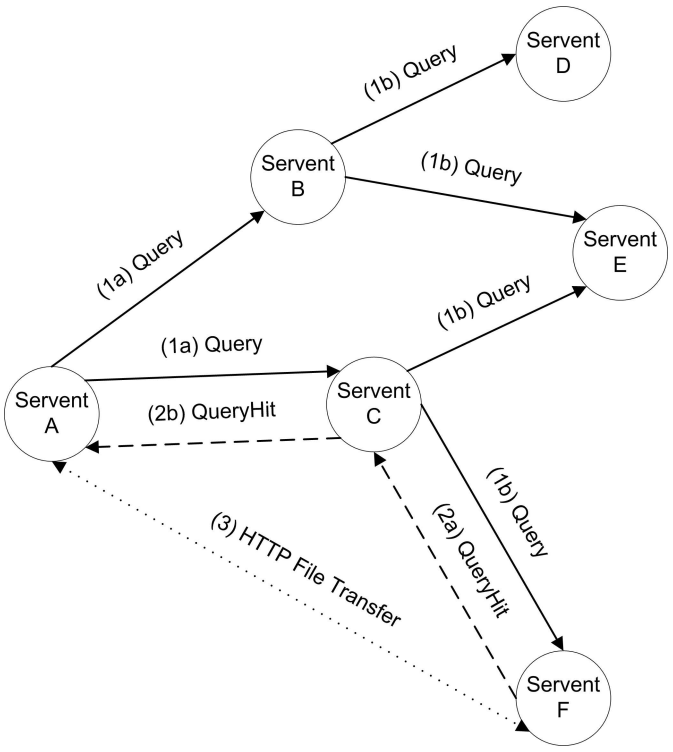


Abbildung 2.3: Suche im Gnutella Netz: Servent A sucht nach Datei die Servent F besitzt und lädt diese herunter.

Implementierung entstanden [Ivko01], stetig weiterentwickelt und um Fähigkeiten ergänzt.
So bestand laut [Ripe01] der Verkehr im Gnutella Netz im November 2000 noch zu nur 36% der Nachrichten aus Nutzverkehr (QUERY-Nachrichten). Bereits im Juni 2001 ergaben die Messungen hingegen 92%.

### Das Ultrapeer Konzept

Die zufällige Auswahl von Verbindungen zwischen Gnutella Servents hat große Nachteile für Nutzer mit geringer Bandbreite die etwa über ein Modem Anschluss ins Internet finden. Die Leitung solcher Nutzer ist schon allein mit dem von den breitbandig angeschlossenen Servents generierten Verkehr ausgelastet. Für Dateitransfers bleibt nahezu keine Kapazität frei.
Dieses Problem addressiert Gnutella Protokoll Version 0.6 [GnP6] mit dem Ultrapeer Konzept.

Durch die Kategorisierung von Servants in Leaf und Ultrapeer Systeme, wird versucht dem ungeordneten Netzwerk ein gewisse Hierarchie überzustülpen.
Ein Leaf Servant unterhält nur wenige Verbindungen und zwar lediglich zu Ultrapeer Servants die quasi als Proxy für die Leaf Servants agieren. Die Ultrapeer Servants haben neben den Verbindungen zu ihren Leaf Servants auch Verbindungen zu anderen Ultrapeers und zu Servants die das Ultrapeer Konzept nicht unterstützen. Dadurch soll gewährleistet werden das das Gnutella Netz nicht in einzelne Teile zerfällt.
Ein Ultrapeer leitet nun nicht mehr alle Query Nachrichten die aus dem Gnutella Netz an ihn dringen an seine Leaf Servants weiter, sondern entscheidet anhand des Query Routing Protocols (QRP) ob diese eine Antwort kennen könnten. QRP funktioniert mittels Hashtabellen, die beim Verbindungsaufbau vom Leaf auf den

Ultrapeer übertragen werden und die Hashwerte aller auf dem Leaf Servant freigegebener Dateinamen enthalten. Dadurch kann der Ultrapeer für eingehende Query Nachrichten via QRP feststellen welche Leaf Servants zumindest potentiell auf die Anfrage reagieren können und filtert dadurch viele (wenn auch nicht alle) für die Leaf Servants unnötige Anfragen heraus.

Ein neues Problem das durch die Einführung von Ultrapeers entsteht ist die Frage welche Servants denn nun Ultrapeers werden sollen und welche nicht.
Zunächst lässt sich durch eine gewisse Mindestanforderung an Bandbreite, Speicher und Rechenleistung eine gewisse Vorauswahl treffen, welche Servants gut geeignet wären („Ultrapeer-fähig“) die zusätzliche Belastung durch eine Stellung als Ultrapeer zu übernehmen.
Aber ohne zentrale Instanz und somit ohne eine Gesamtübersicht über das Gnutella Netz lässt sich die aktuelle Belastung nur aus dem beobachtbaren Verhalten abschätzen. Daher teilen die Ultrapeers beim Verbindungsaufbau mit anderen Servants mit ob sie aufgrund ihrer aktuellen Belastung mehr Ultrapeers brauchen, was einen Ultrapeer-fähigen Leaf Servant dazu veranlasst zu einem Ultrapeer zu werden oder ob sie eher zu wenige Verbindungen haben, was im Falle eines Verbindungsaufbaus zwischen zwei Ultrapeers dazu führt das einer von beiden zu einem Leaf wird und seine nicht-Ultrapeer Verbindungen schließt.

### 2.2.3 Distributed Hashtables

Ähnlich wie Gnutella verfolgen auch Distributed Hashtables (DHTs, auf deutsch: Verteilte Hash-Tabellen) einen vollkommen dezentralen Ansatz. Unterschiede zeigen sich allerdings wenn man die Verteilung der Daten und die Positionierung der einzelnen Knoten im Overlay-Netz betrachtet. Während das bei Gnutella abhängig vom Nutzerverhalten, also nicht vorhersagbar geschieht, wird in DHTs versucht durch die Vorgabe einer Struktur effizientere Routingverfahren zu ermöglichen [RaSS02].
Es existieren einige Varianten für die Gestalt dieser Struktur: Exemplarisch erwähnt werden sollen hier Chord [SMKK+01], in dem Knoten auf einem Ring angeordnet werden und CAN (Content-Adressable Network) [RFHK+01] bei dem die einzelnen Knoten auf einem d-dimensionalen Torus plaziert werden.

Trotz unterschiedlicher Topologien haben alle DHTs große Gemeinsamkeiten:
So wird jeder Knoten durch eine eindeutige Knoten ID[1] gekennzeichnet. Diese Knoten ID dient nicht nur zur Identifizierung, sondern lässt sich auch als Position in der virtuellen Topologie des Overlay-Netzes interpretieren.
Jeder Knoten im Netz ist nun für einen bestimmten Bereich an Schlüsseln verantwortlich, deren Werte in einer passend zur Topologie definierten Metrik einen geringen Abstand zur Knoten-ID haben. Ausserdem lässt sich aus dieser Metrik eine Nachbarschaftsbeziehung zwischen Knoten ableiten.

Die beiden grundlegenden Operationen auf einem DHT sind:

**Put(Schlüssel, Wert):** Legt einen Wert unter einem bestimmten Schlüssel ab. Der Schlüssel wird ebenso wie die Knoten-IDs auf die virtuelle Topologie des Overlay Netzes abgebildet und soll durch den ihm am nächsten liegenden Knoten gespeichert werden.

[1] üblicherweise finden 128 bis 160 Bit große Zahlen Verwendung

Um diesen Knoten zu ermitteln routet der initiale Knoten, der den Wert ablegen möchte, die Anfrage zu einem ihm bekannten Knoten dessen Knoten ID dem Schlüssel am nähesten ist. Dieser routet die Anfrage weiter an den wiederum ihm als näher am Schlüssel bekannten Knoten, bis die Anfrage beim für den Schlüsselbereich Verantwortlichen gelandet ist und dort der Wert gespeichert werden kann.

**Get(Schlüssel):** Holt den Wert der unter dem angegebenen Schlüssel abgelegt wurde.
Analog zum Ablegen des Werts, routet der initiale Knoten die Anfrage weiter an einen Knoten der näher am Schlüssel liegt und dieser wiederholt das bis auch hier der verantwortliche Knoten gefunden ist und dieser den Wert zurückgibt.

Auf Grundlage von DHTs lassen sich auch File-Sharing Netze betreiben, indem man aus den Dateien mit einer Hash-Funktion eindeutige Schlüssel generiert und im Netz unter diesen Schlüsseln Rechner publiziert die diese Dateien freigeben. Wie das im Einzelnen geschieht zeigt Abschnitt 2.2.3.2, doch zuerst soll ein etwas genauerer Blick auf einen konkreten DHT Algorithmus geworfen werden.

#### 2.2.3.1 Kademlia

In [MaMa02] schlagen P. Maymounkov und D. Mazieres ein Konzept zur Realisierung von Distributed Hashtables namens Kademlia vor.

In Kademlia finden Schlüssel mit 160 Bit Länge Verwendung. Alle am Kademlia Netz teilnehmenden Rechner besitzen eine ebenfalls 160 Bit breite quasi-eindeutige Knoten ID. Werte werden auf möglichst nahe an ihrem zugehörigen Schlüssel liegenden Knoten abgespeichert.

Um die Entfernung zwischen Knoten und Schlüsseln zu bestimmen, definiert Kademlia XOR als Metrik: Die Entfernung zwischen zwei IDs x und y beträgt also $d(x, y) = x \otimes y$, wobei $\otimes$ für die bitweise XOR Verknüpfung steht.

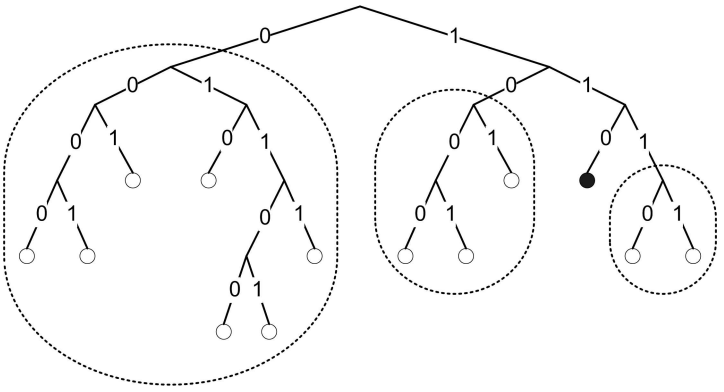

Abbildung 2.4: Baumdarstellung der Knotem im Kademlia Netz. Die Unterbäume für den schwarz eingefärbten Knoten mit kürzestem eindeutigen Präfix 110 sind hier eingezeichnet.

Kademlia betrachtet alle Knoten als Blätter eines Binärbaums. Die Position der Blätter bestimmt sich aus deren kürzestem eindeutigen Präfix in der Binärdarstellung der Knoten ID.

Für jeden einzelnen Knoten lässt sich dieser Binärbaum nun in immer kleiner werdende Unterbäume unterteilen die den Knoten selbst nicht beinhalten (siehe Abbildung 2.4).

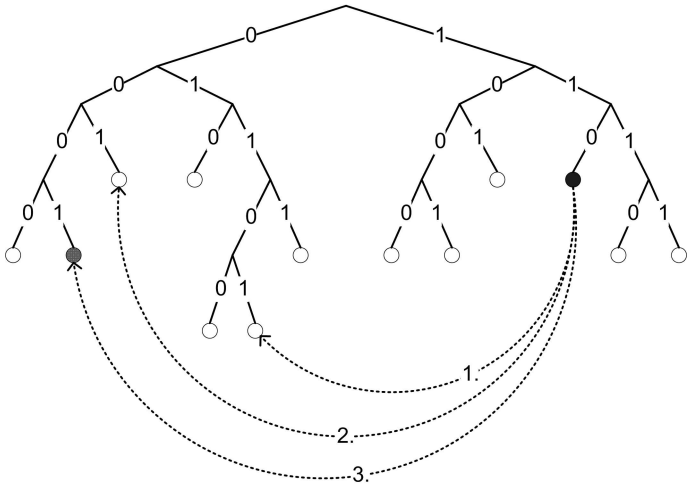


Abbildung 2.5: Rekursive Knotensuche: Der Knoten mit kürzestem eindeutigen Präfix 110 sucht nach dem Knoten mit kürzestem eindeutigen Präfix 0001.

Angenommen jeder Knoten wüsste von mindestens einem weiteren Knoten in jedem seiner Unterbäume, dann könnte er jeden beliebigen Knoten erreichen, indem er sich rekursiv bei Knoten, die mit jedem Schritt näher am gesuchten Knoten stehen, durchfragt:
In Abbildung 2.5 wird angenommen der schwarz eingefärbte Knoten mit dem kürzesten eindeutigen Präfix 110 suche nach dem Knoten mit Präfix 0001. Anhand der Such-ID 0001 weiss der Knoten, dass der gesuchte in seinem 0-Unterbaum liegen muss und fragt dort einen ihm bekannten Knoten, nämlich den mit Präfix 01101. Dieser weiss nun über die Such-ID, dass er einen Kontakt aus seinem 00-Unterbaum zurückgeben muss (im Beispiel kennt er dort den Knoten mit Präfix 001). Dieser liefert wiederum auf Anfrage einen Knoten aus seinem 000-Unterbaum, der hier im Beispiel dann schon der Gesuchte mit Präfix 0001 ist.

Man sieht hier sehr schön, dass durch eine solche Konstruktion die Knoten mit abnehmender Distanz immer mehr ihrer Nachbarn kennen.

Um von einem Knoten jeden anderen zu erreichen, genügt es also einen Kontakt in jedem möglichen der theroetisch maximal 160 Unterbäume zu kennen.
Dem Umstand Rechnung tragend, dass Knoten hin und wieder das Netzwerk verlassen, werden statt nur einem gleich mehrere Kontakte pro Unterbaum in Listen gespeichert. Diese Listen heißen k-Buckets und halten jeweils k Einträge[2] von (IP Adresse, UDP Port, Knoten ID)-Tripeln.
Aus der Verantwortung jedes k-Buckets für einen bestimmten Unterbaum und der XOR-Metrik folgt, dass sich im i-ten k-Bucket nur Knoten mit Abstand $2^i$ bis $2^{i+1}$ von der eignenen ID befinden. Die untersten k-Buckets (für die kleinsten Unterbäume) sind mit großer Wahrscheinlichkeit leer, da sich nur wenige Knoten erst in den hintersten Stellen der ID unterscheiden werden.

[2] k sollte so gewählt werden, dass innerhalb einer Stunde (dem Kontakt Update Intervall von Kademlia) nur äußerst unwahrscheinlich k zufällig gewählte Rechner vom Netz getrennt werden. Die Kademlia Autoren empfehlen k=20.

Wenn ein Kademlia Knoten eine Nachricht erhält, fügt er die Kontaktinformationen des anderen Knotens in den passenden k-Bucket ein. Ist der k-Bucket bereits voll, wird ein PING an den ältesten[3] Eintrag im k-Bucket geschickt und dieser nur ersetzt falls eine Antwort darauf ausbleibt. Dadurch soll die Angreifbarkeit des Netzes durch massenhaftes Fluten neuer Knoten ins Netz verringert werden, denn bereits bekannte Knoten werden damit nie aus den k-Buckets entfernt solange sie noch antworten. Dabei spielt es keine Rolle ob die Nachricht eine fremd initiierte Anfrage oder eine Antwort auf eine selbst gestartete Anfrage war. Hier zeigt sich eine Stärke von XOR als Metrik: Durch die Symmetrie von XOR spart Kademlia Kommunikationsaufwand, indem Knoten auch durch eingehende Anfragen nützliche Informationen über ihre Umgebung lernen.

Kademlia sieht vier Nachrichtentypen vor:

**PING:** Testet ob ein Knoten Online ist.

**STORE:** Speichert (Schlüssel, Wert)-Paar auf Knoten.

**FIND_NODE:** Als Parameter wird eine ID übergeben. Der Empfänger gibt die ihm bekannten nähesten k Kontakt-Tripel (IP Adresse, UDP Port, Knoten ID) zu dieser ID zurück. Diese Tripel müssen nicht alle aus dem selben k-Bucket stammen wenn der näheste nicht voll ist.

**FIND_VALUE:** Als Parameter wird eine ID übergeben. Wurde der Empfänger bereits per STORE angewiesen unter der ID einen Wert zu speichern, dann gibt er nur den entsprechenden Wert zurück. Ansonsten verhält sich der Empfänger wie bei FIND_NODE.

Mit Hilfe rekursiver Aufrufe von FIND_NODE lassen sich die k nächsten Knoten zu einer gegebenen ID finden und dort per STORE speichern. Die durch einen Schlüssel identifizierte Werte können über rekursive Aufrufe von FIND_VALUE wiedergefunden werden.
Um die rekursive Knotensuche zu beschleunigen, lassen sich Anfragen an ähnlich weit entfernte Knoten asynchron bearbeiten, indem parallel eigentlich redundante Anfragen an verschiedene Knoten abgeschickt werden. Ausserdem lassen sich Suchschlüsselanfragen auch auf weiter entfernten Knoten cachen: Dazu speichert der suchende Knoten die Ergebnisse nach erfolgreichem Lookup auf dem letzten Knoten auf dem Knotensuchpfad, welcher auf die Suchanfrage nicht mit dem Ergebnis antwortete.

Um die Persistenz abgespeicherter Werte zu sichern, werden (Schlüssel, Wert)-Paare nach einer Stunde von den für sie per STORE verantwortlich gemachten Knoten erneut publiziert. Nach 24 Stunden läuft die Gültigkeit der Werte ab und der Knoten, der die Werte ursprünglich ins Netz stellte muss sie selbst aktualisieren.

Ein bisher noch nicht verbundener Knoten kann dem Kademlia Netzwerk beitreten, wenn er mindestens einen verbunden Knoten kennt. Dazu trägt er den bekannten Knoten in den passenden k-Bucket ein und führt dann eine rekursive Knotensuche nach seiner eigenen ID durch. Dadurch füllt er nicht nur seine eigenen k-Buckets sondern wird auch bei den Knoten auf dem Weg seiner Suche bekannt.

[3] Die k-Buckets sind nach dem Zeitpunkt der letzten Nachricht sortiert.

Anschließend führt er einen Bucket Refresh auf allen Buckets aus die weiter als sein nähester Nachbar entfernt sind: Für jeden Bucket wird nach einer zufällig aus dem Bucket-Bereich gewählten ID eine Knotensuche durchgeführt.

Kademlia findet in einigen verbreiteten Peer-to-Peer Anwendungen Verwendung, darunter Overnet [Over], das im nächsten Abschnitt näher beschrieben wird, sowie in den neuen eMule Clients ab Version 0.40 [eMul].

#### 2.2.3.2 Overnet

Overnet wurde von MetaMachine als Nachfolger zum eDonkey2000 Netz konzipiert [Over]. Im Gegensatz zu diesem benötigt es keine zentralen Index-Server zur Suche sondern arbeitet völlig dezentral. Nach [BhSV03] basiert Overnet auf dem Kademlia Protokoll.
Da MetaMachine mit den Overnet Client durch Werbeeinblendungen in der grafischen Oberfläche und den Verkauf von werbefreien Pro Versionen Geld verdienen will, gibt es den Overnet Client nur in kompilierter Form und nicht im Sourcecode. Trotzdem existieren durch Reverse Engineering entstandene Open Source Clients wie z. B. MLdonkey [MLdo] die Overnet-fähig sind.
Damit und mit dem Wissen um die Kademlia Grundlagen und einem Protokoll-Analyse Tool wie Ethereal [Ethe] lässt sich die Arbeitsweise des Overnet Client recht gut nachvollziehen.
Die folgende Erkenntnisse wurden mit dem Overnet Command Line Client Version 0.51.2 für Linux und Ethereal Version 0.9.14 mit angewandtem EDONKEY Dissektor gewonnen.

