technische universität
dortmund

PG-Zwischenbericht

PG 526

„UbiMuC“
-
Ubiquitous Multimedia-Community

Fakultät für Informatik
Institut für eingebettete Systeme

Autoren:
Björn Bosselmann, Markus Görlich, Thomas Grabowski,
Nils Kneuper, Michael Kupiec, Lutz Krumme,
Fabian Marth, Michael Puchowezki, Markus Straube,
Stephan Vogt, Stefan Wahoff, Jens Wrede

Betreuer:
Dr. Heiko Falk
Constantin Timm

Abgabedatum:
30. September 2008

Inhaltsverzeichnis

1 Motivation (Jens, Nils) 7
- 1.1 Aufgabenstellung . . . 7
  - 1.1.1 Ziele . . . 7
  - 1.1.2 Minimalziele . . . 8
- 1.2 Struktur des Dokuments . . . 8

2 Anwendungsfälle für UbiMuC 9
- 2.1 P2P-Funktionalität . . . 10
  - 2.1.1 Verbindungsaufbau . . . 10
  - 2.1.2 Personensuche und -verwaltung . . . 11
  - 2.1.3 Inhaltssuche . . . 12
  - 2.1.4 Veröffentlichung von Inhalten . . . 13
  - 2.1.5 Weiterleitung von Daten . . . 14
- 2.2 Multimedia . . . 15
  - 2.2.1 AV-Konferenz . . . 15
  - 2.2.2 AV-Streaming . . . 16

3 „WIR“ – WIR Interaction Relay (Björn, Lutz) 17
- 3.1 Einleitung . . . 17
- 3.2 Aufgaben der WIR-Schicht . . . 17
- 3.3 BiDi-Pipes . . . 18
  - 3.3.1 Verbindungsaufbau . . . 18
- 3.4 Datencontainer . . . 19

4 Netzwerk-Konzepte 21
- 4.1 Peer-to-Peer-Netzwerke (Fabian, Markus G.) . . . 21
  - 4.1.1 Serverbasierte Peer-to-Peer-Netzwerke . . . 21
  - 4.1.2 Reine Peer-to-Peer-Netzwerke . . . 22
  - 4.1.3 Hybride Peer-to-Peer-Netzwerke . . . 23
  - 4.1.4 Fazit . . . 24
- 4.2 Finden vorhandener Knoten eines Peer-to-Peer-Netzwerkes (Fabian, Markus G.) . . . 24
  - 4.2.1 Serverbasierte P2P-Variante und Hybride P2P-Variante . . . 24
  - 4.2.2 Reine P2P-Variante . . . 24
  - 4.2.3 Fazit . . . 25
- 4.3 Verschmelzen von Netzen (Thomas, Michael P.) . . . 26
  - 4.3.1 Allgemeines zum Netmerge . . . 26
  - 4.3.2 Szenarien beim Netmerge . . . 26
  - 4.3.3 Weiteres Szenario . . . 28

5 P2P Frameworks 31
- 5.1 Detaillierter Blick auf JXTA (Michael, Markus S.) . . . 31

5.2 Detaillierter Blick auf GNUnet (Fabian, Markus G.) . . . 32
5.2.1 Routingfunktionen von GNUnet (Jens, Michael, Markus S.) . . . 34
5.2.2 Der Austausch von Hostlisten (Björn, Lutz, Stefan, Nils) . . . 36
5.2.3 Die Suche nach Inhalten im Netzwerk mithilfe von Metadaten (Stephan, Fabian, Markus G.) . . . 37
5.3 Entropy (Björn) . . . 40
5.3.1 Überblick . . . 40
5.3.2 Fazit . . . 40
5.4 GiFT (Markus S.) . . . 40
5.4.1 Spezifikationen . . . 40
5.4.2 Protokoll-Plugins . . . 41
5.4.3 Bewertung . . . 42
5.5 Mute (Stephan) . . . 43
5.5.1 Beschreibung . . . 43
5.5.2 Test . . . 43
5.5.3 Fazit . . . 43
5.6 Vergleich von JXTA mit GNUnet (Fabian, Markus G., Markus S., Michael) 43