Das eigentliche Transferprotokoll zum Dateientausch wurde bei Overnet vom Vorgänger eDonkey2000 übernommen. Neu ist die verteilte Quellensuche nach dem Kademlia Prinzip.
Overnet benutzt zur Identifikation der einzelnen Clients 16-Byte (128 Bit) breite Identifikatoren.
Als Hash-Algorithmus findet MD4 [Rive92] Verwendung, welcher zur Erzeugung von ebenfalls 128 Bit großen Prüfsummen dient.

Um die Publikation einer neuen Datei im Overnet-Netz zu beobachten wurde eine Datei im Freigaben-Verzeichnis von Overnet erstellt und daraufhin der Netzwerkverkehr des Command Line Clients abgehört, nachdem dieser über den Konsolenbefehl „refresh“ angewiesen wurde sein Freigabenverzeichnis neu einzulesen.
Der Overnet Command Line Client beginnt daraufhin mit der rekursiven Suche nach dem verantwortlichen Knoten für die Speicherung des MD4 Hashwerts des Dateiinhalts: Er schickt parallel Anfragen an andere Overnet Peers die als Ergebnis weitere Peers liefern deren ID immer näher an dem MD4 Hash der Datei liegen. Hat er einige nahe am Hashwert liegende Knoten gefunden speichert der Overnet Command Line Client dort unter dem Dateihash als Schlüssel seine eigene Knoten ID als Wert. Dadurch können andere Peers die den Dateihash kennen ihn als Quelle für den Dateitransfer ausfindig machen.
Anschließend führt der Overnet Command Line Client eine Knotensuche nach dem MD4 Hash des Dateinamens durch, um unter den dafür verantworlichen Knoten den Dateihash abzulegen. Zusätzlich zum Datei-Hashwert werden als Meta-Daten der Dateiname selbst, die Dateigröße und die aktuelle Verfügbarkeit unter dem Dateinamen-Hash als Schlüssel abgelegt.

Die Publikation neuer Dateien im Netz lässt sich also durch zwei einzelne Publish Operationen beschreiben:
`Publish(Schlüssel := Hash(Dateiinhalt), Wert := Knoten ID)` und
`Publish(Schlüssel := Hash(Dateiname), Wert := Hash(Dateiinhalt))`

Eine genauere Betrachtung der Publikation bei komplexen Dateinamen ergibt, dass der Overnet Command Line Client die Dateinamen an bestimmten Stellen auftrennt (z. B. vor und nach „-“, „_“, „.“ und „ “) und für jeden Teil des Dateinamens einen extra Hashwert berechnet und publiziert. Das ermöglicht die Suche auch nach Teilen von Dateinamen, sofern der Ersteller der Datei bei der Namensgebung eine sinnvolle Trennregelung beachtet.

Bei der Untersuchung der rekursiven Knotensuche fällt auf, dass meist 4 Kontakte als Ergebnis geliefert werden und die Werte unter 4 Knoten deren ID nahe des Schlüssels liegen abgelegt werden. Dies lässt vermuten, dass der Kademlia-Paramter k bei der Overnet Implementierung den Wert 4 trägt.

Die Suche nach Dateien funktioniert dann entsprechend: Ist der Dateihash bekannt[4] kann mit einer Suche nach den darunter abgelegten Quellen ein oder mehrere Peers gefunden werden mit denen dann ein Transfer ausgehandelt werden kann.
Kennt der Anwender nur den Dateinamen muss zunächst eine Suche nach dem Hash des Dateinamens durchgeführt werden, deren Resultat dann der Dateihash ist.
Etwas raffinierter verhält sich die Suche nach mehreren Suchbegriffen: Dabei wird der Knoten der für das erste Suchwort zuständig ist befragt, bekommt aber zusätzlich die restlichen Suchworte als Such-Information mitgeteilt. Darauf schickt der befragte Knoten alle unter dem Hash des ersten Suchbegriffs abgelegten Werte zurück, deren Meta-Daten ebenfalls zu den restlichen Begriffen passen.
Da der EDONKEY Dissektor der verwendeten Ethereal Version das Paket mit den Zusatzinformationen zu den weiteren Suchbegriffen fehlinterpretiert fiel dieses Verhalten erst nach genauer Untersuchung der uninterpretierten Rohdaten auf, nachdem immer nur völlig korrekte Antworten die auf alle Suchbegriffe passen auf eine einfache Hash-Anfrage nach dem ersten Suchbegriff-Hash zurückkamen.
Durch dieses Verfahren wird im Vergleich zu dem ebenfalls denkbaren Vorgehen, jeden Suchbegriff zu hashen und danach zu suchen und die Schnittmengenbildung erst auf dem suchenden Peer durchzuführen, Bandbreite zu Lasten eingeschränkter Redundanz gespart.

### 2.2.4 Vergleich der Netze

Nachdem die technischen Grundlagen verbreiteter Peer-to-Peer Netze betrachtet wurden, soll nun deren Eignung für bestimmte Einsatzszenarien anhand der Kriterien Skalierbarkeit, Robustheit und Sucheffizienz erörtert werden.

#### 2.2.4.1 Skalierbarkeit

Die klassischen Filesharing Netze der ersten Generation wie Napster und eDonkey2000 setzen mindestens einen zentralen Index-Server voraus, dessen Bandbreite und Rechenleistung die Größe des Netzes stark limitiert. Natürlich lässt sich die

[4] Overnet erlaubt wie eDonkey das direkte Verlinken von Dateien im Netz mittels spezieller ed2k-URLs die den Dateihash beinhalten. Diese Links können z. B. im WWW veröffentlicht oder per eMail ausgetauscht werden.

unterstützte Netzgröße durch den Einsatz lestungsfähigerer und zusätzlicher Server bis zu einem gewissen Grad erhöhen. Allerdings sind dem Wachstum auf diesem Wege Grenzen gesetzt. Abhängig davon ob ein Peer nur einen authorativen Index-Server zugewiesen bekommt, oder prinzipiell bei allen anfragen kann, stellen sich dem Netzwachstum Hürden entgegen.
Im ersten Fall zerfällt das Netz in viele kleine Teilnetze mit ihrem jeweils eigenen Index-Server, zwischen denen keine Kommunikation stattfindet.
Im anderen Fall wird die Belastung des Netzes zwar auf mehrere Index-Server verteilt, die Obergrenze der maximal unterstützte Netzgröße wird damit aber nur etwas angehoben. Denn mit jedem weiteren Nutzer kommt nicht nur die Verwaltung der von ihm freigegebenen Daten hinzu sondern vor allem die von ihm erzeugte Belastung durch Suchen, die alle Server gleichermaßen belastet. Daher ist davon auszugehen das die maximale Netzgröße deutlich schlechter als linear mit der Serveranzahl skaliert.

Im Gnutella Netz werden keine zentralen Infrastrukturrechner vorausgesetzt. Das Netz besteht aus den teilnehmenden Peers selbst. Jedoch ist die Annahme, dass Gnutella daher nahezu unbegrenzt skaliert falsch. In [Ritt01] zeigt Jordan Ritter, dass der durch das Fluten von Suchen im Netz erzeugte enorme Verkehr zum Flaschenhals für das gesamte Netz wird.
Erschwerend kommt hinzu, dass nur schwach angebundene Servants, die z. B. über eine Modemverbindung angeschlossen sind, zusätzliche Engpässe für die Weiterleitung von Nachrichten im Netz darstellen, was allerdings durch die Einführung hierarchischer Strukturen mittels Ultrapeers gemildert werden kann.

Distributed Hashtables wurden von Beginn an mit dem Ziel guter Skalierbarkeit konzipiert. Dazu ist es unumgänglich auf jegliche zentrale Infrastruktur zu verzichten die einen Flaschenhals darstellen würde. Trotzdem konnten die Fehler von Gnutella durch eine effizientes Such-Routing vermieden werden (siehe Abschnitt 2.2.4.3).
Ein weiterer wichtiger Punkt der Beachtung finden muß um lokale Engpässe zu vermeiden, ist die Behandlung von Schlüsselkollissionen und besonders populären Anfragen: Ersteres Problem läßt sich durch die Wahl eines gut streuenden Hashalgorithmus einschränken, letzteres durch eine dynamische Replikation auf zusätzlichen Knoten zumindest abschwächen.
Nach [ZhJK02] lässt sich die Skalierbarkeit von DHTs weiter steigern, indem in der Routingtabelle naturgemäß auftretende Wahlmöglichkeiten nicht zufällig, sondern in Hinblick auf die dem Overlay zugrundeliegende Netzinfrastrukturparameter wie Latenz oder Hop-Entfernung aufgelöst werden („locality awareness“).
In Kademlia findet ein derartiges Schema implizit Verwendung, da Nachrichten über UDP parallel redundant abgesendet werden und dadurch Knoten mit niedriger Latenz und geringer Stauwahrscheinlichkeit auf lange Sicht bevorzugt werden.

#### 2.2.4.2 Robustheit

Was die Robustheit angeht schneiden die klassischen Index-Server basierten Architekturen erwartungsgemäß schlecht ab. Der Index-Server stellt einen Single Point Of Failure dar, so dass schon der Ausfall eines einzelnen Rechners, ob durch einen Defekt in der Hardware oder durch gezielte Angriffe, das Netz (oder zumindest grosse Teile des Netzes bei mehreren Index-Servern) seiner Suchfunktionalität und Quellensuchfunktion berauben und damit lahmlegen kann.

Da alle Servants im Gnutella Netz im Prinzip gleich sind, gibt es keine besonders exponierten Rechner die sich für einen Angriff eignen würden. Gleiches gilt für Ausfälle aufgrund von Defekten. Prinzipiell hat der Verlust einzelner Servants also keinen großen Einfluss auf das Netz und seine Leistungsfähigkeit.
Dies ändert sich durch die Einführung von Ultrapeers, da dadurch wieder lohnenswerte Ziele für Angriffe ausgemacht werden können. Zwar können sich Servants durch ein sogenanntes Fallback wieder auf das alte Gnutella Netz ohne Servants mit herausgehobener Stellung zurückbesinnen wenn sie keine Ultrapeers erreichen, allerdings geht das dann wiederum zu Lasten der Skalierbarkeit.

Ein weiterer unerwünschter Effekt tritt zu Tage, wenn man die tatsächliche Verteilung der Daten im Netzwerk betrachtet. Adar und Huberman haben in Messungen festgestellt [AdHu00], dass über 70% der Gnutella Nutzer keinerlei Dateien freigeben. Darüber hinaus kamen in 50% der Suchen die Antworten von 1% der Nutzer. Adar und Huberman nennen dieses Verhalten der Anwender die nur herunterladen ohne selbst Dateien zum Netzwerk beizusteuern „Free-Riding“. Abgesehen von den psychologischen Hintergründen entstehen dadurch auch handfeste technische Probleme. Es existieren somit besonders lohnenswert angreifbare Systeme deren Ausfall das System nur schwer kompensieren kann, und die eigentliche Peer-to-Peer Idee hinter Gnutella verkommt wieder zu einem Client/Server Modell nur mit ungleich höheren Lokalisationsaufwand.
Dieses Phänomen betrifft allerdings alle Filesharing Netze in denen Nutzer quasi anonym auf Kosten anderer Teilnehmer dem Free-Riding nach gehen können, und lässt sich nicht auf Gnutella beschränken.

In Distributed Hashtables werden die Peers über pseudo-zufällige Verteilungen auf eine wie auch immer geartete virtuelle Topologie abgebildet. Durch diese Zufälligkeit soll eine im Mittel ausgeglichene Größe der Verantwortungsbereiche der einzelnen Knoten erzielt werden, was eine besonders exponierte und damit für Angreifer lohnende Stellung einzelner Knoten verhindert.
Darüber hinaus ist beispielsweise in Kademlia Netzen vorgesehen, alle (Schlüssel, Wert)-Paare nicht nur auf einem Knoten abzulegen sondern mehrfach im Netz zu replizieren. Dies soll nicht nur die Antwortzeiten durch Caching-Effekte verringern, sondern vor allem eine gewisse Ausfalltoleranz gegenüber dem plötzliche Verschwinden einzelnen Knoten gewährleisten.

#### 2.2.4.3 Sucheffizienz

Wenn es um komplexe Suchen geht sind Index-Server nahezu unschlagbar. Prinzipiell können alle möglichen Suchkriterien angewandt werden wie z. B. Mindest- oder Maximalgröße der Dateien, Bitrate von Audio-/Videoaufnahmen, Dateityp. Auch unscharfe Suchen, also etwa nach „Dateiname klingt ähnlich wie ...“ sind möglich. Einzige Einschränkung ist der Umfang und die Komplexität der Datenbasis des Index-Servers sowie dessen Rechenleistung.
Gefunden werden entsprechend alle Einträge die auf dem angefragten Index-Server verzeichnet sind. Bei Netzen mit mehreren Servern lassen die Implementierungen dem Nutzer oft die Wahl ob nur der eigene oder mehrere Index-Server angefragt werden sollen.
Soll das komplette Netz durchsucht werden, müssen eben alle Server befragt werden, was für die Infrastruktur eine nicht zu vernachlässigende Belastung darstellt.
Die Kosten der Suche tragen bei den klassischen Netzen die Index-Server alleine, sie

müssen den lokalen Dateien-Katalog nach den gewünschten Kriterien absuchen, was sehr rechenintensiv sein kann und dann das Ergebnis zurückgeben. Dafür wird die Suchanfrage nicht durch einzelne Peers limitiert und braucht minimalen Kommunikationsaufwand.

Ebenso flexibel lassen sich die Suchen in Gnutella implementieren, denn jeder Servant kann ja seinen lokalen Dateibestand mit den Suchkriterien einer eingehenden Suchnachricht abgleichen. Unscharfe Suchen sind natürlich genauso möglich.
Problematisch ist die Frage nach der Erreichbarkeit. Wie gross der Teil des Netzes ist, der durch die Suchanfrage abgedeckt wird hängt im wesentlichen von der gewählten TTL und der Anzahl der im Mittel miteinander verbundenen Servants ab. Je höher diese beiden Parameter sind, desto mehr Servants werden erreicht, aber auch desto mehr Nachrichten werden unnötigerweise an Servants verschickt, die bereits von der Suchanfrage wissen. Messungen zeigen [Ripe01], dass bereits nach 7 Hops 95% der Servants erreicht sind. Dennoch kann Gnutella aber bei keiner realistischen Größe der TTL garantiern, dass alle Servants im Netz befragt wurden.
Die Folgekosten für eine Suchanfrage ergeben sich aus dem sehr hohen Kommunikationsaufwand des Flut-Algorithmus der sehr viele Servants im Netz belastet.

Einer der grössten Nachteile von Distributed Hashtables ist, das nur scharfe Suchen möglich sind. Durch die Konstruktion des Suchalgorithmus der sich anhand des Hashwerts durch das Overlay hangelt, ist das eine prinzipielle architekturbedingte Einschränkung.
Durch die Kombination mehrerer scharfer Suchen lassen sich über Umwege auch komplexere Suchanfragen realisieren [HHHL$^+$02] z. B. indem man statt mit kompletten Suchbegriffen mit allen mögliche n-grammen eines Suchbegriffs arbeitet.
Doch die DHT Suchalgorithmen haben auch ihre Stärken. So lässt sich eine Suche in einem DHT-Overlay mit $n$ Knoten in $O(log(n))$ Schritten durchführen und dabei ein Ergebnis garantieren falls der Schlüssel im Netz vorhanden ist. Durch diesen logarithmischen Zusammenhang lassen sich sehr effiziente Suchen selbst in großen Netzen realisieren.
Die Kosten für eine Suche teilen sich auf in die Kommunikationskosten und die im Vergleich zu Index-Servern sehr geringe (weil Hash-Lookup sowohl für Routing als auch für Schlüssel) Rechenzeiten. Beide Kosten müssen für alle beteiligten $O(log(n))$ Knoten auf dem Weg der Suchanfrage miteinkalkuliert werden.

# 3. Entwurf

In diesem Abschnitt soll ausgehend von der Analyse der von Peer-to-Peer Netzen gebotenen Funktionalität und einer Zielsetzung der gewünschten Instant Messaging Fähigkeiten die Brücke zwischen diesen beiden Anwendungen geschlagen werden. Es soll ein für die Implementierung geeigneter, modularer Entwurf erstellt werden.

Dabei steht zunächst die Suche nach geeigneten Anwendungen die als konkrete Basis für die generische Peer-to-Peer Anwendung sowie für die Instant Messaging Oberfläche dienen können. Darauf aufbauend wird eine abstrakte Schnittstellendefinition erarbeitet die einen möglichst großen Teil der verbreiteten Anwendungen abdeckt. Nachdem damit die Anbindung an die externen Anwendungen erledigt ist, wird ein Entwurf für die eigentliche Anwendung erstellt, deren Aufgabe die Umsetzung von Instant-Messaging Ereignissen auf die Fähigkeiten von Peer-to-Peer Netzen ist.

## 3.1 Ein Peer-to-Peer Netz für die Anwendung

Bei der Suche nach einem geeigneten Peer-to-Peer Netz spielen folgende Kriterien eine Rolle. An erster Stelle stellt eine gewisse Verbreitung des Peer-to-Peer Netzes eine Hürde da. Ohne Nutzerakzeptanz wird das beste Peer-to-Peer Netz nicht in seiner gewünschten Form funktionieren können. Da die zu entwerfende Anwendung nicht im luftleeren Raum arbeiten soll, wird ein bereits in Betrieb existierendes Peer-to-Peer Netz vorausgesetzt.
Eine weiteres wichtiges Kriterium ist die Möglichkeit und Komplexität der Anbindung an eine existierende Anwendung. Damit der unnötige Aufwand ein bestehendes Peer-to-Peer Protokoll erneut implemtieren zu müssen elegant umgangen werden kann, soll auf eine bestehende Anwendung aufgesetzt werden. Um davon zu profitieren muss allerdings der Aufwand für diese Anbindung an die externe Anwendung in Grenzen gehalten werden.

Folgend findet sich eine Untersuchung einiger verbreiteter Peer-to-Peer Anwendungen hinsichtlich dieser Kriterien.

**eMule, xMule**

Der eMule Client [eMul] ist ein Open Source Windows Client für das eDonkey2000 Netz. Für Linux, BSD und MacOS X existiert eine Portierung namens xMule [xMul]

die allerdings meist auf einer älteren Version als der aktuellen Windowsversion aufsetzt.
Beiden gemeinsam ist eine grafische Oberfläche, die sich von der Strukturierung an den üblichen Aufgaben einer Filesharing Anwendung wie Suchen, laufende Downloads anzeigen, laufende Uploads anzeigen, etc. anlehnt.

Für die starke Verbreitung des eDonkey2000 Netzes spricht allein schon folgende Tatsache: Beobachtet man auf einem frisch (über einen Endkunden Internet Service Provider wie z. B. T-Online) ins Internet eingewählten Rechner eingehende Pakete auf dem eDonkey Standard Port 4662, dauert es meist nicht länger als eine Minute bis die ersten Verbindungsanfragen eingehen, die hinter der per DHCP vergebenen IP Adresse einen nun nicht mehr aktiven eDonkey Client vermuten.

Ein praktisches Feature zur Fernsteuerung eines eMule Clients ist der integrierte Web-Server über den sich sämtliche Funktionen des Clients fernsteuern lassen. Leider existiert diese Möglichkeit bislang noch nicht in den xMule Portierungen. Ausserdem ist die Ausgabe des Web-Frontends eine sehr benutzerfreundliche grafiklastige Oberfläche, was ein Parsen des HTML-Codes nicht sehr leicht macht.
Bei beiden Anwendungen möglich ist allerdings die Anbindung über direkte Integration in den wie bereits erwähnt offenen Quellcode.

**MLDonkey**

MLDonkey [MLdo] ist ein Multi-Netzwerk Client der die folgenden Peer-to-Peer Netze unterstützt: eDonkey, Overnet, Bittorrent, Gnutella, Gnutella2, Fasttrack, Soulseek, Directconnect und Opennap. Der MLDonkey Client ist in Objective Caml implementiert und läuft daher unter anderem auf Windows, Linux und MacOS X.

Leider unterstützt bislang nur das eDonkey Plugin von MLDonkey ebenfalls einen Upload von Dateien, was die übrigen Peer-to-Peer Netze von einer weiteren Nutzung für die Entwicklung einer Instant Messaging Anwendung ausschließt.

MLDonkey selbst ist als Hintergrundanwendung ohne grafische Oberfläche konzipiert, die den Zugriff auf ihre Dienste über dreierlei Wege ermöglicht. Über einen integrierten Webserver nimmt MLDonkey ebenso bereitwillig Befehle entgegen wie über eine Telnet Sitzung. Außerdem bietet MLDonkey ein spezielles GUI Protokoll an über das grafischen Oberflächen eine Fernsteuerung über primitive Befehle über TCP ausführen können.
Dieses GUI Protokoll eignet sich als Schnittstelle für eine Integration in die zu erstellende Instant Messaging Anwendung hervorragend.

In Tests zeigte sich allerdings, dass MLDonkey ein recht merkwürdiges Verhalten an den Tag legt, was den Upload von Dateien selbst über eDonkey angeht. Eine probeweise freigegebene Datei ließ sich zwar von anderen Rechnern finden, und diese konnten laut Weboberfläche auch eine direkte Verbindung mit dem freigebenden MLDonkey Client herstellen, allerdings liess sich über Stunden hinweg kein Download starten obwohl keine weiteren Clients in der Warteschlange waren. Bis auf zwei einzelne erfolgreiche Transfers, die sich trotz gleicher Bedingungen nicht reproduzieren ließen, war es nicht möglich über den MLDonkey Client Dateien zu anderen Clients zu übertragen.
Das disqualifiziert den MLDonkey Client in den aktuellen getesteten Versionen 2.5.4 bis 2.5.11 natürlich, da eine zugesicherte Übertragungsfähigkeit für die Instant Messaging Anwendung vonnöten ist.

#### Overnet Command Line Client

Der Overnet Command Line Client [Over] ist eine konsolenbasierte Anwendung die in kompilierter Form für Windows, Linux und MacOS X verfügbar ist. Wie aus dem Namen ersichtlich basiert er auf dem Overnet Peer-to-Peer Netz.

Overnet ist nicht zuletzt dank der neueren hybriden grafischen Overnet Clients von MetaMachine, die sowohl eDonkey2000 als auch Overnet unterstützen, weit verbreitet. Homepage und Overnet Client schätzen um die 900000 aktive Anwender.

Für die Integration des Overnet Command Line Clients in andere Anwendungen bieten sich zwei Lösungsvarianten an. Zum einen lässt er sich als Kindprozess mittels umgeleiteter Ein- und Ausgaben fernsteuern. Zum anderen ist der Client mit einem eigenen GUI Interface ausgestattet, welches sich über TCP ansprechen lässt. Dieses GUI Interface wird zum Beispiel vom ed2k-gtk-gui Projekt [edGU] verwendet.

#### Fazit

Aufgrund der guten Verbreitung und der einfachen Integration über ein eigenes GUI Interface fällt die Wahl für eine Peer-to-Peer Anwendung für die Implementierung der generischen Instant Messaging Anwendung auf den Overnet Command Line Client.

## 3.2 Schnittstellendefiniton für generische Peer-to-Peer Anwendungen

Um einen einfachen Austausch der Peer-to-Peer Anwendung zu ermöglichen wird hier eine generische Abstraktion entworfen, hinter der sich die Unterschiede zwischen den konkreten Netzen und Anwendungen verbergen lassen. Diese Abstraktion stellt fortan die einzige Schnittstelle dar, über die mit der Peer-to-Peer Anwendung interagiert werden darf, um die Generizität der Schnittstellenabstraktion sicherzustellen.

Folgende Funktionalität lässt sich in allen betrachteten Peer-to-Peer Anwendungen, in welcher konkreten Ausprägung auch immer, identifizieren:

**Datei freigeben:** Die Freigabe von Dateien im Netz ist bei allen Filesharing Netzen eine zentrale Fähigkeit. Dabei wird je nach zugrundeliegender Netzart nicht nur die Datei lokal zum entfernten Zugriff freigegeben, sondern gegebenfalls auch Publikationsinformation im Netz verbreitet, also z. B. ein zentraler Indexserver benachrichtigt.

**Dateifreigabe aufheben:** Lokal freigegebene Dateien können natürlich auch wieder vom entfernten Zugriff ausgenommen werden. Durch das Peer-to-Peer Netz oder entfernte Anwender eventuell schon replizierte Dateien sind davon meist[1] nicht betroffen.

**Suche:** Die Suche nach einem oder mehreren Suchbegriffen dient dem Auffinden von Dateien, bei denen der Dateiname oder Teile davon bekannt sind. Für die abstrakte Schnittstelle soll die Suchfähigkeit auf die scharfe Suche nach Zeichenketten, die im Dateinamen enthalten sind, beschränkt werden. Spezielle

[1] Die effektive Semantik hängt dabei vom zugrunde liegenden Peer-to-Peer Netz ab

Suchoptionen wie die Einschränkung nach Meta-Informationen, die zwar in einigen aber nicht in allen Peer-to-Peer Netzen möglich sind, fallen hier der Vereinheitlichung zum Opfer.

**Mitteilung Suchergebnis:** Die Peer-to-Peer Anwendungen liefern die Ergebnisse meist sobald sie gefunden wurden. Da in fast allen Peer-to-Peer Netzen mehr als eine Instanz befragt werden kann, kommen die Ergebnisse nicht synchron an. Für die generische Schnittstelle bedeutet das, dass keine Aussage über das zeitliche Verhalten von Suchergebnissen gemacht werden kann und prinzipiell jederzeit nach Suchbeginn ein oder mehrere Ergenisse zurückgemeldet werden können.

**Download einer Datei:** Ist eine den Suchkriterien entsprechende Datei gefunden, lässt sich der Download dieser Datei dann über Angabe einer Referenz auf das Suchresultat starten.

**Mitteilung Beendeter Download:** Sobald der Download beendet ist, sollte die Peer-to-Peer Anwendung über die Schnittstelle eine Benachrichgung abgeben. In vielen Peer-to-Peer Anwendungen ist es möglich den Fortschritt des Transfers in feingranularen Schritten nachzuverfolgen. In der generischen Schnittstelle wird aber lediglich die erfolgreiche Beendigung eines Downloads als Ereignis definiert, was aufgrund der angepeilten Nutzung mit kleinen Textdateien einen nicht allzu schwerwiegenden Kompromiss zugunsten einer Vereinfachung der Schnittstelle darstellt.

## 3.3 Eine grafische Oberfläche für die Anwendung

Auf der Suche nach einer geeigneten grafischen Oberfläche für die Instant Messaging Anwendung, spielen vor allem die beiden Kriterien der Zweckmäßigkeit und der einfachen Integration eine Rolle.
Zweckmäßig sollte die grafische Oberfläche insofern sein, als dass sie sich an die durch weit verbreitete IM Anwendungen eingeführten Konventionen wie eine Buddy Liste und separate Chat-Fenster für Nachrichten hält, und dadurch dem Anwender eine schnelle Einarbeitung ermöglicht.
Die einfache Integration der Oberfläche in die eigentliche Instant Messaging Anwendung ist schließlich wünschenswert, um den Implementierungsaufwand im Rahmen zu halten.

### Individuell erstellte Oberfläche

Ein naheliegender Ansatz stellt der Gedanke dar, eine eigene grafische Oberfläche zu implementieren. Vorteil dieser Variante ist sicherlich, die völlige Freiheit beliebige Fähigkeiten in die Oberfläche einzubauen oder bewusst weg zu lassen.
Nachteil ist ganz klar der hohe Aufwand für die Implementierung einer eigenen Oberfläche, der eigentlich aufgrund der Vielzahl von bereits existierenden quelloffenen Implementierungen nicht nötig ist.
Um den Aufwand der vor allem in der Interaktion mit einer Grafik-Bibliothek wie QT oder GTK+ besteht zu reduzieren, ließe sich die grafische Oberfläche auch als reine Textmodus-Oberfläche implementieren, allerdings schränkt das wiederum den Benutzerkomfort erheblich ein.

### Kopete

Kopete [Kope] ist ein Multi-Protokoll fähiger Instant Messenger der Teil des KDE Projekts ist. Alle unterstützten Protokolle (darunter u. a. auch ICQ und Jabber) sind bei Kopete als Plugins realisiert, deren konkrete Implementierung dadurch vor Kopete versteckt wird. Der Kern von Kopete dient lediglich als einheitliche grafische Oberfläche für die einzelnen Plugins.

Kopete eignet sich daher recht gut als generische Instant Messaging Oberfläche, eine Anbindung lässt sich über das dazubinden eigener Plugins realisieren.

### EB-lite

EB-lite [EBli] ist ein komplett modular konzipierter Multi-Protokoll fähiger Instant Messenger, der aus folgenden Komponenten besteht: eb-lite ist die Kernkomponente, der „Core“, der für die Handhabung der einzelnen Protokolle zuständig ist, welche wiederum als Service Plugins realisiert sind. EbQT ist eine grafische Oberfläche die über TCP mit dem Core Verbindung aufnehmen kann und dann dem Nutzer die vom Core generierten Ereignisse präsentiert.

Für die Anbindung von EB-lite als grafische Oberfläche ergeben sich zwei Ansatzpunkte: Zum einen läßt sich ein eigenes Service Plugin für den Core *eb-lite* implementieren, wobei die Kopplung durch das Linken mit dem Code der Kernkomponente geschieht. Zum anderen lässt sich eine wesentlich lockerere Kopplung erreichen indem man eine eigene Core implementiert, die dann lediglich über TCP mit der EbQT GUI kommuniziert.

### Fazit

Durch den modularen konzipierten Aufbau und seine gut dokumentierte GUI-Core Schnittstelle prädestiniert sich EbQT als Oberfläche der Wahl für die Implementierung der generischen Instant Messenger Anwendung. Da die Kopplung über TCP Nachrichten wesentlich lockerer ist als über das Binden mit existierendem Quellcode, wird EbQT damit dann auch gegenüber Kopete der Vorzug gegeben.

## 3.4 Schnittstellendefiniton für generische grafische Instant Messenger Oberflächen

Nachdem nun mehrere Möglichkeiten für die Anbindung an eine grafische Oberfläche diskutiert wurden wird klar, dass auch hier eine generische Schnittstelle für einen einfachen Austausch der GUI Komponente von Nutzen sein kann.
Daher soll nun eine generische Schnittstelle entworfen werden, hinter deren Abstraktion sich die Details der einzelnen Oberflächenimplementierungen verbergen lassen.

Folgende Ereignisprimitive lassen sich zu diesem Zweck identifizieren:

**Nachricht senden:** Eine Textnachricht soll an einen Kontakt geschickt werden. Die grafische Oberfläche liefert als Information den zu sendenden Text und eine Referenz auf den Kontakt der als Addressat gewünscht wurde.

**Nachricht empfangen:** Eine Textnachricht wurde empfangen. Als Folge wird die grafische Oberfläche angewiesen die eingegangene Nachricht im entsprechenden, mit dem Kontakt assoziierten Fenster, anzuzeigen bzw. ein neues Textchat Fenster dafür zu erstellen.

**Änderung an Onlinestatus vornehmen:** Der Anwender hat über die grafische Oberfläche eine Änderung an seinem Onlinestatus vorgenommen. Die Oberflächenschnittstelle gibt dies mit diesem Ereignis an die Instant Messaging Anwendung weiter. Soll ein automatisches Away z. B. bei längerere Nichtbenutzung der Maus unterstützt werden muss dies daher die grafische Oberfläche implementieren.

**Änderung an Onlinestatus empfangen:** Der Onlinestatus eines Kontakts hat sich geändert und die grafische Oberfläche wird angwiesen dies über die Buddy Liste anzuzeigen und gegebenenfalls je nach GUI auch eine akustische Benachrichtigung abzuspielen.

Die Definition der Schnitstelle beschränkt sich auf den Austausch von Textnachrichten und des Onlinestatus, da dies die beiden zentralen IM Fähigkeiten sind. Für die Repräsentation des Onlinestatus genügen einer der drei Zustände Online, Offline und Away sowie ein zusätzlicher Statustext.

## 3.5 Anwendungsentwurf

Durch die Klärung der Frage der Anbindung an die externe Peer-to-Peer Anwendung, sowie an eine geeignete GUI, steht nun dem Entwurf der eigentlichen Anwendung nichts mehr im Wege. Deren zentrale Aufgabe besteht darin die Ereignisse, die an der abstrakten Instant Messaging Oberflächenschnittstelle auftreten können, auf Ereignisse für die abstrakte Peer-to-Peer Schnittstelle abzubilden und umgekehrt.

### 3.5.1 Abbildung zwischen IM Ereignissen und Dateien

Da die abstrakte Peer-to-Peer Schnittstelle Kommunikation zwischen entfernten Rechnern lediglich über den Austausch von Dateien zulässt, müssen sämtliche IM Ereignisse auf Dateien abgebildet werden, um sie dem Kommunikationspartner zugänglich zu machen.
Dieses Vorgehen impliziert eine ganze Reihe weiterer Entwurfsentscheidungen:

#### 3.5.1.1 Polling Verfahren

Für den potentiellen Empfänger einer Nachricht gibt es nur eine Möglichkeit von neuen Nachrichten im Peer-to-Peer Netz zu erfahren: Die Suchfunktion der abstrakten Peer-to-Peer Schnittstelle.
Daher muss er in regelmäßigen Zeitabständen nach neu verfügbaren Dateien suchen („Polling“). Dies setzt wiederum vorraus, dass sich die für ihn bestimmten Dateien einem gewissen Namensschema gehorchen, um nach einem eindeutigen Suchbegriff suchen zu können.

Wie bereits in der Analyse (siehe Abschnitt 2.2.4.3) gesehen, stellen Suchen und meist auch parallel dazu die entsprechenden Publikationsvorgänge für alle Peer-to-Peer Netzarten eine nicht zu vernachlässigende Belastung dar:

In Index-Server basierten Netzen vermindert die erhöhte Such- und Publikationslast die Kapazität der zentralen Server. In Gnutella-ähnlichen Netzen stellen Suchen generell eine starke Belastung dar, wenn auch die Publikationskosten minimal sind, und in Distributed Hashtables ziehen sowohl Suchen als auch Publikationsvorgänge Kosten für logarithmisch zur Netzgröße wachsenden Anzahl Knoten nach sich.
Um diese Belastung zu minimieren stellen sich folgende Fragen:

#### 3.5.1.2 Welche Ereignisse sollen auf welche und wieviele Dateien abgebildet werden?

Mit Blick auf die abstrakte Instant Messaging Oberflächenschnittstelle, lassen sich zwei grundsätzlich verschiedene Ereignisse identifizieren, die auf Dateien abgebildet werden müssen.
Zum einen finden sich Chat-Nachrichten, die von einem Anwender an genau einen anderen Anwender geschickt werden müssen.
Zum anderen gibt es aber auch die Online Status Informationen eines Anwenders die an mehrere fremde Anwender verschickt werden sollen.

Der naheliegende Ansatz der sich daraus ergibt, ist nun eine Datei für den Status des Anwenders im Peer-to-Peer Netz freizugeben, und für jeden Kontakt eine weitere, die die gesendeten Textnachrichten enthält.
Ein Anwender mit $c$ Kontakten in der Buddy-Liste müsste also $1 + c$ Dateien publizieren um seinen Onlinestatus und seine versendeten Nachrichten abzuschicken. Der Suchaufwand um den Onlinestatus seiner Kontakte herauszufinden und eventuelle Textnachrichten von ihnen zu empfangen, ließe sich bei Kodierung des Absenders in den Dateinamen auf $c$ Suchanfragen minimieren. Die Suche nach einem Empfänger, was nur noch 1 Suchanfrage bedeuten würde, ist nicht direkt möglich, da die Onlinestatusdatei an viele Empfänger gerichtet ist.
Man sieht, dass hierbei schon einiger Such- und Publikationsaufwand generiert wird, und es stellt sich die Frage nach einer effizienteren Abbildung der IM Ereignisse auf die Dateien.

Ein Ansatz diesen Aufwand herabzusetzen, stellt die Verwendung im folgenden als Kontextdateien bezeichneter Konstrukte dar. Für jeden Kontakt auf der Kontaktliste wird eine solche Kontextdatei im Peer-to-Peer Netz publiziert, die neben dem Online Status alle gesendeten Nachrichten, sowie weitere Verwaltungsinformationen enthält. Das bedeutet aber auch, dass zwei Kontakte die miteinander kommunizieren wollen sich gegenseitig auf ihrer Buddy-Liste haben müssen, was allerdings in nahezu allen traditionellen Instant Messaging Anwendungen aus Spam-Schutz Gründen ähnlich gehandhabt wird.
Was die Kosten angeht schneidet diese Lösung deutlich besser ab: Für den Anwender mit $c$ Kontakten bedeutet das insgesamt $c$ Dateien publizieren zu müssen, was eine geringfügige Verbesserung darstellt. Bei der Suche jedoch reduziert sich der Aufwand auf ein Minimum weil über die 1:1 Beziehung zwischen allen Kontakten durch die Kontextdatei eine einzige Suchoperation nach dem in den Dateinamen kodierten Empfänger ausreicht.

#### 3.5.1.3 Aktualisierungsrate

Die Belastung der Peer-to-Peer Netzinfrastrukur durch die Instant Messaging Anwendung hängt allerdings nicht nur von der Anzahl der Such- und Publikationsvorgänge ab, sondern auch von deren Frequenz.

Regelmäßige Updates des Onlinestatus sorgen dafür, dass vom Netz getrennte Anwendungen durch Timeout Bedingungen erkannt werden. Außerdem kommen mit der Zeit auch neue Textchat-Nachrichten hinzu, die in der Kontexdatei gespeichert werden müssen und dadurch einen neuen Publikationsvorgang erzwingen.
Man sieht, dass hier für die Aktualisierungsraten ein Kompromiss zwischen je länger desto geringere Netzbelastung und umso kürzer desto anwenderfreundlicheren Reaktionszeiten auf entfernte IM Ereignisse getroffen werden muss.
Um das Peer-to-Peer Netz nicht unnötig zu belasten, wird daher ein Mindestintervall vorgegeben, das nach der Veröffentlichung einer neuen Datei abgewartet werden muss bevor erneut dieselbe Kontextdatei mit aktuelleren Daten veröffentlicht werden darf. Existieren nach Ablauf dieses Mindestintervalls noch keine neuen Nachrichten, wird die Datei erst rechtzeitg vor dem Ablauf der Gültigkeit ihres Online Status aktualisiert. Damit wird der Kontakt von entfernten Clients nicht fälschlicherweise als Offline erkann wird.

Eine weitergehende Betrachtung der Problematik einer optimalen Wahl der Timer findet sich im Abschnitt Test und Betrieb, Abschnitt 5.2.1.