6 Die Seminarphase und deren Auswertung 46
6.1 P2P (Fabian) . . . 46
6.2 JXTA (Michael K.) . . . 46
6.3 State-of-the-art Multimedia Codecs und -Standards (Stefan) . . . 47
6.4 Streaming Protokolle (Michael P.) . . . 47
6.5 Vergleich, Test und Evaluierung von Streaming-Servern und -Clients (Markus G.) . . . 48
6.6 Allgemeine Sicherheitsalgorithmen (Jens) . . . 48
6.7 Sicherheitsalgorithmen für eingebettete Systeme (Markus S.) . . . 48
6.8 Sicherheitsprotokolle und Implementation (Björn) . . . 49
6.9 C/C++ (Stephan) . . . 50
6.10 N810 (Nils) . . . 50
6.11 WLAN-Netze (Thomas) . . . 50
6.12 Konfigurationsfreie Vernetzung - Zeroconf, Bonjour, mDNS (Lutz) . . . 51
6.13 Abwägung der Bedeutung der einzelnen Seminare für den weiteren Verlauf (Stefan) . . . 51

7 ZeroConf Netzwerke mithilfe von Avahi 54
7.1 ZeroConf Netzwerke und P2P-Frameworks (Lutz, Thomas) . . . 54
7.2 ZeroConf Netzwerke mit dem N810 nutzen (Lutz, Thomas) . . . 55

8 Unsere Arbeiten an GNUnet 56
8.1 GNUnet auf dem N810 (Björn) . . . 56
8.2 Die Einbindung von Avahi in GNUnet (Björn, Lutz) . . . 57
8.3 Die Suche nach Personen mithilfe einer Distributed Hash Table (Markus S., Stefan, Nils) . . . 59

8.3.1 Einleitung . . . 59
8.3.2 Grundlagen . . . 60
8.3.3 Anwendungsszenarien . . . 61
8.3.4 Fazit . . . 63

9 Audio-/Video-Codecs **64**
9.1 Skalierung von Mediaplayern bei verschiedenen Codecs und Bitraten (Björn, Stephan, Nils) . . . 64
9.1.1 Einleitung . . . 64
9.1.2 Erste Testreihe: Serie mit wenigen Bewegungen . . . 65
9.1.3 Zweite Testreihe: Film mit schnellen Bewegungen . . . 65
9.1.4 Dritte Testreihe: Musikstück . . . 66
9.1.5 Fazit . . . 67
9.2 Betrachtung zu unterstützender Multimedia Codecs (Björn, Stephan, Nils) 68
9.2.1 Einleitung . . . 68
9.2.2 Geeignete Formate für Sprachübertragung . . . 68
9.2.3 Geeignete Formate für die Videoübertragung . . . 69

10 Die Entwicklung der Benutzerschnittstelle **70**
10.1 Einführung in GTK (Stephan, Fabian, Markus G.) . . . 70
10.2 Die grundlegende GUI von UbiMuC (Stephan, Fabian, Markus G.) . . . 71
10.2.1 Theoretische Vorüberlegungen . . . 71
10.2.2 Praktische Implementierung . . . 72
10.2.3 Fazit . . . 73
10.3 Die Schnittstelle zu GNUnet (Jens, Thomas, Michael P., Michael K.) . . . 73
10.4 Die Integration des MPlayer (Björn, Lutz) . . . 74
10.4.1 Einleitung . . . 74
10.4.2 Datenübertragung . . . 74
10.4.3 Beispiel . . . 75

11 Zusammenfassung und Ausblick **76**
11.1 Zusammenfassung der bisherigen Ergebnisse der Projektgruppe (Markus S) 76
11.2 Ausblick (Lutz) . . . 80

A Zusammenfassungen der Seminarthemen **82**
A.1 P2P (Fabian) . . . 82
A.2 JXTA (Michael K.) . . . 83
A.3 State-of-the-art Multimedia Codecs und -Standards (Stefan) . . . 84
A.4 Streaming Protokolle (Michael P.) . . . 85
A.5 Vergleich, Test und Evaluierung von Streaming-Servern und -Clients (Markus G.) . . . 87
A.6 Allgemeine Sicherheitsalgorithmen (Jens) . . . 88
A.7 Sicherheitsalgorithmen für eingebettete Systeme (Markus S.) . . . 89
A.8 Sicherheitsprotokolle und Implementation (Björn) . . . 91