#### 3.5.1.4 Kodierung des Dateinamens

Um eine geeignete Kodierung für den Dateinamen der Kontextdatei zu finden, hilft es, sich den Vorgang der Suche und dem anschließenden Download einer neuen Datei vor Augen zu halten.
Zunächst sollte der Dateiname einen für die Suche nach dem Empfänger eindeutigen Identifikator tragen.
Außerdem wäre es wünschenswert, schon bei der Suche zu erkennen, ob die betreffende Datei veraltete oder neue Informationen trägt. Denn in vielen Peer-to-Peer Netzen werden Publikationsinformationen zwischengespeichert, was zur Folge hat, dass nicht alle zurückgelieferten Suchergebnisse tatsächlich noch als verfügaber Dateien im Netz vorhanden sein müssen. Daher wird ein Zeitstempel in den Dateinamen integriert, aus dem zusammen mit einem Identifikator des Absenders, die Aktualität der Datei schon aus dem Dateinamen des Suchergebnis kombiniert werden kann, ohne einen Download zu starten der eventuell nie erfolgreich beendet werden würde.

Die globale Gültigkeit von Zeitstempeln wird durch die Verwendung einer zeitzonenunabhägige Uhrzeit (z. B. UTC) erreicht. Daher sollten die eingesetzten Rechner regelmäßig ihre Uhrzeit mit Time-Servern abgleichen, um durch eine vernachlässigbare Fehlergröße der lokalen Uhrzeit die eigene Kommunikationsfähigkeit nicht zu beeinträchtigen. Ein Rechner mit falsch gestellter Uhr wird aber niemals die Kommunikation zwischen Dritten beeinträchtigen, was durch die Realisierung über 1:1 Beziehungen der Kontextdateien gesichert ist.

#### 3.5.1.5 Verschlüsselung & Authentifizierung

Auf der Abstraktionsebene oberhalb der Peer-to-Peer Anwendung lassen sich keine Zugriffsrechte auf Dateien für einzelne Peer-to-Peer Teilnehmer vergeben, sondern Dateien nur global freigeben. Daher sollten die ausgetauschten Dateien verschlüsselt werden um nur berechtigten Empfängern die Möglichkeit zu geben, deren Inhalt zu lesen.
Gleiches gilt für den Empfang von Dateien. Da oberhalb der Anwendung nicht sicher festgestellt werden kann, von wem eine Datei im Netz publiziert wurde, empfiehlt es sich den Absender der Dateien durch eine sichere Authentifizierung zu identifizieren.

### 3.5.2 Kontaktlisten Management

In der generischen Oberflächenschnittstelle ist keine Funktionalität für die Verwaltung der Kontaktliste vorgesehen, um die Vorraussetzungen für die konkrete Oberfläche nicht unnötig hoch zu setzen. Daher wird das komplette Management der Kontaktliste in die generische Instant Messaging Anwendung für Peer-to-Peer Netze integriert.
Alle Kontakte werden über eine eindeutige 128 Bit Zufallszahl identifiziert, die sogenannte Kontakt ID. Diese wird beim ersten Start automatisch ausgewürfelt, wenn sie noch nicht vorhanden ist.
Die Verwaltung der Kontaktliste obliegt dem lokalen Anwender, der Austausch von Kontakt IDs über das Peer-to-Peer Netz wird nicht explizit vorgesehen. Da sich die IM Anwender häufig sowieso persönlich kennen (siehe Abschnitt 2.1.1) oder wenigstens per E-Mail Kontakt zu einander aufnehmen können, sollte der Austausch der Kontakt IDs und der kryptografischen Schlüssel auf diesem Weg geschehen.

### 3.5.3 Forward Caching

Prinzipiell kann jeder Client nicht nur nach für ihn sinnvollen Dateien im Peer-to-Peer Netz suchen, sondern nach allen beliebigen. So stellt sich die Frage nach sinnvollen Anwendungsfällen für diese Fähigkeit.
Eine Möglichkeit besteht darin, Dateien die für andere Teilnehmer veröffentlicht wurden zu cachen um deren Erreichbarkeit zu erhöhen, sogenanntes „Forward Caching“. Angenommen Teilnehmer A und Teilnehmer B können beide keine eingehende Verbindung zulassen. Teilnehmer C hingegen kann sowohl Dateien mit A als auch mit B tauschen, weil er eingehende Verbindungen nicht durch eine Firewall blockiert und die zugrundeliegende Peer-to-Peer Anwendung für einen erfolgreichen Dateitransfer mindestens einen Teilnehmer mit eingehenden Verbindungen benötigt[2]. Dann könnte über den als Relaystation fungierenden Teilnehmer C trotzdem eine auf direktem Wege nicht mögliche Kommunikation zwischen A und B stattfinden.
Das setzt voraus, dass ständig von neuem möglichst aktuelle Nachrichten für fremde Teilnehmer nachgeladen werden müssen.
Ein weiterer Vorteil dieses Forward Caching besteht in der Erhöhung der Erreichbarkeit von Teilnehmern in Peer-to-Peer Netzen, in denen nur eine gewisse Umgebung abgesucht wird und nicht das ganze Netz durchsucht werden kann (z. B. bei Gnutella).
Der Nutzen von Forward Caching insbesondere in Netzen mit globaler Suchweite dürfte sich aber in Grenzen halten. Vor allem angesichts der Tatsache, dass durch dieses Verfahren zusätzliche Suchkosten entstehen, so dass spätere Tests (siehe Abschnitt 5.2.2) erst noch zeigen müssen inwieweit Forward Caching sinnvoll ist.

### 3.5.4 Dateinamen und Suchen

Nachdem nun alle Anforderungen an die Bildung eines Dateinamens klar sind, soll ein Blick auf dessen Struktur und die dadurch implizierte Bildung der Suchzeichenketten geworfen werden.

Der Name der Kontextdatei wird nach folgendem Schema gebildet:
*<id_from>-<id_to>-<cache_mark>.<timestamp>*

[2]Dies ist eine realistische Annahme, die für die meisten aktuellen File-Sharing Anwendungen zutrifft, siehe z. B. Push-Mechanismus bei Gnutella (Abschnitt 2.2.2).

Dabei steht <*id_to*> für die ID des Empfängers. Diese ID stellt dann auch gleichzeitig den Suchbegriff dar, denn wenn jeder IM Teilnehmer nach seiner eigenen ID sucht ist dadurch sichergestellt, dass alle Nachrichten ihren Empfänger errreichen, sofern dieser Online ist.
Aus den beiden Werten von <*id_from*>, der ID des Absenders und <*timestamp*>, dem Zeitpunkt der Veröffentlichung, kann der Suchende herausfinden ob die Nachricht aktueller ist als die zuletzt von dem betreffenden Kontakt empfangene.
Der Teil des Dateinamens der mit <*cache_mark*> bezeichnet ist, bedarf etwas genauerer Erklärung. Er stellt einen weiteren Zeitstempel dar, nur dass dieser zu Testzwecken mit wenigen Clients „verschmiert“ werden kann. Er bildet den Suchbegriff für eine Suche nach aktuellen, fürs Forward Caching nützlichen Dateien. Da in der generischen Schnittstelle nur scharfe Suchen zugelassen sind, wird dieser verschmierte Zeitstempel benötigt, der den exakten Zeitstempel auf bestimmte Intervalle rundet und dadurch eine Suche nach den im letzten Intervall veröffentlichten Nachrichten ermöglicht. Sobald die Anzahl der teilnehmenden Clients eine gewisse Grenze überschreitet, und mehrere Clients pro Sekunde eine Nachricht veröffentlichen, kann natürlich auch nach dem sekundengenauen Zeitstempel <*timestamp*> gesucht werden und die zusätzliche Zeichenkette <*cache_mark*> kann entfallen.

### 3.5.5 Spezialfälle beim Overnet Command Line Client

Beim Einsatz der konkreten Implementierung der Peer-to-Peer Schnittstelle für den Overnet Command Line Client traten nicht vorhersehbare Effekte auf, deren Berücksichtigung im Entwurf der Instant Messaging Anwendung für den reibungslosen Ablauf nötig wurden, und die hier kurz beschrieben werden sollen.

#### Überlauf des Suchnamensraums

In den ersten Implementierungen kam es nach etwa halbstündigen einwandfreien Testläufen in denen Instant Messaging Nachrichten ohne Probleme über das Peer-to-Peer Netz zwischen zwei entfernten Rechnern ausgetauscht wurden, zu einer schrittweisen Degradierung der Verbindungsqualität. Zunächst starb die Kommunikation in eine Richtung ab und kurz darauf auch in die andere.
Erstaunlicherweise ließ sich dieses Phänomen durch einen Neustart der Anwendungen nicht beheben, sondern hielt oftmals bis zu einem ganzen Tag an. Am darauf folgenden Tag ließ sich das exakt gleiche Verhalten von neuem reproduzieren.

Eine Untersuchung des Phänomens ergab, dass nach einer gewissen Zeit die Suchanfragen einfach keine brauchbar aktuellen Ergebnisse mehr zurück liefern.
Anhand von mit Ethereal [Ethe] erstellter Protokoll-Dumps von im Overnet Command Line Client durchgeführten Testsuchen ließ sich feststellen, dass jeder befragte Peer maximal 100 Ergebnisse zurückliefert. Bei genauerer Betrachtung dieser 100 Ergebnisse fiel auf, das es sich dabei um die 100 ältesten auf dem Knoten gespeicherten Ergebnisse handelt. Die Beschränkung auf 100 Ergebnisse ist eine verständliche Einschränkung um Ressourcen zu sparen, sowohl in Hinblick auf lokalen Speicher wie auch auf Übertragungskapazität. Warum jedoch nur die ältesten Ergebnisse vorgehalten werden lässt sich sinnvoll lediglich als Flood-Protection gegen das Fluten bestimmter Knoten mit sinnlosen Werten begründen. Ohne Kenntnis des Quellcodes bleibt das aber spekulativ.

Unabhängig von der Intention dieser Maßnahme ist somit aber der Grund für das Absterben der Kommunikation geklärt:

Zur Erinnerung: Overnet trennt den Dateinamen zwischen Seperatorzeichen wie Bindestrich und Punkt in Teile auf und speichert den Dateihash unter den Schlüssel aller Dateinamensteile. Nachdem nun 100 verschiedene Dateien unter den Knoten, die für den Empfänger-Teil des Dateinamens zuständig sind abgelegt wurden, kommen eben keine weiteren neuen Nachrichten mehr durch und der Empfänger sieht bei der Suche nur noch alte Dateien.
Dies dauert an, bis entweder die Knoten, die die betreffenden Schlüssel speichern aus dem Netz verschwinden, oder das Wiederveröffentlichungs-Intervall, das bei Overnet vermutlich zwischen 12 und 24 Stunden liegt, abläuft.

Die Lösung für dieses Problem ist so einfach wie effektiv: Der Überlauf des Suchnamensraums wird verhindert indem rechtzeitig vor dem Überlauf regelmäßig ein neuer Namensraum eröffnet wird!
Das geschieht folgendermaßen: Der Empfänger-Teil des Dateinamens wird um eine sogenannte Timefix Zeichenkette ergänzt, die sich regelmäßig nach einem für alle Anwedungen erkennbaren Muster ändert und dadurch die für die Empfänger-Zeichenkette zuständigen Knoten wechselt, bevor es dort zu einem Überlauf kommt.
Für die globale Vorhersagbarkeit dieser Timefix Zeichenkette bieten sich „verschmierte“ Zeitstempel an, die eigentlich wie normale Zeitstempel gebildet werden, aber größere Intervalle wie z. B. 10 Minuten abdecken und nicht sekundengenau auflösen.
Die Wahl dieses Intervalls $t_{timefix\_intervall}$ ergibt sich analytisch aus der minimalen Zeit $t_{min\_republish}$, die vergehen muss bis eine Kontextdatei erneut veröffentlicht werden darf und der Namensraumgröße von 100 Dateien, sowie der Anzahl Kontakte $c$ von denen Nachrichten erwartet werden:

$$t_{timefix\_intervall} < \frac{100 * t_{min\_republish}}{c}$$

Die Timefix Zeichenkette muss dann sowohl beim Veröffentlichen als auch bei der Suche nach der Datei als Teil des Empfängers angegeben werden.

Die Bildung des Dateinamens erfolgt demzufolge nach folgendem Schema:
*<id_from>t<timefix>-<id_to>t<timefix>-<cache_mark>.<timestamp>*

Die zusätliche Timefix Zeichenkette *t<timefix>* sorgt dafür, dass die jeweils durch die Seperatorzeichen „-“ getrennten Zeichenketten in gewissen Zeitabständen andere Hashwerte produzieren und dadurch andere Knoten die Verantwortung für die Speicherung der Werte bekommen.

# 4. Implementierung

Im folgenden Kapitel soll die Implementierung der generischen Instant Messenger Anwendung für Peer-to-Peer Netze, die den Namen „IM4P2P“ tragen wird, dokumentiert werden.

Dazu wird zunächst die Realisierung der generischen Anbindung an die externen Peer-to-Peer und GUI Anwendungen über ein eigenes Framework diskutiert. Anschließend wird die Implementierung der eigentlichen IM4P2P Programmlogik erläutert.

## 4.1 Zielplattform

Die Anwendung IM4P2P wird als einfache Konsolenanwendung unter Linux erstellt. Die Interaktion mit dem Benutzer geschieht über die grafische Oberfläche einer externen Anwendung. Bei der Implementierungssprache fiel die Wahl auf C++. Das Programm wurde unter einer aktuellen Linux Distribution mit einem Kernel der Version 2.4.21, glibc 2.3.2 und gcc 3.3.1 erstellt und getestet, sollte aber prinzipiell mit jedem Standard C++ Compiler übersetzbar und unter allen Linux und UNIX-Derivaten entweder direkt lauffähig bzw. einfach zu portieren sein.

Für den Overnet Command Line Client existieren Binärpakete für Linux, MacOS X und Win32 [Over].
Die Instant Messenger Oberfläche EbQT ist im Quellcode verfügbar [EBli], benötigt allerdings für die grafische Darstellung Qt Bibliotheken in der Version 3.

## 4.2 Das IM4P2P Framework

Um einen einfachen Austausch der Komponente Peer-to-Peer Anwendung sowie der Komponente grafische Oberfläche zu ermöglichen, soll deren Integration über die im vorherigen Kapitel entworfenen generischen Schnittstellen erfolgen.
Die einfache Austauschbarkeit der konkreten Implementierung und die generischen Eigenschaften der Schnittstelle sollen dabei über abstrakte Klassenschnittstellen gewährleistet werden.

Für die Implementierung der konkreten Schnittstellen wird von IM4P2P dazu ein gemeinsames Framework zur Verfügung gestellt, an dessen Vorgaben sich die Schnittstellenimplementierungen halten müssen, um eine reibungslose Interaktion zu ermöglichen.

In den folgenden Abschnitten wird dieses IM4P2P Framework und die Anbindung der generischen Schnittstellen an IM4P2P näher beschrieben.

### 4.2.1 Allgemeines

Zunächst sollen einige allgemeine Funktionen des IM4P2P Frameworks betrachtet werden, die sowohl von den Schnittstellenimplementierungen als auch von der IM4P2P Implementierung selbst genutzt werden können.

#### Verzeichnisstruktur

Die Klassendefinitionen (*.h) und -implementierungen (*.cpp) des von IM4P2P und dem damit eng verzahnten IM4P2P Frameworks befinden sich gemeinsam mit dem Makefile im Hauptverzeichnis des IM4P2P Projekts.
Die konkreten Schnittstellenimplementierungen befinden sich in jeweils eigenen Unterverzeichnissen, um die modulare Aufteilung schon auf dieser Ebene erkennbar zu machen und gegebenenfalls zwischen mehreren Implementierungen einer Schnittstelle durch einfaches ändern des Unterverzeichniseintrags im Makefile zu wechseln.

#### Ausnahmebehandlung

Um die Lesbarkeit und Wartbarkeit des Quellcodes zu erhöhen, erlaubt das IM4P2P Framework den Schnittstellenimplementierungen, Fehler mittels C++ Exceptions anzuzeigen. Zu diesem Zweck definiert das Framework eine eigene Klassen-Hierarchie unterhalb der Klasse `GenerealException`, die die konkreten Implementierungen verwenden und je nach Bedarf erweitern dürfen.

#### Objektorientierte Kapselung für Systemaufrufe

Für eine einfachere Handhabung von Systemaufrufen sind im IM4P2P Framework spezielle Klassen mit deren Umgang betraut: `TCPSocket` für zuverlässige Netzwerkkommunikation über Sockets, `File` für Datei- und Verzeichnisoperationen und `Timestamp` um vom System einen aktuellen sekundengenauen Zeitstempel zu beziehen.

#### Globale Konfigurationsdatei

Die Klasse `Configuration` ermöglicht allen Programmkomponenten den Zugriff auf eine globale Konfigurationsdatei namens `im4p2p.conf` und die darin abgelegten (Name, Wert) -Paare.
Da diese Konfigurationsdatei allen Komponenten von IM4P2P zugänglich sein soll, aber trotzdem nur eine Instanz von `Configuration` benötigt wird, ist die Klasse nach dem Entwurfsmuster Singleton konzipiert, das den Zugriff auf die einzig gültige Instanz über eine statische Member-Funktion ermöglicht.

### 4.2.2 Die generische Peer-to-Peer Anwendungsschnittstelle

Ziel der generischen Peer-to-Peer Anwendungsschnittstelle soll zum einen sein, die konkrete Implementierung der Peer-to-Peer Anwendungsanbindung hinter der im vorigen Kapitel (siehe Abschnitt 3.2) definerten generischen Schnittstelle zu verbergen. Zum anderen soll aber trotzdem eine effiziente Kopplung mit der IM4P2P Anwendung realisiert werden können.

Um für die Kommunikation zwischen IM4P2P Framework und den konkreten Schnittstellenimplementierungen eine gemeinsame Begriffswelt zu bilden wurden die wichtigsten Parameter in generischer Form als Klassen definiert.
Die Klasse `P2PFile` beschreibt eine Datei, die von der Anwendung heruntergeladen wurde oder verteilt werden soll mit ihrem Dateinamen, Inhalt und Größe.
Die Klasse `P2PResult` beschreibt ein Suchresultat, das von der Anwendung zurückgeliefert wird mit Namen, Größe und Verfügbarkeit[1]. `P2PResult` darf von der konkreten Schnittstellenimplementierung in abgeleiteter Form polymorph verwendet werden um z. B. einen Suchidentifikator oder einen Hashwert für die interne Datenhaltung zu ergänzen.