A.9 C/C++ (Stephan) . . . 92
A.10 N810 (Nils) . . . 93
A.11 WLAN-Netze (Thomas) . . . 94
A.12 Konfigurationsfreie Vernetzung - Zeroconf, Bonjour, mDNS (Lutz) . . . 95

B N810 **98**
B.1 Allgemeines zur Architektur (Nils, Stefan, Jens) . . . 98
B.2 Die Hardware und deren Features (Nils, Stefan, Jens) . . . 98
B.2.1 Der OMAP2420 . . . 98
B.2.2 Der DSP des N810 . . . 98
B.2.3 Kommunikationsmöglichkeiten . . . 99
B.2.4 Eingabegeräte . . . 99
B.2.5 Stromversorgung . . . 100
B.2.6 Datenspeicher . . . 101
B.3 Betrachtung der Batterielaufzeit des N810 in verschiedenen Szenarien (Nils, Stefan, Jens) . . . 102

C Unsere Arbeitsumgebung **104**
C.1 Die Arbeitsumgebung im CI-Lab (Markus S.) . . . 104
C.2 Verwendete Werkzeuge zur Koordinierung von Arbeitsergebnissen (Markus S.) . . . 105
C.3 Das Software Development Kit Scratchbox (Markus S.) . . . 106

Literatur **108**

Abbildungsverzeichnis **111**

Tabellenverzeichnis **111**

1 Motivation (Jens, Nils)

Mit der zunehmenden Verbreitung von drahtlosen Netzwerktechnologien und deren Integration in Mobiltelefonen, Netbooks und Internet-Tablets wächst gleichzeitig das Bedürfnis nach entsprechender Software zur Nutzung und Interaktion mit den verschiedenen Technologien. Für den Multimediasektor gibt es derweil Insellösungen, die beispielsweise das Streaming von Inhalten durch einen dedizierten Medienserver zu drahtlosen Teilnehmern ermöglichen. Auf Seiten der drahtlosen Kommunikation durch Mobiltelefone gibt es dank Smartphones und UMTS nun auch die Möglichkeit der Internet-Telefonie per VoIP.

Neu sind in diesem Kontext allerdings die sog. „Netbooks“ und Internet-Tablets. Diese möglichst handlichen Geräte zeichnen sich in erster Linie dadurch aus, ein Taschencomputer mit Internet-Zugang zu sein. Dabei werden auf kleinstem Raum Speicherplatz, Schnittstellen und Eingabemöglichkeiten realisiert, so dass der mobilen Internetnutzung nichts im Wege stehen soll. Dieser Vision stehen allerdings noch einige Hürden gegenüber, so beispielsweise die flächendeckende Möglichkeit des Netzwerkzugangs durch offene WLANs oder auch die Unterstützung durch dem Geräteumfeld angepasste Software. Da mit Hilfe eines WLAN auch kurzlebige Ad-Hoc-Netzwerke aufgebaut werden können, wäre die Realisierung einer vielseitigen Peer-to-Peer-Umgebung mit Kommunikations- und Streaming-Möglichkeiten für solch ein Internet-Tablet ein lohnenswertes Unterfangen. Einzellösungen gibt es dabei durch Skype, Streaming-Dienste und P2P-Plattformen zu Hauf, eine integrierte Umgebung auf Basis eines Ad-Hoc-Netzwerkes ist allerdings Neuland.

1.1 Aufgabenstellung

Genau eine solche Umgebung soll die „Ubiquitous Multimedia Community“-Projektgruppe (kurz UbiMuC) realisieren: Auf Basis eines bestehenden P2P-Frameworks und einem Nokia N810 [1] als Zielumgebung soll eine Anwendung erstellt werden, welche dem Nutzer über der reinen Dateiaustausch-Option hinaus auch noch Funktionen wie Video-Streaming, AV-Konferenzen und Chat-Möglichkeiten bereitstellt. Vorrangiges Ziel ist dabei jedoch nicht die Eigenentwicklung der gesamten P2P-Basisfunktionalitäten, sondern eher die Erweiterung eines bestehenden Frameworks um die erwähnten Funktionen. Dies setzt natürlich voraus, dass die einzelnen P2P-Framework-Kandidaten sowohl hinsichtlich ihrer Anpassungsmöglichkeiten, als auch des rechtlichen Rahmens (GPL) überprüft werden.

1.1.1 Ziele

Neben der Implementierung einer P2P-Multimedia-Community auf der mobilen Plattform soll die erstellte Anwendung ebenfalls hinsichtlich ihrer Performance und des Ressourcenverbrauchs getestet werden. Darunter fallen unter anderem die Ermittlung von

Durchsatzwerten, Delay und Jitter, ebenso wie Overhead und insgesamte Routingeffizienz.

1.1.2 Minimalziele

Als Minimalziel steht die Entwicklung eines Prototyps für eine P2P-Community im Vordergrund. Dabei soll es zumindest möglich sein, Dateien zwischen den Endgeräten austauschen zu können. Dieser Transfer ist mittels Verschlüsselungsfunktionen abzusichern. Im Hinblick auf den Aspekt des Testens wird erwartet, dass wenigstens die Transcodierung von Multimediainhalten sowie Dateiübertragung an sich und die Effizienz der Verschlüsselung getestet wird.

1.2 Struktur des Dokuments

Zur Gewinnung eines ersten Überblicks über die zu realisierenden Ziele folgen zunächst die identifizierten Anwendungsfälle und die daraus resultierenden ersten Arbeitsansätze. Auf den Anwendungsfällen aufbauend wird die WIR-Schicht zur Steuerung der Applikation näher betrachtet und skizziert. Ebenso wird auf die Konsequenzen und Probleme unterschiedlicher Netzwerktopologien eingegangen, da diese beim Entwurf berücksichtigt werden müssen. Ein weiteres Kapitel beschäftigt sich ausschließlich mit einigen P2P-Frameworks, welche dort detailliert betrachtet werden. Basierend auf den Auswertungen der Serminarphase folgen dann die ersten Konzeptarbeiten am P2P-Framework »GNUnet« und dessen Kopplung mit dem Ad-Hoc-Netzwerkdienst »Avahi«. Um den Aspekt der Multimedia-Kommunikation und des Streamings abzudecken, beschäftigt sich ein Kapitel mit der Codec-Auswahl und Testreihen zur Hardwarebelastung. Das vorletzte Kapitel behandelt die Entwicklung der grafischen Benutzerschnittstelle, während zum Abschluss ein Ausblick auf kommenden Arbeiten und Vorhaben gegeben wird.

2 Anwendungsfälle für UbiMuC

Zur möglichst präzisen Abgrenzung der unterschiedlichen Anforderungen an die UbiMuC-Umgebung, wurden Anwendungsfälle ausgearbeitet, die einige potenzielle Nutzungsszenarien der Software verdeutlichen. Im Mittelpunkt der Betrachtung stehen dabei nicht nur die offensichtlichen Handlungen eines Nutzers innerhalb der P2P-Anwendung, sondern auch grundlegende Szenarien wie die Weiterleitung von Daten oder die Verbindung zum Netzwerk an sich. Entsprechend der im ersten Kapitel angeschnittenen Anforderungen wurden insgesamt acht unterschiedliche Anwendungsfälle erkannt und ausgearbeitet. Zentral sind dabei vor allen Dingen die eigentlichen „Anwendungsmöglichkeiten“ beim Nutzer, die in Abb.1 dargestellt sind:

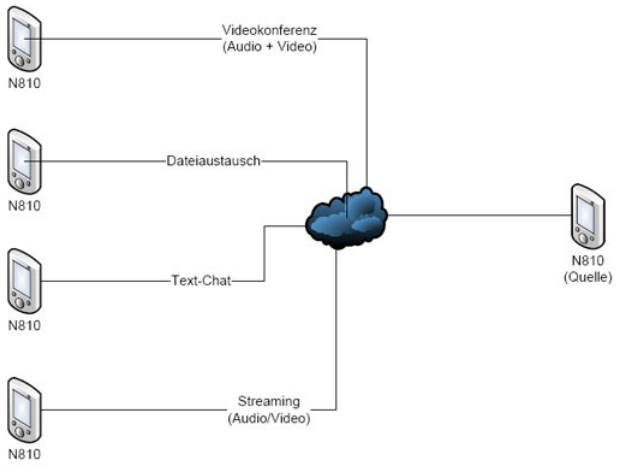

Abbildung 1: UbiMuC-Anwendungsfälle

Ein typisches Szenario für die Nutzung der UbiMuC-Software besteht aus einem spontan gebildeten Ad-Hoc-Netzwerk mit mehreren N810, welche alle unterschiedliche Ziele verfolgen können. Während einige Teilnehmer beispielsweise in einer Videokonferenz eine Unterhaltung führen, können andere das bestehende Netzwerk nach Dateien durchsuchen und im Anschluss untereinander tauschen. Parallel können auch reine Text-Chats zwischen den Nutzern laufen, ebenso wie Streaming von Multimediadateien wie MP3s oder Videos durchgeführt werden kann.

Neben diesen Nutzungsoptionen gibt es noch das Problem, wie dem P2P-Netzwerk überhaupt beigetreten werden kann. Durch die vorhandenen Netzwerkfähigkeiten des N810 bieten sich folgende zwei Arten an: Ad-Hoc-Verbindung über ein bestehendes WLAN oder Verbindung über ein Weitverkehrsnetz, wie dem Internet. Dem Nutzer sollte also beim Start der Anwendung die Möglichkeit offengelassen werden, welche Art der Verbindung er aktuell bevorzugt.

2.1 P2P-Funktionalität

In Bezug auf die Basisfunktionen eines P2P-Frameworks werden in diesem Bereich die Verbindungsaufnahme, Server- und Inhaltssuche sowie das Veröffentlichen von Inhalten und die Datenweiterleitung durch benachbarte Peers betrachtet. Ebenfalls wird die Personensuche und -verwaltung betrachtet, welche ein wichtiger Bestandteil der Multimedia-Community ist.

2.1.1 Verbindungsaufbau

Für die Verbindung über das WAN (Wide Area Network) muss der Client über eine Server-Adresse verfügen, über welche er weitere Peer-Adressen herausfinden kann. Alternativ kann er sich auch eine gültige Adresse von einem bekannten Peer mitteilen lassen, um so die Verbindung aufzubauen. Abbildung 2 verdeutlich dabei ein Szenario, in welchem sich ein neuer Client in ein bestehendes Netzwerk einklinken möchte und bereits Kenntnis über die Adresse eines Servers hat. Er baut dabei eine Verbindung zu einem Registry-Server auf, dem weitere Adressen zu anderen Teilnehmern bekannt sind. Dabei ist es aus Gründen der gesteigerten Serviceverfügbarkeit auch denkbar, dass es mehrere Server-Dienste gibt, welche untereinander Nachrichten über die bei ihnen gemeldeten Teilnehmer austauschen.

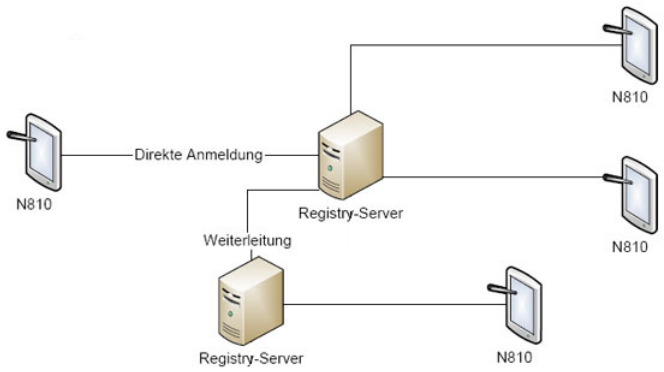

Abbildung 2: Anwendungsfall: Verbindungsaufbau

Befindet sich der neue Teilnehmer nicht innerhalb des WAN-Modus, sondern möchte einem Ad-Hoc-Netzwerk beitreten, benötigt er ebenfalls die Adresses eines vorhandenen Servers. Es muss also ein weiterer Peer in Reichweite gefunden werden, um von diesem die Adresse zu erhalten. Dies sollte mittels einer Art Broadcast geschehen, sodass alle erreichten Peers dem Neuankömmling eine Adresse mitteilen. Dieser kann sich dann zu der mitgeteilten Adresse verbinden und dem Netzwerk erfolgreich beitreten. Abbildung 3 zeigt eine Skizze des Aufbaus. Um die Adressverteilung innerhalb des Netzwerks zu bewerkstelligen, spielt der Avahi-Dienst die zentrale Rolle: Dieser soll sicherstellen, dass

die Adressaushandlung innerhalb des Ad-Hoc-Netzes automatisch auf Kollisionsfreiheit überprüft wird, so dass neue Peers nicht mit bereits vergebenen Adressen beitreten können.