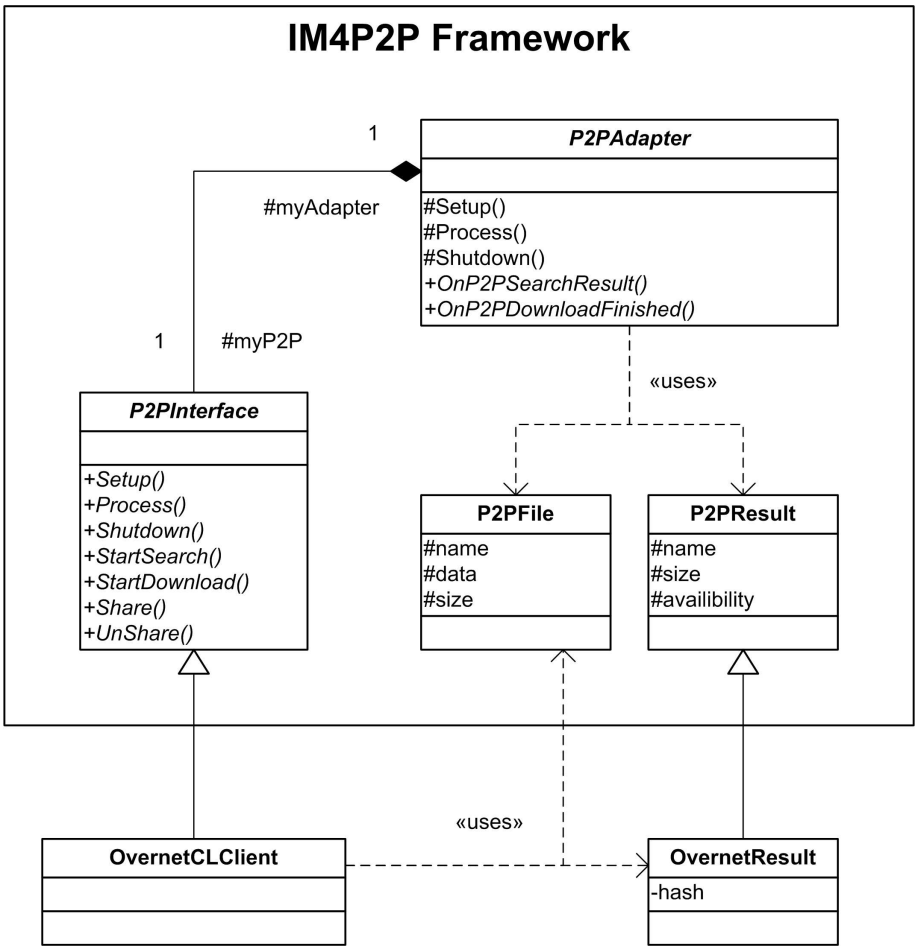


Abbildung 4.1: UML Klassendiagramm der Peer-to-Peer Anwendungsintegration in IM4P2P

[1]Diese drei Parameter werden von allen verbreiteten Peer-to-Peer Anwendungen bei Suchresultaten mitgeliefert. Die meisten geben noch weiter Metadaten wie z. B. Bitrate bei mp3 Dateien an die aber für IM4P2P keine Rolle spielen.

Die Peer-to-Peer Anwendungsschnittstelle wird innerhalb des IM4P2P Frameworks durch die abstrakte Klasse `P2PInterface` repräsentiert. Die von dieser abstrakten Klasse angebotene Funktionalität muss durch die konkrete Peer-to-Peer Anwendungsanbindung erbracht werden, die zu diesem Zweck die folgenden von der Klasse `P2PInterface` geerbten Funktionen implementieren muß:

**Setup():** Der Verbindungsaufbau[2] zur Peer-to-Peer Anwendung wird veranlasst, gegebenfalls eine Verbindung der Anwendung mit dem Peer-to-Peer Netz hergestellt.

**Process():** Die Nachrichtenschleife soll abgearbeitet werden: Von der Peer-to-Peer Anwendung generierte Ereignisse werden verarbeitet und dem Framework über die entsprechenden Ereignisfunktionen der Adapterklasse (s. nächster Absatz) mitgeteilt.

**Shutdown():** Die Verbindung zur Peer-to-Peer Anwendung soll abgebaut werden weil IM4P2P terminiert.

**StartSearch():** Eine Suche nach einer als Parameter übergebenen Zeichenkette soll gestartet werden.

**StartDownload():** Ein Download der durch einen Zeiger auf ein `P2PResult` Objekt identifiziert wird soll gestartet werden.

**Share():** Eine Datei die durch ein `P2PFile` Objekt näher beschrieben wird soll freigegeben werden.

**UnShare():** Eine freigegebene Datei soll von der Freigabe ausgenommen werden. Als Parameter dient der Dateiname.

Diese Implementierung der konkreten Peer-to-Peer Anwendungsanbindung geschieht für den Overnet Command Line Client in der Klasse `OvernetCLClient`.

Die steuernde Verantwortung für die Peer-to-Peer Anwendungsschnittstelle trägt innerhalb des IM4P2P Frameworks die Adapterklasse `P2PAdapter`. Diese Klasse besitzt das instanziierte Objekt der konkreten Anwendungsschnittstelle vom Obertyp `P2PInterface` und sorgt dafür, dass dieses rechtzeitig initialisiert und vor Beendigung von IM4P2P wieder zerstört wird. Darüberhinaus dient `P2PAdapter` wie bereits erwähnt mit den folgenden überladbaren Ereignisfunktionen für die Schnittstellenimplementierung als Gegenstelle für innerhalb der Nachrichtenschleife aufgelaufene Nachrichten:

**OnP2PSearchResult():** Hierüber werden durch ein `P2PResult` Objekt beschrieben Suchergebnisse der Adapterklasse mitgeteilt.

**OnP2PDownloadFinished():** Hiermit wird ein beendeter Download in Form eines `P2PFile` Objekts gemeldet.

Die Gründe `P2PAdapter` als eine abstrakte Klasse zu entwerfen werden erst später ersichtlich werden, soweit sei nur einmal angemerkt, dass die Ereignisfunktionen als „pure virtual“ Funktionen deklariert wurden.

[2] Ob die Verbindung zur Peer-to-Peer Anwendung über Interprozess-Kommunikation z. B. mit Sockets geschieht, oder indem der IM4P2P Quellcode mit dem Quellcode der Anwendung integriert gelinkt wird bleibt der konkreten Implementierung überlassen.

### 4.2.3 Die konkrete Peer-to-Peer Anwendungsschnittstelle: Overnet Command Line Client

Als konkrete Peer-to-Peer Anwendung wurde in Abschnitt 3.1 der Overnet Command Line Client als idealer Kandidat identifiziert. Für die Implementierung wurde die Linux Version 0.51.2 verwendet.

Der Overnet Command Line Client lässt sich über ein GUI Protokoll fernsteuern. Der Client selbst ist zwar Closed Source, allerdings existiert mit ed2k-gtk-gui [edGU] ein quelloffenes Projekt das eine GUI für diesen implementiert. Als Teil dieses Projekts ist in der Datei *gui_core_protocol. h* [edGP] die grundlegende Arbeitsweise des GUI Protokolls dokumentiert.
Kleinere Unklarheiten, die diese Dokumentation offen liess, konnten durch das Mithören einer Sitzung zwischen ed2k-gtk-gui und Command Line Client mittels Ethereal [Ethe] geklärt werden.

Das GUI Protokoll nutzt TCP zur Datenübertragung und definiert ein eigenes Nachrichtenformat (siehe Tabelle 4.1), das für die Kommunikation in beide Richtungen verwendet wird. Über dieses Protokoll können sämtliche Dienstprimitive des Command Line Client wie Suchen, Download starten, Status abfragen, etc. angestossen und Rückmeldungen wie Suchergebnisse entgegengenommen werden.

| | | |
|---|---|---|
| Header: | 1 Byte | Magic Byte $E3h$ |
| | 4 Byte | Länge des Payload |
| Payload: | 1 Byte | Message ID |
| | n Byte | Optionale Daten (abhängig von Message ID) |

Tabelle 4.1: Das Overnet GUI Protokoll Nachrichtenformat

Die Darstellung der Nachrichten in diesem Wire-Protocol wird in `OCLCMsg` und den davon abgeleiteten Klassen `OCLCMsgRecv` sowie `OCLCMsgSend` vorgenommen.

Die eigentliche Schnittstellenlogik, welche die Nachrichten über dieser Klassen in der korrekten Abfolge versendet und empfängt ist in `OvernetCLClient` implementiert. Gesteuert wird sie dabei durch Aufrufe an die generische Schnittstelle und durch vom Overnet Command Line Client generierte Ereignisse.

### 4.2.4 Die generische grafische Instant Messenger Oberflächenschnittstelle

Als Ziel der generischen Instant Messenger Oberflächenschnittstelle soll hier, parallel zur Zielsetzung bei der vorangegangenen Definition der generischen Peer-to-Peer Anwendungsschnittstelle, wiederum das Verbergen der konkreten Oberflächenimplementierung hinter der generischen Schnittstellendefiniton (sieher Abschnitt 3.4) gelten. Ein weiteres Ziel ist die effiziente softwaretechnische Integration ins IM4P2P Framwork.

Für die Kommunikation zwischen IM4P2P Framework und konkreter Schnittstellenimplementierung wird wiederum eine eigene Daten-Abstraktion geschaffen: Die

Klasse `GUIContact` repräsentiert einen generischen Kontakt auf der Buddy-Liste mit seinem anzuzeigenden Spitznamen, seinem Online Status und dem aktuellen Statustext.

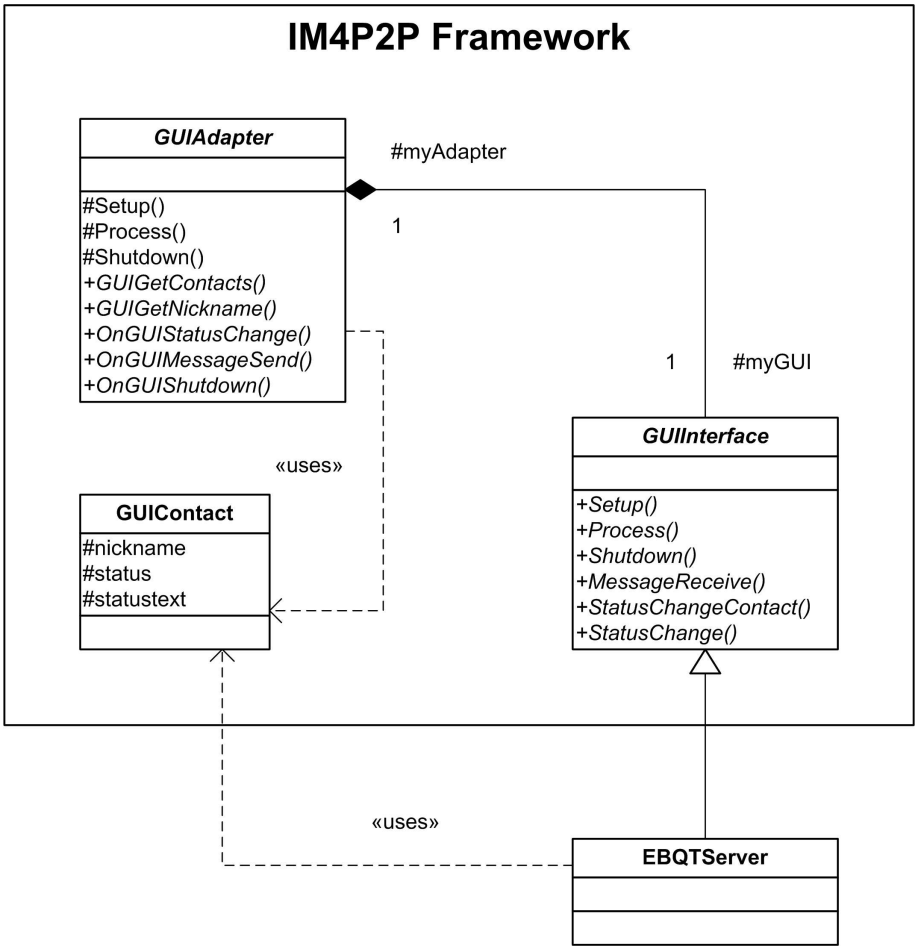


Abbildung 4.2: UML Klassendiagramm der Integration der IM Oberfläche in IM4P2P

Die generische Oberflächenschnittstelle wird innerhalb des IM4P2P Frameworks durch die abstrakte Klasse `GUIInterface` repräsentiert. Die konkrete Implementierung der Oberflächensschnittstelle muß dazu die folgenden von `GUIInterface` geerbten Funktionen implementieren:

**Setup():** Veranlasst die Initialisierung der grafischen Oberfläche. Wird eine externe Anwendung zur Darstellung benutzt muß diese dann in diesem Schritt kontaktiert werden.

**Process():** Die Nachrichtenschleife soll abgearbeitet werden: Vom Anwender über die grafische Oberfläche generierte Ereignisse werden verarbeitet und über die entsprechenden Ereignisfunktionen der Adapterklasse mitgeteilt (s. nächster Absatz).

**Shutdown():** IM4P2P terminiert demnächst und benachrichtigt hiermit die GUI.

**MessageReceive():** Eine Nachricht für den von einem `GUIContact` Objekt repräsetierten Kontakt soll angezeigt werden.

**StatusChangeContact():** Der Status eines von einem `GUIContact` Objekt repräsentierten Kontakts hat sich geändert.

**StatusChange():** Der Status des Benutzers hat sich geändert.

Die Implementierung der konkreten Oberflächenschnittstelle geschieht für die EbQT GUI in der Klasse `EBQTServer`.

Analog zur Integration der Peer-to-Peer Anwedungsschnittstelle übernimmt auch bei der IM Oberflächenschnittstelle eine Adapterklasse die zentrale Verantwortung für die Ansteuerung der konkreten Implementierung der Oberflächenschnittstelle. Die Klasse `GUIAdapter` besitzt die instanziierte konkrete Oberflächenimplementierung, sieht allerdings als Schnittstelle wie gehabt nur die Oberklasse aus dem IM4P2P Framework vom Typ `GUIInterface`. `GUIAdapter` ist verantwortlich für die Initialisierung und das ordnungsgemäße Zerstören des Objekts bei Programmende. Außerdem bietet die Adapterklasse der konkreten Implementierung Ereignisfunktionen als Gegenstelle für die Meldung von während der Nachrichtenschleife abgearbeiteten Ereignissen an:

**OnGUIStatusChange():** Hierüber wird der Adapterklasse eine Änderung des Online Status und/oder des Online Status Text durch den Anwender mitgeteilt.

**OnGUIMessageSend():** Hiermit wird die Adapterklasse darüber informiert, dass der Anwender eine Nachricht an einen bestimmten durch ein `GUIContact` Objekt beschriebenen Kontakt auf der Buddy Liste abgeschickt hat.

**OnGUIShutdown():** Die Anwendung IM4P2P wird vom Anwender über die IM Oberfläche beendet.

Auch hier sei wiederum mit dem Verweis auf eine spätere Erklärung angemerkt, dass die Klasse `GUIAdapter` aufgrund der virtuell deklarierten und implementierungslosen Ereignisfunktionen eine abstrakte Klasse ist.
Die beiden anderen öffentlichen Funktionen von `GUIAdapter`, nämlich **GUIGetContacts()** und **GUIGetNickname()** sind keine Ereignisfunktionen, werden aber ebenfalls von der konkreten Implementierung der Oberflächenschnittstelle benutzt um eine Liste von `GUIContact` Objekten die die Kontakte auf der Buddy Liste repräsentieren und den anzuzeigenden Spitznamen des Anwenders vom IM4P2P Framework zu erfahren.

### 4.2.5 Die konkrete grafische Instant Messenger Oberflächenschnittstelle: EbQT

In der Implementierung fand die aktuelle Version Alpha 11 von EB-lite Verwendung. EbQT ist die Oberflächen-Komponente dieser modular konzipierten Instant-Messaging Anwendung, deren eigentliche Instant Messaging Core namens EB-lite normalerweise einen Server bereit stellt an dem sich die GUI anmeldet und der für die gesamte Netzwerkkommunikation mit dem Instant Messaging Netz verantwortlich ist.
IM4P2P übernimmt nun die Rolle genau dieses Instant Messaging Cores.

Um die Komplexität im Rahmen zu halten beschränkt sich die Implementierung auf die grundlegenden Fähigkeiten des IM Core Servers und ignoriert erweiterte Fähigkeiten wie mehrere GUIs für denselben Account, Accountmanagement und Datenhaltung die für IM4P2P nicht von Bedeutung sind.

Das verwendete GUI Protokoll ist in [EBlG] ausführlich dokumentiert. Es basiert auf TCP und definert Befehle die aus mehreren Paramtern bestehen, wobei der erste Parameter meist den Befehl selbst kennzeichnet (siehe Tabelle 4.2).

| | | |
|---|---|---|
| Header: | 1 Byte | Anzahl Parameter: $n$ |
| 1. Parameter: | 2 Byte<br>$l_1$ Byte | Länge der Zeichenkette: $l_1$ (MSB zuerst)<br>Zeichenkette (ohne Null-Terminator) |
| ... | ... | ... |
| $n$. Parameter: | 2 Byte<br>$l_n$ Byte | Länge der Zeichenkette: $l_n$ (MSB zuerst)<br>Zeichenkette (ohne Null-Terminator) |

Tabelle 4.2: Das EB-lite GUI Protokoll Nachrichtenformat

Die Nachrichten werden von der Klasse `EBQTMsg` und den davon abgeleiteten `EBQTMsgSend` und `EBQTMsgRecv` kodiert und verschickt bzw. empfangen und dekodiert.

Die Implementierung des Instant Messaging Core Servers, der die komplette Schnittstellenlogik enthält und die Nachrichten verschickt und interpretiert, geschieht in der Klasse `EBQTServer`.

## 4.3 IM4P2P

Die Implementierung von IM4P2P baut auf dem bislang definierten Framework mit seinen Schnittstellen auf. Aufgabe der Anwendung IM4P2P ist es, die von den konkreten Schnittstellenimplementierungen generierten Ereignisse zu verarbeiten und selbst wiederum über die Schnittstelle Aktivitäten anzuregen um letztlich die Instant Messaging Fähigkeit über Peer-to-Peer Netze zu realisieren.

### 4.3.1 Integration der generischen Schnittstellen in IM4P2P

Nachdem die generische Anbindung sowohl der konkreten Peer-to-Peer Anwendungsschnittstelle als auch der konkreten grafischen Instant Messenger Oberflächenschnittstelle realisiert wurde, stellt sich nun die Frage wie sich diese einzelnen generischen Schnittstellen geschickt im IM4P2P Framework zusammenführen lassen.

An dieser Stelle klärt sich dann auch das Geheimnis um die abstrakte Natur der Adapterklassen. Es war nicht beabsichtigt einen der beiden Klassen `P2PAdapter` oder `GUIAdapter` zu instanziieren. Vielmehr dienen die Adapter lediglich als unveränderliche Gegenstelle innerhalb des IM4P2P Frameworks für die konkrete Schnittstellenimplementierungen ohne dabei zu viele Implementierungsdetails nach außen hin festlegen zu müssen.
Die eigentliche Implementierung der Umsetzung zwischen Ereignissen der Instant Messaging Oberfläche und Ereignissen der Peer-to-Peer Anwendung geschieht in einer einzelnen Klasse, die durch Mehrfachvererbung die Funktion beider Adapter

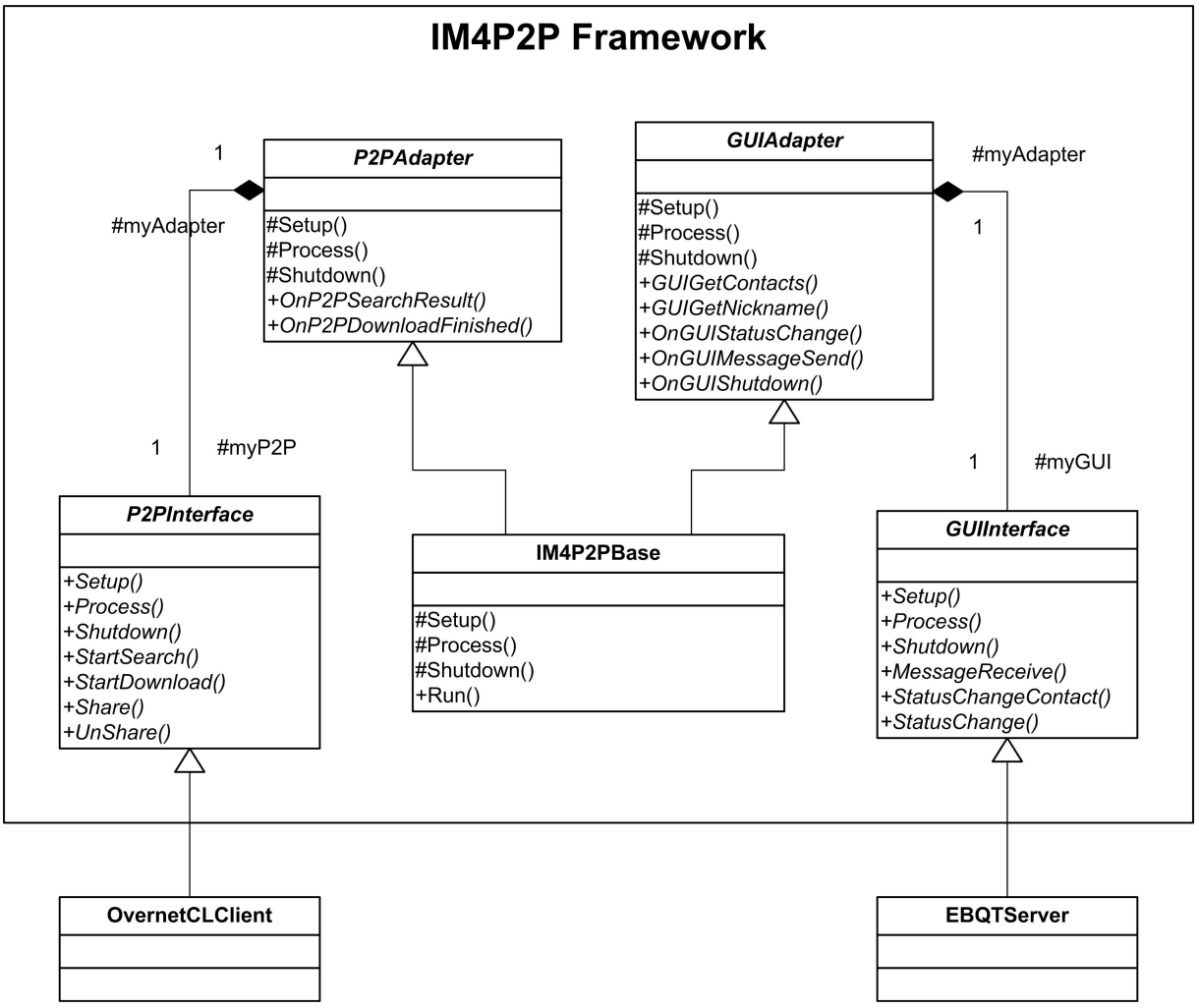


Abbildung 4.3: UML Klassendiagramm der Integration der generischen Schnittstellen im IM4P2P Framework

übernimmt (siehe Abbildung 4.3): `IM4P2PBase` ist sowohl ein `P2PAdapter` als auch ein `GUIAdapter`.

Die Klasse `IM4P2PBase` bekommt über ihren Konstruktor die instanziierten konkreten Schnittstellenimplementierungen übergeben, erwartet aber lediglich Objekte vom Typ `GUIInterface` bzw. `P2PInterface` und gewährleistet dadurch die generische Sichtweise auf die Schnittstellen. Der Austausch einer dieser generischen Komponenten kann dann ganz einfach innerhalb der Datei *main.cpp* erfolgen, indem bei der Initialisierung des `IM4P2PBase` Objekts andere konkrete Schnittstellenobjekte instanziiert werden:

```
int main()
{
   // ...
   IM4P2PBase im4p2p(new EBQTServer, new OvernetCLClient);
   im4p2p.Run();
   // ...
}
```

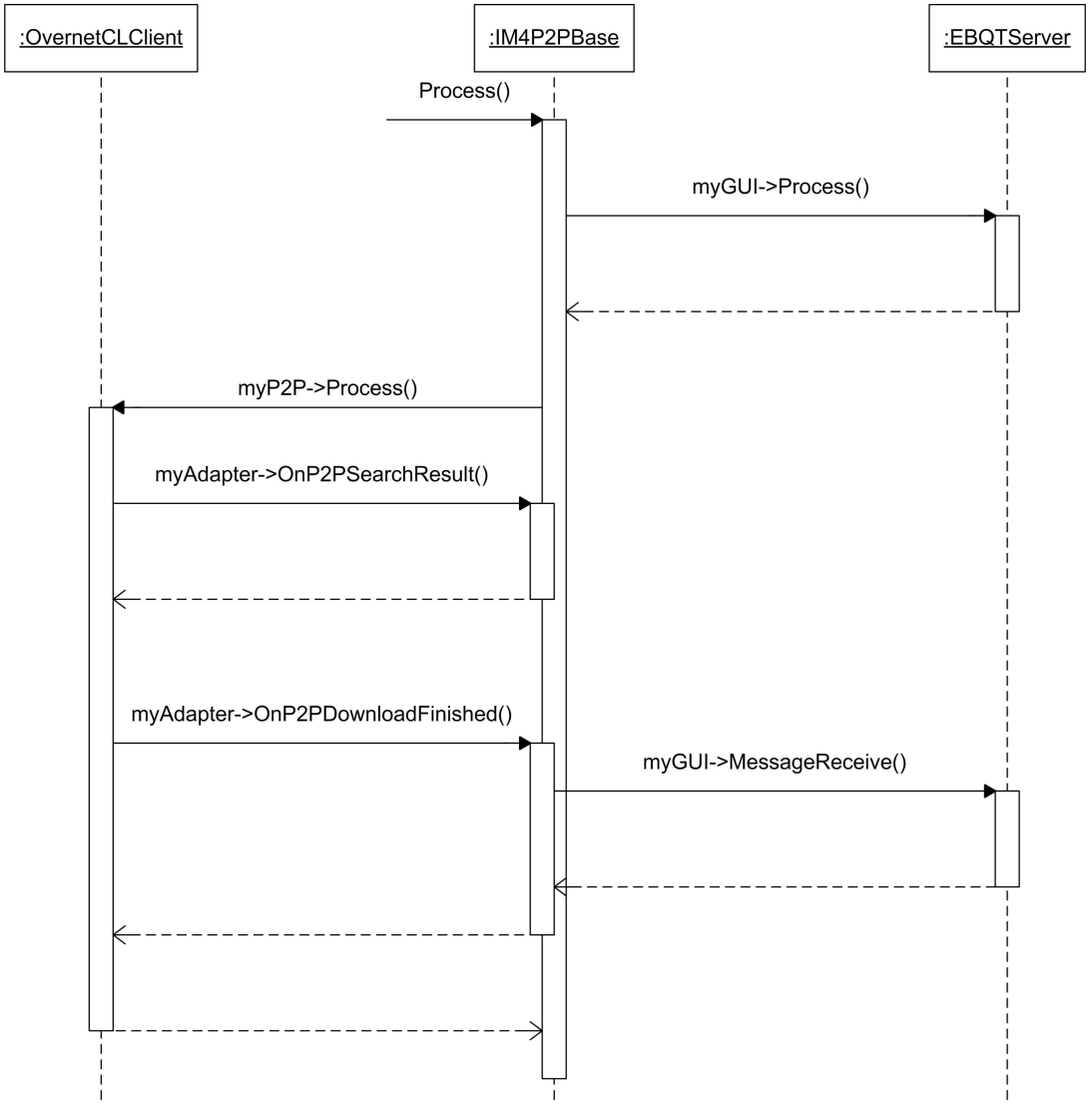


Abbildung 4.4: UML Sequenzdiagramm: Zusammenspiel von `IM4P2PBase` mit den konkreten Schnittstellenimplementierungen

Zur Verdeutlichung der Zusammenarbeit der konkreten Schnittstellenimplementierungen mit `IM4P2PBase` soll nun in Abbildung 4.4 das Verhalten während einer Iteration der Nachrichtenschleife betrachtet werden.
Nachdem `IM4P2PBase` die eigene Nachrichtenschleife in `IM4P2PBase::Process()` betritt, startet es die Nachrichtenauswertung der angeschlossenen grafischen Oberfläche mit `myGUI->Process()`. In diesem Beispiel sind bei der konkreten Implementierung der Oberflächenschnittstelle seit der letzten Anfrage keine neuen Ereignisse aufgelaufen.
Dann wird die Nachrichtenauswertung der Peer-to-Peer Anwendung durch den Aufruf von `myP2P->Process()` angestoßen.
Hier sind zur Verdeutlichung zwei Ereignisse angenommen. Zum ersten hat die Schnittstelle von der Anwendung ein weiteres Suchergebnis erhalten (das von einer älteren Suchanfrage stammt) und meldet dies ihrem zugewiesenen `P2PAdapter` dessen Rolle ja die davon abgeleitete Klasse `IM4P2PBase` einnimmt.
In diesem Fall scheint `IM4P2PBase` mit dem Suchresultat nicht viel anzufangen zu können z. B. weil es einen veralteten Zeitstempel trägt, und veranlasst daraufhin

keinen neuen Start eines Downloads.
Das zweite Ereignis ist das Eintreffen einer heruntergeladenen Datei. Nach der Auswertung der Datei (die in diesem Diagramm aus Übersichtsgründen weggelassen wurde) scheint `IM4P2PBase` eine neue Nachricht für einen Kontakt auf der Kontaktliste erhalten zu haben und benachrichtigt daher über `myGUI->MessageReceive()` die grafische Oberflächenschnittstelle.

Die Zusammenführung der beiden Adapterfunktionalitäten in einer gemeinsamen Klasse erweist sich als äußerst praktisch:
Bei der Betrachtung der Abläufe während der IM4P2P Nachrichtenschleife fällt auf wie stark ereignisgesteuert der Kontrollfluß voranschreitet: Jedes Ereignis der Peer-to-Peer Schnittstelle kann weitere Aktionen für die GUI Schnittstelle oder für die Peer-to-Peer Schnittstelle selbst nach sich ziehen, und diese Aktionen lösen meist weitere Ereignisse zu einem spätere Zeitpunkt aus.
Daher ist es sinnvoll die gegenseitigen Abhängigkeiten von ein- und ausgehenden Benachrichtigungen an die angeschlossenen Schnittstellen innerhalb einer einzigen Klasse zu kapseln und dadurch einen wesentlich übersichtlicheren Quellcode zu erhalten.

### 4.3.2 Die Kontaktliste

Die Kontaktliste wird in IM4P2P durch eine Liste von `IM4P2PContact` Objekten repräsentiert. Die Klasse `IM4P2PContact` erbt dazu alle Datenelemente eines generischen Kontakts die in `GUIContact` definiert sind, also Online Status, Statustext und Spitznamen. Darüberhinaus verfügt `IM4P2PContact` über weitere Datenelemente, nämlich ein `P2PFile` Objekt das die jeweilige Kontextdatei repräsentiert, einen Zeitstempel, der den Zeitpunkt der letzten Nachricht hält, um über einen Timeout feststellen zu können wann ein Kontakt offline geht, sowie die 128 Bit Kontakt ID die den Kontakt eindeutig identifiziert.

### 4.3.3 Die Kontextdatei

Die Klasse `IM4P2PContextFile` implementiert die im Entwurf (siehe Abschnitt 3.5.1.2) eingeführte Kontextdatei. Ziel der Kontextdatei ist die Erstellung eines Kommunikationskontextes zwischen jeweils zwei kommunizierenden Anwendungen durch die getauschten Dateien. Dabei enthält dieser Kontext alle gesendeten Nachrichten sowie den Onlinestatus.
Um die Kontextdatei im Laufe einer Sitzung bei jeder neuen Nachricht nicht ständig weiter anwachsen zu lassen wird innerhalb des Kontextes ein gewisses Sendefenster erstellt, und nur die noch nicht quittierten Nachrichten verbleiben innerhalb der Kontextdatei. Da Clients auch kurzfristig unerkannt aus dem Netz verschwinden können und kurz darauf wieder auftauchen, wird diese Quittierung über Zeitstempel realisiert, da diese gegenüber Zufallszahlen den Vorteil bieten immer nur in aufsteigender Reihenfolge aufzutreten.

Die Kontextdatei selbst enthält zwei Sorten Daten, einerseits einige Attribute wie Online Status, Statustext und Quittierungszeitstempel und andererseits eine Liste der noch nicht quittierten Nachrichten.
Diese Daten werden in regelmäßigen Abständen in die freizugebende Datei gespeichert bevor diese über die Peer-to-Peer Anwendung veröffentlicht wird.

Die Kontextdatei hat folgendes Format:

```
ATTRIBUT1:WERT1<LF>
ATTRIBUT2:WERT2<LF>
ATTRIBUT3:WERT3<LF>
<LF>
ZEITSTEMPEL1:<LÄNGE(32Bit)>NACHRICHT1
ZEITSTEMPEL2:<LÄNGE(32Bit)>NACHRICHT2
```

Dabei steht `<LF>` für ein Line Feed Zeichen (ASCII $0A_{hex}$). Das doppelte Line Feed markiert das Ende des Attribut-Teils, der darauf folgende Nachrichtenteil kann leer sein. Da innerhalb von Nachrichten selbst wiederum ein Line Feed vorkommen kann, muss die Länge der Nachricht über den vorangestellten 32 Bit Wert `<LÄNGE(32Bit)>` bekanntgegeben werden.

Eine Verschlüsselung und Authentifizierung von Nachrichten ist bislang in IM4P2P nicht implementiert. Die Kontextdatei Klasse `IM4P2PContextFile` wäre allerdings der richtige Platz für kryptografische Ergänzungen.

### 4.3.4 Implementierung von Sonderfällen

Trotz aller Bemühung um Generizität der Anwendung, lassen sich Sonderfälle bei bestimmten Peer-to-Peer Anwendungen, die bei allgemeinen Anwendungen keine Berücksichtigung finden müssen, nicht gänzlich vermeiden.
Um diesen Sonderfällen Rechnung zu tragen, muss zwangsläufig die Ereignisbehandlung in der Klasse `IM4P2PBase` geändert werden. Damit von diesen Änderungen aber das IM4P2P Framework nicht verwässert wird empfiehlt es sich diese Änderungen in einer von `IM4P2PBase` abgeleiteten Klasse außerhalb des Frameworks vorzunehemen. Im folgenden soll dies am Beispiel der bereits angesprochenen Sonderfälle im Overnet Peer-to-Peer Netz gezeigt werden.

#### Anpassung an den Overnet Command Line Client

Das Phänomen des Überlauf des Namensraums wie es in Abschnitt 3.5.5 dokumentiert ist erzwingt eine andere Namensgebung von Dateien als in IM4P2PBase vorgesehen. Diese Änderungen werden in der von `IM4P2PBase` abgeleiteten Klasse `IM4P2P_Alpha` vorgenommen. Dazu werden alle Methoden überladen, die mit der Namensgebung von Dateien und Suchanfragen betraut sind und um die Timefix Zeichenkette ergänzt.

# 5. Test und Betrieb

In diesem Kapitel wird zunächst ein Blick auf die Konfigurationsmöglichkeiten von IM4P2P und den beiden externen Anwendungen EbQT und dem Overnet Command Line Client geworfen.
Anschließend wird die Implementierung unter variierten Konfigurationsoptionen verschiedenen Testläufen unterzogen und die Ergebnisse dieser Tests im Hinblick auf das zugrunde liegende Peer-to-Peer Netz bewertet.

## 5.1 Konfiguration

Im folgenden werden nun die nötigen Schritte zur Konfiguration der einzelnen Komponenten erläutert um ein reibungsloses Zusammenspiel mit der Anwendung IM4P2P zu ermöglichen. Danach wird IM4P2P selbst eingerichtet und schließlich der Startvorgang erläutert.

### 5.1.1 Konfiguration von EbQT

EbQT lässt sich über das übliche `configure` und `make` im Unterverzeichnis `ebqt/` des Quelltextpakets von EB-Lite kompilieren. Eine Installation über `make install` ist nicht unbedingt nötig. EbQT lässt sich aus einem beliebigen Verzeichnis starten.

### 5.1.2 Konfiguration des Overnet Command Line Client

Die benötigten Dateien für den Overnet Command Line Client sind als Archiv erhältlich und sollten in ein eigenes Verzeichnis entpackt werden, da der Client im Betrieb unterhalb seines Arbeitsverzeichnisses eigene Konfigurations- und andere temporäre Dateien ablegt.
Nachdem der Client aus seinem Arbeitsverzeichnis heraus gestartet wurde, sollten zuerst einmal durch die Eingabe folgender Befehle einige Konfigurationsparameter angepasst werden:

**a <directory name>** Fügt ein freigegebenes Verzeichnis hinzu. Mindestens eins wird für die Konfiguration von IM4P2P benötigt.

**pass <name> <password>** Setzt einen Benutzer und ein Passwort für den Fernsteuerungszugang.

**aport <port>** Setzt den TCP Port über den der Fernsteuerungszugang erfolgt.

Je nach Situation sind auch einige der weiteren Befehle von Nutzen:

**g** : Dient zur Abfrage des Status. Wichtig hierbei ist, ob nach einer kurzen Startverzögerung der Status `(open)` oder zumindest `(firewalled)` angezeigt wird, also eine Verbindung ins Overnet Netzwerk hergestellt werden konnte.

**port <port>** Setzt TCP Port für die Verbindung ins Overnet Netz.

**uport <port>** Setzt UDP Port für die Verbindung ins Overnet Netz.

**dumax <max download> <max upload>** Setzt Obergrenze für den Downstream und Upstream des Clients in KB/s.

**?** Zeigt Liste aller Befehle.

Damit der Overnet Command Line Client nach dem Start auf Befehle über den Fernsteuerzugang hört, muss er mit dem Kommandozeilenparameter `-g` gestartet werden.

### 5.1.3 Konfiguration von IM4P2P

Die Konfiguration von IM4P2P geschieht über die Konfigurationsdatei `im4p2p.conf` im Arbeitsverzeichnis der Anwendung. Darin sind folgende Variablen definiert:

**EBQT_SERVER_PORT** Der Port auf dem der Core Server für EbQT laufen soll.

**EBQT_COOKIE** Eine acht Zeichen lange beliebige Zeichenkette, die normalerweise keiner Änderung bedarf. Das Cookie wird beim Start des EbQT Clients benötigt und dient als Minimal-Challenge zur Authentifizierung am Server.

**OVERNET_CLIENT_HOME** Das Arbeitsverzeichnis des Overnet Command Line Client.

**OVERNET_CLIENT_COMMAND** Der Name der ausführbaren Datei (hängt von Version ab).

**OVERNET_PORT** Der TCP Port der für den Fernsteuerungszugriff eingestellt wurde.

**OVERNET_HOST** Der Rechner auf dem der Overnet Command Line Client läuft.

**OVERNET_ADMIN_LOGIN** Der für den Fernsteuerungszugriff eingerichtete Benutzername.

**OVERNET_ADMIN_PASS** Das für den Fernsteuerungszugriff eingerichtete Benutzerpasswort.

**OVERNET_DIR_INCOMING** Das Verzeichnis in dem heruntergeladene Dateien vom Client abgelegt werden, normalerweise OVERNET_CLIENT_HOME/incoming/.

**OVERNET_DIR_SHARE** Ein Verzeichnis in dem Dateien vom Client freigegeben werden können.

**OVERNET_DIR_TEMP** Das Verzeichnis in dem temporäre Dateien vom Client abgelegt werden, normalerweise OVERNET_CLIENT_HOME/temp/.

**IM4P2P_INTERVAL_SEARCH** Die Zeitspanne die zwischen zwei Suchanfragen gewartet werden muß.

**IM4P2P_INTERVAL_PUBLISH** Die Zeitspanne die mindestens gewartet wird bis eine Datei erneut veröffentlicht wird.

**IM4P2P_INTERVAL_OUTDATED** Die Zeitspanne nach der eine Datei als veraltet gilt, und damit nicht mehr betrachtet wird. Spätestens nach einem halben OUTDATED Intervall wird eine Datei erneut veröffentlicht, daher sollte IM4P2P_INTERVAL_OUTDATED > 2∗IM4P2P_INTERVAL_PUBLISH gelten.

**IM4P2P_CACHE_GRANULARITY** Die zeitliche Auflösung des Caches in Sekunden bestimmt aus wie vielen cache-baren Nachrichten potentiell ausgewählt wird.
Sollte auf allen beteiligten Clients die miteinander kommunizieren gleich sein.[1]

**IM4P2P_CACHE_SIZE** Die Größe des Caches in Anzahl Dateien ist lediglich ein Richtwert. Die tatsächliche Größe des Caches kann darüber liegen wenn mehrere cache-würdige Dateien gleichzeitig eintreffen.

**IM4P2PALPHA_TIMEFIX_GRANULARITY** Bestimmt die Zeitspanne innerhalb derer eine einizigartige Timefix Zeichenkette Verwendung finden soll. Nach Ablauf dieser Zeitspanne muss eine neue Timefix Zeichenkette generiert werden.
Sollte auf allen beteiligten Clients die miteinander kommunizieren gleich sein.

Aus den für den Overnet Command Line Client angegebenen Verzeichnissen (OVERNET_DIR_*) sollte man vorher wichtige Dateien entfernen, da IM4P2P dort ungefragt neue Dateien anlegen und löschen wird.

Die Datei `identity.conf` enthält die Identitätsdaten des IM4P2P Clients und wird beim ersten Start von IM4P2P automatisch generiert, falls noch nicht vorhanden.
Die Datei `contacts.conf` enthält alle bekannten Kontakte mit zeilenweisen Einträgen im selben Format wie in `identity.conf` und kann von Hand editiert werden um die Kontaktliste anzupassen.

[1] Da diese Größe die Rundungsgrößen für verschmierte Zeitstempel angibt, ist es wichtig das alle Clients auf dieselben Werte runden und danach suchen können.

### 5.1.4 Starten von IM4P2P

Beim Start von IM4P2P werden zunächst die Konfigurationsdateien ausgewertet. Sollte vom Anwender noch keine Instanz des Overnet Command Line Client im Fernsteuerungsmodus gestartet worden sein, startet IM4P2P den Client als eigenen Kindprozess, der dann auch wieder mit der Terminierung von IM4P2P beendet wird. Darauf wird automatisch eine Verbindung mit dem Client hergestellt und gewartet bis dieser eine Verbindung zum Overnet Netz etabliert hat.
Dann startet der GUI Server und wartet auf eine eingehende Verbindung. Der Anwender sollte nun das GUI Frontend EbQT starten, und unter „Manually-launched eb-lite:“ nach der Angabe der GUI Serveradresse und eines Authcookies mit „Connect“ eine Verbindung zum von IM4P2P bereitgestellten GUI Server herstellen.
Im Normalfall, wenn EbQT und IM4P2P unter demselben Nutzerkonto ausgeführt werden, sind die korrekte GUI Serveradresse und das korrekte Authcookie bereits eingestellt, da der Server diese Daten in einer Konfigurationsdatei im Home Verzeichnis des Anwenders ablegt die EbQT beim Start ausliest. Ansonsten können diese Daten beim Start von IM4P2P der Konsolenausgabe entnommen werden.

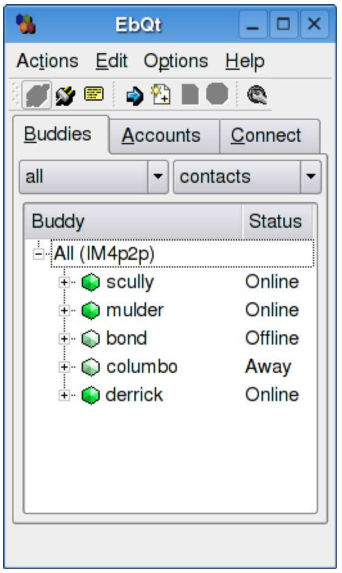


Abbildung 5.1: IM4P2P Kontaktliste

Nach der Herstellung der Verbindung wechselt EbQT selbstständig unter den Karteireiter „Buddies“ der die Kontaktliste darstellt. Über das dritte Icon von links unter der Menüleiste lassen sich Away-Status und Statusmeldungen setzen und wieder aufheben.


Abbildung 5.2: Online Status Dialog

Durch Anklicken eines Kontakts lässt sich ein seperates Textchat Fenster öffnen über das die Kommunikation mit dem gewünschten Kontakt erfolgen kann.

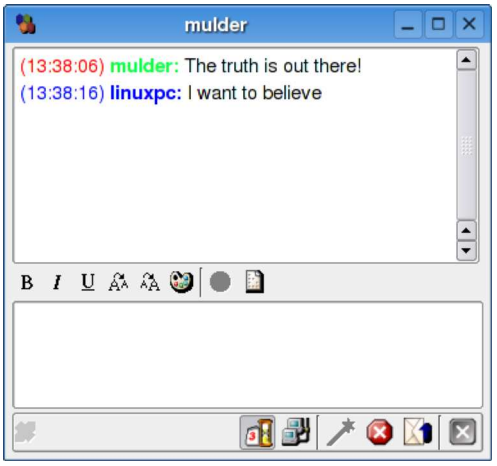


Abbildung 5.3: Textchat Fenster

IM4P2P sollte über das Menü der GUI über den Eintrag „Actions->Shutdown IM4p2p“ beendet werden, da dadurch ein sauberer Abbau aller Verbindungen gewährleistet wird.

## 5.2 IM4P2P Tests

Nachdem IM4P2P jetzt konfiguriert ist, soll es im Betrieb einigen Tests unterzogen werden. Besonderes Augenmerk liegt dabei zunächst auf der Wahl günstiger Konfigurationen für die Such- und Publikations-Intervalle. Nach einer Bewertung der Forward Cache Effektivität folgt darauf eine genauere Betrachtung der durch IM4P2P erzeugte Verkehrslast, sowohl lokal als auch in den Auswirkungen auf das gesamte Peer-to-Peer Netz.

### 5.2.1 Ping-Pong Tests

Der Ping-Pong Test dient zur Messung der Round Trip Time (RTT) von Nachrichten. Dazu sendet Client A eine Nachricht mit dem Inhalt „PING“ an Client B und dieser antwortet mit einer „PONG“ Nachricht. Die dabei verstrichene Zeit wird von Client A gestoppt und stellt die RTT dar.

Um den Vorgang zu automatisieren, wurde die zentrale Steuerungsklasse `IM4P2PBase` (bzw. die von ihr abgeleitete Klasse `IM4P2P_Alpha` welche die Anpassungen an den Overnet Client enthält) in einer spezialisierten Klasse namens `IM4P2P_PingPong` um automatisches Senden von Ping und Pong Nachrichten ergänzt.
Indem über die GUI eines Clients das Schlüsselwort „PINGPONG“ an einen anderen Client geschickt wird, lässt sich der Ping-Pong Test zwischen diesen beiden Clients starten. Im Laufe des Tests werden fünf mal hintereinander abwechselnd Ping und Pong Nachrichten zwischen den beiden Clients hin- und hergesendet. Die gemessene Transferzeit wird als Konsolenausgabe von IM4P2P angezeigt. Als Ergebnis eines solchen Testlaufs ergibt sich dann die über fünf Läufe gemittelte RTT zwischen zwei Clients.

Im Test werden 6 Clients auf 6 verschiedenen Rechnern verwendet, darunter befinden sich 5 innerhalb eines Subnetzes und ein weiterer in einem anderen Subnetz. Im Kommunikationsverhalten ließen sich keine meßbaren Unterschiede zwischen der Kommunikation von Clients innerhalb des gleichen Subnetzes und der Kommunikation zwischen Clients in verschiedenen Subnetzen feststellen. Das lässt sich damit erklären, das die Transferzeit bei den verwendeten, einige hundert Bytes großen Kontextdateien, vernachlässigbar klein ist und als größter Faktor der Suchaufwand im Overnet Netz in die Übertragungszeit eingeht. Diese Suche findet aber sowieso größtenteils durch Kommunikation mit vielen entfernten Rechnern statt, so dass die Suchzeiten unabhängig von der netztopologischen Nähe zwischen suchendem und veröffentlichendem Client ist.
Um weitere Effekte etwa durch nahe beieinander liegende Overnet Knoten IDs auszuschließen, wird als Round Trip Time das arithmetische Mittel aus jeweils sechs Ping-Pong Testläufen zwischen verschiedenen Clients gewählt. Es gehen also insgesamt 30 einzelne Round Trip Time Werte in den jeweiligen Mittelwert ein.
Forward Caching wird während der Ping-Pong Tests deaktiviert.

Der Einfluß der beiden durch die Parameter IM4P2P_INTERVAL_PUBLISH und IM4P2P_INTERVAL_SEARCH spezifizierten Intervallgrößen auf die Round Trip Time soll nun in einer ersten Betrachtung untersucht werden. Dazu werden beide Werte parallel auf den gleichen Wert gesetzt und in Schritten von 10 Sekunden variiert (siehe Abbildung 5.4).

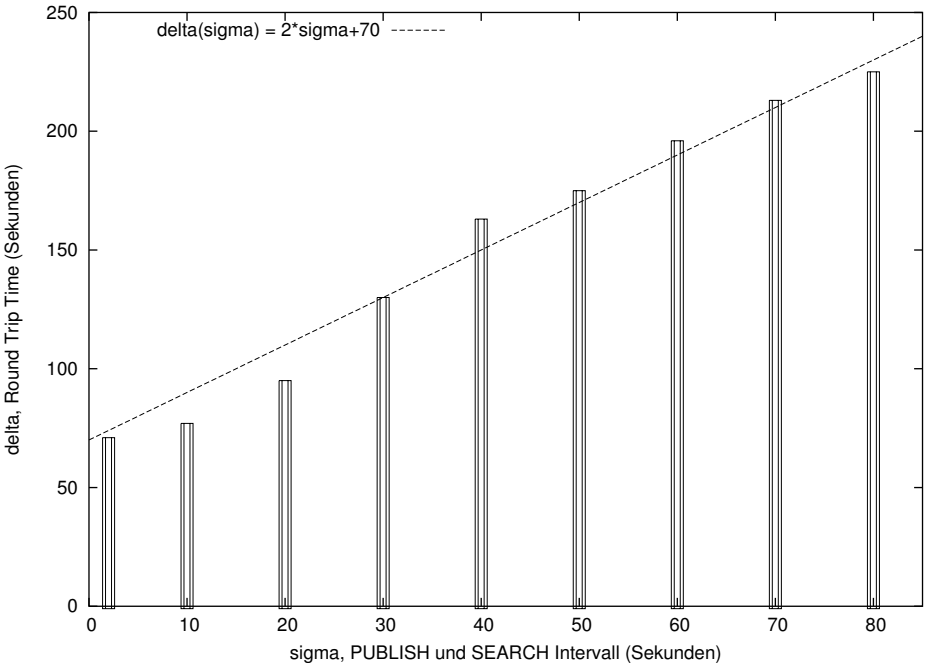


Abbildung 5.4: Mittlere Round Trip Times unter parallel variierten PUBLISH und SEARCH Intervallen.

Es zeigt sich, wie zu Erwarten war, dass mit zunehmender Intervalllänge auch die Round Trip Time steigt. Die RTT $\delta$ scheint dabei in einem linearen Zusammenhang mit den verwendeten Intervalllängen $\sigma$ zu stehen. Um diese Gesetzmäßigkeit zu beschreiben, lässt sich optisch eine Funktion (wie eingezeichnet) an die Testergebnisse an-„fitten“:

$$\delta(\sigma) = 2 * \sigma + 70$$

Die Abhängigkeit $2 * \sigma$ erscheint sinnvoll, angesichts der Tatsache, dass in die Round Trip Time die Wartezeiten auf beiden Seiten der Kommunikationsstrecke eingehen: Zunächst muss Client A im Mittel $\frac{1}{2}$ PUBLISH abwarten bis er die Datei mit seinem PING veröffentlichen kann. Dann dauert es erneut im Mittel $\frac{1}{2}$ SEARCH bis Client B eine Suchanfrage abgibt. Für das PONG kommen dann erneut die selben Wartezeiten zustande, was sich dann auf 2 PUBLISH bzw. 2 SEARCH summiert.
Verlängert man die Gerade nach links bis auf eine theoretische PUBLISH und SEARCH Intervalllänge von 0, bleibt der Offset von +70 Sekunden, der als untere Schranke für die RTT angesehen werden kann. Im Mittel ist also mit dem Polling Verfahren eine RTT unterhalb von 70 Sekunden nicht erzielbar, wohlgemerkt traten in Einzelfällen auch RTTs unter 40 Sekunden auf.
Der Offset von 70 Sekunden lässt auch Rückschlüsse auf das zugrunde liegende Overnet Netzwerk zu: So scheint die effektive Publikationszeit (also die Zeit von der Veröffentlichung, für die Suche und bis zum Beginn[2] des ersten Downloads) im Mittel etwa 35 Sekunden (halbe Round Trip Time für $\sigma = 0$ Sekunden) zu betragen.

Nachdem nun IM4P2P_INTERVAL_PUBLISH und IM4P2P_INTERVAL_SEARCH in Kombination betrachtet wurden, stellt sich die Frage welches als einzelnes den größeren Einfluß auf die Round Trip Time hat. Um dies festzustellen, wird jeweils eines der beiden Intervalle fest auf 30 Sekunden gestellt und das jeweils andere in den bekannten 10 Sekunden Schritten variiert (siehe Abbildung 5.5).

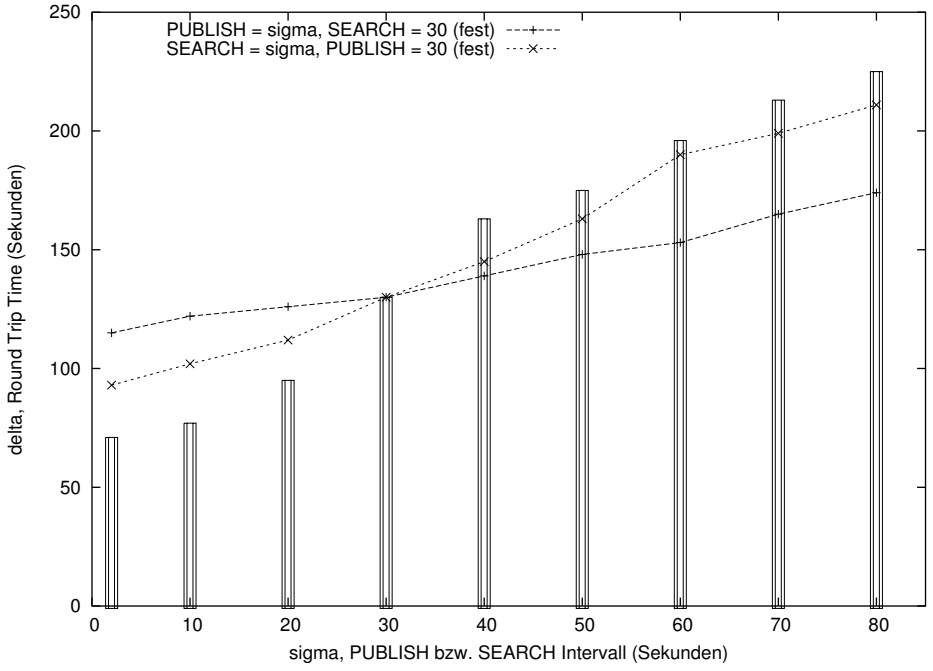


Abbildung 5.5: Verhalten der Round Trip Times unter Änderung jeweils nur eines Parameters (PUBLISH oder SEARCH). Zur Verdeutlichung sind die Werte aus dem ersten Ping-Pong Test erneut als Balken eingezeichnet.

Der markanteste Punkt im Ergebnisschaubild liegt bei $\sigma = 30$ Sekunden. Dort treffen sich die beiden Round Trip Times logischerweise, da dort in beiden Fällen sowohl PUBLISH als auch SEARCH auf 30 Sekunden stehen.

[2]Die geringe Größe der Kontextdatei erlaubt das Gleichsetzen der Zeitpunkte Downloadbeginn und -ende

Wie zu erwarten war, verhalten sich die RTTs links von 30 Sekunden etwas schlechter als im ersten Versuch und rechts von 30 Sekunden etwas besser, denn im Vergleich zum ersten Versuch wird ja nur eines der beiden Intervalle variiert, während das andere fest auf 30 Sekunden eingestellt bleibt.
Die RTT Werte im Versuch mit variablem PUBLISH Intervall verlaufen deutlich flacher als die RTTs unter variiertem SEARCH Intervall. So fällt die Linie mit variiertem SEARCH Intervall links von 30 Sekunden deutlich stärker ab und steigt rechts von 30 Sekunden auch sehr viel stärker an als die andere. Daraus lässt sich schließen, dass das SEARCH Intervall einen größeren Einfluss auf die Round Trip Time ausübt als das PUBLISH Intervall.
Der geringere Einfluß des PUBLISH Intervalls auf die RTT dürfte größtenteils der Entwurfs-Optimierung zuzurechnen sein, die Kontextdatei nur dann sofort nach Ablauf des PUBLISH Intervalls erneut zu veröffentlichen wenn auch wirklich neue Nachrichten vorhanden sind, und ansonsten abzuwarten bis das OUTDATED Intervall (das in diesem Test bei 300 Sekunden lag) abläuft. Dadurch wird die mittlere Wartezeit bis zur Wiederveröffentlichung erheblich unter den Erwartungswert von $\frac{1}{2}$ PUBLISH gedrückt.
Der sehr flache Verlauf der RTTs unter variierten PUBLISH Intervallen insbesondere zwischen 10 und 40 Sekunden deutet darüberhinaus an wie wenig Nutzen es hat, das PUBLISH Intervall auf kleinere Werte als das SEARCH Intervall zu setzen.

Für die Wahl geeigneter PUBLISH und SEARCH Intervalle zeigen die Tests, dass je nach gewünschter mittlerer Antwortzeit ein geeigneter Wert für SEARCH zu finden ist. Wenngleich mit dem Overnet Netz als Basis für das Polling Verfahren eine gewisse untere Schranke bestehen bleibt, die im Mittel nicht unterschritten wird. Das PUBLISH Intervall sollte daraufhin gleichgroß oder etwas größer gewählt werden um eine sinnvolle Kosten/Nutzen Relation zu erhalten.

### 5.2.2 Forward Cache Effektivität

Um den Nutzen des Forward Cache Verfahrens zu ergründen wird IM4P2P derart angepasst, dass es Dateien im Cache markiert, um empfängerseitig feststellen zu können ob die Datei direkt vom Sender übertragen wurde oder den (eventuell schnelleren) Umweg über einen oder mehrere Forward Caches gemacht hat.

Als Testumgebung dient abermals der aus dem Ping-Pong Test bekannte Aufbau mit 6 Clients. Diesmal ist allerdings auf allen IM4P2P Clients Forward Caching aktiviert. Die IM4P2P Clients sind so eingestellt, dass nach drei normalen Suchen ein Suchvorgang nach cache-würdigen Dateien durchzuführen ist, falls ihr lokaler Cache momentan nicht gefüllt ist. Für den Test mit 6 Clients wird die Zielgröße des Caches auf 10 Dateien gestellt, damit diese Cachesuche in jedem vierten Durchgang ausgeführt werden muss.

Es zeigen sich nur marginale Effekte, in allen Testreihen schwankt der Anteil der Dateien die über einen Cache als Relaystation heruntergeladen werden unter 5%, und nochmals deutlich seltener kommt es dazu, dass die Datei den Umweg über mehr als einen Cache nimmt.
Es ist anzunehmen, dass sich diese Werte in größeren Netzen etwas verbessern, allerdings wurden in den Tests durch die Nutzung breiter verschmierter Cache-Zeitstempel bereits die geringere Anzahl von Rechnern berücksichtigt. Daher wird die Effektivität nicht großartig steigen. Das steht zu erwarten, da die Zeit, welche

die Orginalquelle bis zum Download einer Datei in einen entfernten Cache bereits Online verfügbar ist, von keinem Cache eingeholt werden kann. Und diese Zeitspanne ist vor allem in Distributed Hashtables, wie am Beispiel des Overnet Netz im vorigen Kapitel gesehen, nicht unerheblich.

Für das Overnet Netzwerk als Peer-to-Peer Basis für IM4P2P stellt Forward Caching somit keine Verbesserung dar, sondern erhöht im Gegenteil durch seltenere reguläre Suchen die Antwortzeiten und verursacht einen höheren Verkehr im Netz durch zusätzliche Transfers und Publikationsvorgänge.
In Peer-to-Peer Netzen die nicht so gut mit Quellen umgehen können die nur einmalig vorhanden sind wie Distributed Hashtables ist aber anzunehmen, dass Forward Caching mehr Vorteile bietet.

### 5.2.3 Verkehrslast für den Overnet Client durch IM4P2P

Um die Verkehrslast die durch den Einsatz von IM4P2P für den Overnet Client entsteht abschätzen zu können sollen einige Versuche zur Verkehrslastmessung durchgeführt werden.
Zu diesem Zweck werden die durch verschiedene Aufgabenstellungen an den Overnet Client erzeugten Pakete mit Hilfe von Ethereal [Ethe] mitprotokolliert und anschließend analysiert.

| Aufgabe | Verkehrslast | Δ | „incoming“ | „outgoing“ |
|---|---|---|---|---|
| Leerlauf | 10,5 KB | - | 44% | 56% |
| Suche | 44,7 KB | 34,2 KB | 77% | 23% |
| Publizieren (einfach) | 22,4 KB | 11,9 KB | 55% | 45% |
| Publizieren (Kontextdatei) | 37,7 KB | 27,2 KB | 56% | 44% |

Tabelle 5.1: Mittlere erzeugte Verkehrslast auf IP Ebene durch typische Overnet Aufgabenbereiche im Meßzeitraum von 60 Sekunden

Zunächst wird der Verkehr im Leerlaufzustand gemessen, also ohne dass der Overnet Client vom Nutzer zu Suchen oder Publikationsvorgängen veranlasst wird. Beim dabei gemessenen Verkehr handelt es sich um das ständige „Hintergundrauschen“ durch Anfragen von anderen Clients, welches nach einer gewissen Einschwingphase nach dem ersten Verbindungsaufbau ins Overnet Netzwerk über den Meßzeitraum von 60 Sekunden als konstant angenommen werden kann.
Wie aus Tabelle 5.1 ersichtlich, handelt es sich dabei um relativ geringe 10,5 KB/min bzw. gerade mal 0,175 KB/s. Interessanterweise teilt sich der Leerlauf Verkehr nahezu symmetrisch in eingehenden und ausgehenden Verkehr auf.
Da sämtliche betrachteten Aufgaben wie Suchen und Publizieren vom Overnet Client in deutlich weniger als 60 Sekunden abgearbeitet werden[3], kann nun durch Abzug des mittleren Leerlaufverkehrs vom beobachteten Verkehrsaufkommen innerhalb des immer gleichen Meßzeitraums von 60 Sekunden, der von der Durchführung der jeweiligen Aufgabe generierte Verkehr geschätzt werden (siehe Spalte Δ).
So beträgt der von einer einfachen Suche generierte Verkehr durchschnittlich 34,2 KB. Wobei hier allerdings der Großteil durch die Datenpakete mit Suchergebnissen

[3]Die letzten Suchergebnisse treffen z. B. im Mittel schon nach 20 Sekunden nach Beginn der rekursiven Knotensuche ein.

zum Inhalt erzeugt wird (wie auch am hohen Anteil von 77% des „incoming“ Datenvolumens ersichtlich ist) und daher nur zwischen den zwei End-Knoten Suchender und Verantwortlicher und nicht auf dem kompletten Suchpfad aufläuft. Sucht man dagegen nach einem nicht im DHT gespeicherten Schlüsselwort, so reduziert sich der Verkehr entsprechend um die Verkehrslast der Suchantworten von etwa 30 KB auf den Verkehr einer einzigen Knotensuche von dann also rund 4 KB.
Ein einfacher Publikationsvorgang erzeugt im Mittel einen zusätzlichen Verkehr von 11,9 KB. Darin enthalten ist wie in Abschnitt 2.2.3.2 beschrieben, sowohl die Publikation der veröffentlichenden Knoten ID unter dem Hash des Dateiinhalts, wie auch die Publikation dieses Hashs unter dem Hashwert des Dateinamens. Da diese beiden Vorgänge identisch ablaufen, lässt sich für ein einzelnen atomaren Publish somit ca. 6 KB veranschlagen. Dies ist kein Widerspruch zu den veranschlagten Kosten einer Knotensuche mit 4 KB, denn obwohl ein Publikationsvorgang prinzipiell auch auf einer Knotensuche basiert, müssen hierbei die Werte aus Redundanzgründen unter $k$ Knoten abgelegt werden, anstatt nur bei dem für das Ergebnis verantwortlichen Knoten anzufragen. Der Publikationsvorgang wird als „einfach“ bezeichnet, da bei dieser Messung ein monolithischer Dateiname Verwendung fand, also ein Dateiname ohne Seperatorzeichen, den der Overnet Client unter vielen einzelnen Dateinamen-Teilzeichenketten veröffentlichen müsste.
Im Falle der Publikation einer Kontextdatei kommt genau dieser Aspket zum tragen: Der durch die Veröffentlichung einer Kontextdatei generierte Verkehr ist mit 27,2 KB deutlich höher als der einer einfachen Publikation. Weiterhin nur einmal nötig ist die Veröffentlichung der Knoten ID unter dem Dateihash. Mehrfach nötig ist aber die Publikation der Dateinamensteile, derer gibt es nämlich drei: Die Absender ID, die Empfänger ID sowie den Zeitstempel. Insgesamt ergibt sich also die Notwendigkeit zu 4 atomaren Publish Operationen. Damit bestätigt sich der Verdacht von etwa 6 KB Verkehrslast pro atomarem Publish.
Weiterhin zu bemerken ist, das sich die Lasten eines bzw. daher auch mehrerer Publishs wiederum etwa 1:1 in eingehenden und ausgehenden Verkehr aufteilen.

### 5.2.4 Verkehrslast im Overnet Netz durch IM4P2P

Nach der Betrachtung der Verkehrslast die IM4P2P auf dem lokalen Client erzeugt, folgt nun der Versuch, die schlecht meßbare Belastung für das Overnet Netz durch das im Vergleich zum normalen File-Sharing Betrieb ungewöhnliche Verhalten der Overnet Clients unter Kontrolle von IM4P2P, abzuschätzen.

Die zentrale Kosten verursachende Funktione in einem Kademlia-basierten DHT ist die rekursive Knotensuche. Dabei nähert sich die Anfrage in jedem Schritt mindestens um die Hälfte dem gesuchten Knoten an. In der Bewisskizze in [MaMa02, Abschnitt 3] ist gezeigt das dies im Worst Case in einem Netz mit $n$, im virtuellen Adressraum als nahezu gleichverteilt angenommenen Knoten, in $O(log(n))$ Schritten geschieht. Im folgenden soll von der noch immer sehr konservativen Abschätzung[4] ausgegegangen werden, dass im Mittel $\frac{1}{2} * log_2(n)$ Schritte benötigt werden.
Zusätzlich zu der nötigen Anzahl Schritte hat der in 2.2.3.2 bestimmte Kademlia Parameter $k$ einen großen Einfluß auf die verursachte Netzlast bei einer rekursiven Knotensuche, da er die Anzahl der parallel angefragten Knoten pro Schrittebene

[4] In Paketanalysen des Overnetverkehrs zeigte sich das selbst dieser Wert nur selten erreicht oder überschritten wird, vorausgesetzt die Schätzung der Anzahl Clients des Overnet Command Line Client ist anähernd korrekt.

angibt.
Die Anzahl der pro Knotensuche abzusendenden Anfragen und der zurückkommenden Antworten lässt sich also durch folgenden Term abschätzen:

$$\text{lokale Kosten pro Knotensuche} = \frac{1}{2} * log_2(n) * k$$

Dies sind die Kosten die direkt auf dem suchenden Knoten durch eine Knotensuche entstehen, im Netz selbst entsteht durch die Suche allerdings auch eine Last, die sich über viele Suchen gemittelt als gleichverteilt über die aktiven Knoten annehmen lässt. Angenommen $m$ Overnet Knoten ($0 < m < n$) starten quasi gleichzeitig eine Suche, dann entstehen für alle anderen Knoten folgende Kosten:

$$\text{Kosten durch Beantwortung fremder Knotensuchen} = m * \frac{\frac{1}{2} * log_2(n) * k}{n}$$

Die Gesamtkosten für einen der m an der Suche beteiligten Knoten addieren sich demzufolge auf folgenden Wert:

$$\frac{1}{2} * log_2(n) * k + \frac{m}{n} * \frac{1}{2} * log_2(n) * k = (1 + \frac{m}{n}) * \frac{1}{2} * log_2(n) * k$$

Angenommen das Such- und Publikationsintervall tragen den gleichen Wert, was sich in einem vorhergehenden Kapitel als nicht ungünstig erwiesen hat. Dann sind nach Ablauf dieses Intervalls eine Knotensuche für die Suche nach neuen Dateien nötig, sowie im schlimmsten Fall die Publikation einer neuen Datei für alle $c$ Kontakte auf der Kontaktliste, die mit jeweils vier Knotensuchen (drei Dateinamensteile und ein mal Dateihash) zu Buche schlägt. Weiterhin angenommen eine einzige Knotensuchteiloperation verusacht $b$ Bytes Netzverkehr, dann werden nach jedem Intervall folgende Netzkosten verursacht:

$$\text{Kosten (in Byte) pro Intervall} = 5 * c * b * (1 + \frac{m}{n}) * \frac{1}{2} * log_2(n) * k$$

Für einen Beispielfall von im Mittel $c = 10$ aktiven Kontakten, $b = 214$ Bytes (ein SEARCH Paket plus die Antwort in Form eines SEARCH_NEXT Pakets), sowie eine Netzgröße von $n = 2^{20}$ also etwas über einer Millionen Overnet Knoten, $k = 4$ und einer Verbreitung von IM4P2P auf etwa einem Drittel aller Overnet Knoten, also $m = \frac{1}{3} * n$ ergibt sich eine Transferlast für jeden einzelnen Knoten von 557,3 KB pro Intervall!
Um dann unterhalb der Leistung einer ISDN Leitung von 16KB/s (angenommen der Verkehr teilt sich symmetrisch in ein- und ausgehende Last) zu bleiben sollten die Intervalle nicht unter 35 Sekunden sinken.

Um diese Betrachtung abzuschließen, bleibt zu sagen, dass es sich um eine absolute Worst Case Betrachtung handelt, und viele positive Effekte, wie z. B. der wesentlich geringere Aufwand für erneute Suchen nach derselben Hash ID oder der nicht nötigen Wiederveröffentlichung von Dateien falls keine neuen Nachrichten vorliegen, daher keine Berücksichtigung fanden.
Allerdings wurden hier auch lediglich die Kosten der rekursiven Knotensuche abgeschätzt. Die Rückgabe von Suchergebnissen sowie die Kosten des eigentlichen Dateitransfers fanden keine Berücksichtigung. Diese lassen sich in dieser Betrachtung allerdings vernachlässigen da diese Kosten immer nur zwischen zwei einzelnen Knoten auftreten und nicht noch weitere Knoten im Netz belasten.

# 6. Zusammenfassung und Ausblick

Die Aufgabe bestand darin, Peer-to-Peer Netze in generischer Form um die Funktionalität von Instant Messaging zu erweitern. Eine Untersuchung verbreiteter Instant Messenger Anwendungen ergab eine Liste gewünschter Zielfunktionalitäten, wie den Nachrichtenaustausch zwischen zwei Teilnehmern und die Publikation eines Online Status. Aufbauend auf der Analyse im breiten Einsatz befindlicher Peer-to-Peer Netze wurde dann eine generische Abstraktion der Fähigkeiten dieser Netze entworfen, hinter der die Eigenheiten der konkreten Netze verborgen werden können. Diese Abstraktion enthält im wesentlichen Funktionen zur Freigabe von Dateien und zur Suche nach Dateien im Peer-to-Peer Netz. Zur Durchsetzung einer größtmöglichen Modularität der Anwendung wurde die Instant Messaging Oberfläche ebenfalls duch eine generische Abstraktion repräsentiert.
Eine Zusammenführung dieser beiden Module, Peer-to-Peer Schnittstelle und Instant Messaging Oberfläche, geschah in der zu erstellenden Anwendung IM4P2P, der damit auch die Verantwortung übertragen wurde, die Instant Messaging Ereignisse auf Ereignisse an der Peer-to-Peer Schnittstelle abzubilden. Diese Abbildung wurde über das regelmäßige Freigeben und Suchen von Dateien im Peer-to-Peer Netz realisiert.
Das Ziel der Einbindung bereits existierender Awendungen wurde durch die Wahl des Overnet Command Line Client, einer weit verbreiteten auf einem Distributed Hashtable Verfahren basierenden Peer-to-Peer Anwendung, sowie durch den Einsatz von EbQT als grafischer Oberfläche erreicht.
Die Implementierung von IM4P2P und die Anbindung der externen Anwendungen geschah dann in einer modularen Softwarearchitekuter in C++ unter Linux.
Im abschließenden Test der Implementierung ergaben sich daraufhin einige Erkenntnisse über das zugrunde liegende Peer-to-Peer Netz Overnet, z. B. über das Ausbreitungsverhalten der Informationen über neue Dateien im Netz.

Ansatzpunkte für weitere Arbeiten ergeben sich zum einen aus der Modularität der erstellten Anwendung: Das Verhalten anderer Peer-to-Peer Netze als Basis für IM4P2P dürfte einige interessante Rückschlüsse zulassen.
Zum anderen stellt sich die Frage inwieweit durch eine Implementierung der Instant Messaging Funktionalität auf Peer-to-Peer Protokollebene, unter Inkaufnahme des damit einhergehenden Verlustes der generischen Form der Peer-to-Peer Anbindung,

bestimmte Probleme umgangen werden können, wie etwa die Notwendigkeit zur Verschlüsselung und Authentifizierung oder der hohe Aufwand des Polling Verfahrens.

# Literatur


[AdHu00] E. Adar und B. Huberman. Free Riding on Gnutella. Technischer Bericht, Xerox PARC, 2000.

[AIM] AOL Instant Messenger. `http://www.aim.com/`.

[BhSV03] R. Bhagwan, S. Savage und G. Voelker. Understanding Availability. In *Proceedings 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[CzCH00] M. Czerwinski, E. Cutrell und E. Horvitz. Instant Messaging and Interruption: Influence of Task Type on Performance. In *Proceedings of the Annual Conference of the Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia (OzCHI)*, 2000.

[EBlG] EB-lite GUI/Core-Protokoll Spezifikation. `http://www.everybuddy.com/eb-lite/GUI_SPEC`.

[EBli] EB-lite Homepage. `http://www.everybuddy.com/eb-lite/`.

[edGP] ed2k-gtk-gui Project: Overnet GUI Protokollspezifikation. `http://sourceforge.net/cvs/?group_id=61752`. in ed2k_gui/ed2k_gui/gui_core_protocol.h CVS Revision 1.7.

[edGU] ed2k-gtk-gui Homepage. `http://ed2k-gtk-gui.sourceforge.net/`.

[eDon] eDonkey2000 Homepage. `http://www.edonkey2000.com/`.

[eMul] eMule Project Homepage. `http://www.emule-project.net/`.

[Ethe] Ethereal Network Analyzer Homepage. `http://www.ethereal.com/`.

[Gaim] Gaim Homepage. `http://gaim.sourceforge.net/`.

[GnP4] Gnutella Protocol Specification v0.4 Document Revision 1.2. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

[GnP6] Gnutella Protocol Development: RFC-Gnutella 0.6. `http://rfc-gnutella.sourceforge.net/developer/testing/index.html`.

[Gnuc] Gnucleus Homepage. `http://www.gnucleus.com/Gnucleus/`.

[GnWC] Gnutella Web Caching System. `http://www.gnucleus.com/gwebcache/`.

[GrPa02] Rebecca E. Grinter und Leysia Palen. Instant Messaging in Teen Life. In *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*. ACM Press, 2002, S. 21–30.

[HHHL+02] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker und Ion Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, S. 242–259.

[ICQ] ICQ Homepage. `http://web.icq.com/`.

[ICQS] The ICQ Story. `http://company.icq.com/info/icqstory.html`.

[Isak01] Henrik Isaksson. Version 5 of the ICQ protocol. `http://www.algonet.se/~henisak/icq/icqv5.html`, 2001.

[Ivko01] Igor Ivkovic. Improving Gnutella Protocol: Protocol Analysis And Research Proposals. Technischer Bericht, 2001.

[Jabb] Jabber Software Foundation Homepage. `http://www.jabber.org/`.

[Klim03] Alexey Klimkin. Unofficial eDonkey Protocol Specification v0.6.1. `http://sourceforge.net/projects/pdonkey/`, 2003.

[Kope] Kopete, The KDE Instant Messenger. `http://www.kopete.org/`.

[Kuhl] Rüdiger Kuhlmann. ICQ/OSCAR Protocol Version 8. `http://www.stud.uni-karlsruhe.de/~uck4/ICQ/`.

[Lime] LimeWire Homepage. `http://www.limewire.com/`.

[MaMa02] Petar Maymounkov und David Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, S. 53–65.

[MLdo] MLdonkey Homepage. `http://www.nongnu.org/mldonkey/`.

[Mori] André Moritz. Die Napster-Story. `http://www.findmusic.de/mp3suche/mp3-software-napster-story.htm`.

[MSNM] MSN Messenger Homepage. `http://messenger.msn.com/`.

[NaWB00] Bonnie A. Nardi, Steve Whittaker und Erin Bradner. Interaction and Outeraction: Instant Messaging in Action. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*. ACM Press, 2000, S. 79–88.

[Over] Overnet Homepage. `http://www.overnet.com/`.

[RaSS02] Sylvia Ratnasamy, Scott Shenker und Ion Stoica. Routing Algorithms for DHTs: Some Open Questions. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[RFHK+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp und Scott Schenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2001, S. 161–172.

[Ripe01] M. Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network. Technischer Bericht, University of Chicago, 2001.

[Ritt01] Jordan Ritter. Why Gnutella Can't Scale. No, Really. `http://www.darkridge.com/~jpr5/doc/gnutella.html`, 2001.

[Rive92] R. Rivest. RFC 1320: The MD4 Message-Digest Algorithm, April 1992. Status: INFORMATIONAL.

[Shir00] Clay Shirky. What is P2P... and what isn't? `http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html`, 2000.

[Shut] Alexandr Shutko. OSCAR (ICQ v7/v8/v9) protocol documentation. `http://iserverd1.khstu.ru/oscar/`.

[SMKK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek und Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2001, S. 149–160.

[SoFo] SourceForge Homepage. `http://sourceforge.net/`.

[XMPP] Extensible Messaging and Presence Protocol (XMPP) IETF Working Group. `http://www.ietf.org/html.charters/xmpp-charter.html`.

[xMul] The xMule P2P Project. `http://www.xmule.org/`.

[ZhJK02] Ben Y. Zhao, Anthony Joseph und John Kubiatowicz. Locality Aware Mechanisms for Large-scale Networks. In *Proceedings of the International Workshop on Future Directions in Distributed Computing (FuDiCo)*, 2002.