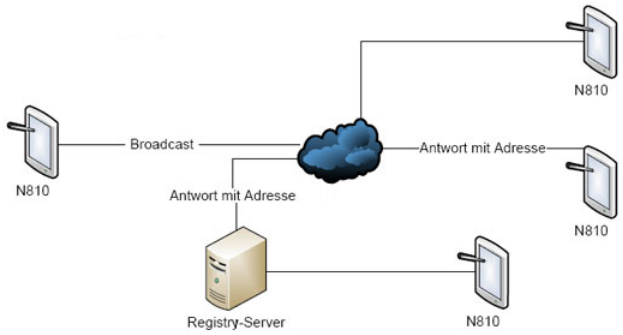

Abbildung 3: Anwendungsfall: Serversuche

2.1.2 Personensuche und -verwaltung

Eine der zentralen Anforderungen an eine Multimedia-Community ist die Personensuche bzw. die Personenverwaltung. Sie bietet die Grundlage für die Kommunikation zwischen mehreren Netzteilnehmern, indem die bekannten Personen in Listen verwaltet werden und der Onlinestatus überprüft wird.

Wenn jetzt eine Person online geht kann sie, wie in Abbildung 4 dargestellt, ihren Onlinestatus bei den anderen Netzteilnehmern ändern, indem sie einen Broadcast an alle Netzteilnehmer schickt. Die Netzteilnehmer, die den Broadcast empfangen haben, können nun überprüfen, ob sie diese Person in ihrer Kontaktliste haben und ggf. ihren Status ändern. Zusätzlich vermerkt sie ihren Onlinestatus in einer Datenbank, wo sie auch den Onlinestatus der anderen Netzteilnehmer abfragen kann.

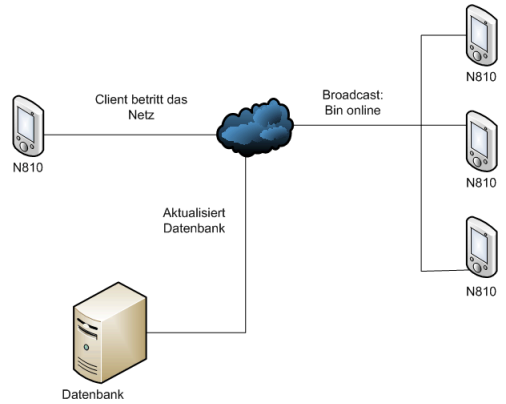

Abbildung 4: Anwendungsfall: Person geht online

Um eine Person zu seiner Kontaktliste hinzuzufügen, sucht der Netzteilnehmer in einer Datenbank, in der alle Netzteilnehmer vermerkt sind. In Abbildung 5 sieht man, wie der Client die Anfrage stellt und von der Datenbank die Informationen, die Adresse und den Onlinestatus des gewünschten Netzteilnehmers erhält.

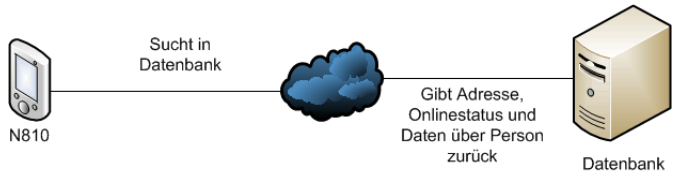

Abbildung 5: Anwendungsfall: Person sucht in Datenbank

2.1.3 Inhaltssuche

Um Dateien über das P2P-Netzwerk auszutauschen, gibt es entsprechend zur vorherigen Verbindungsauswahl ebenso zwei Szenarien: Tausch über das Internet oder Tausch über ein Ad-Hoc-Netz (vgl. Abb. 6).

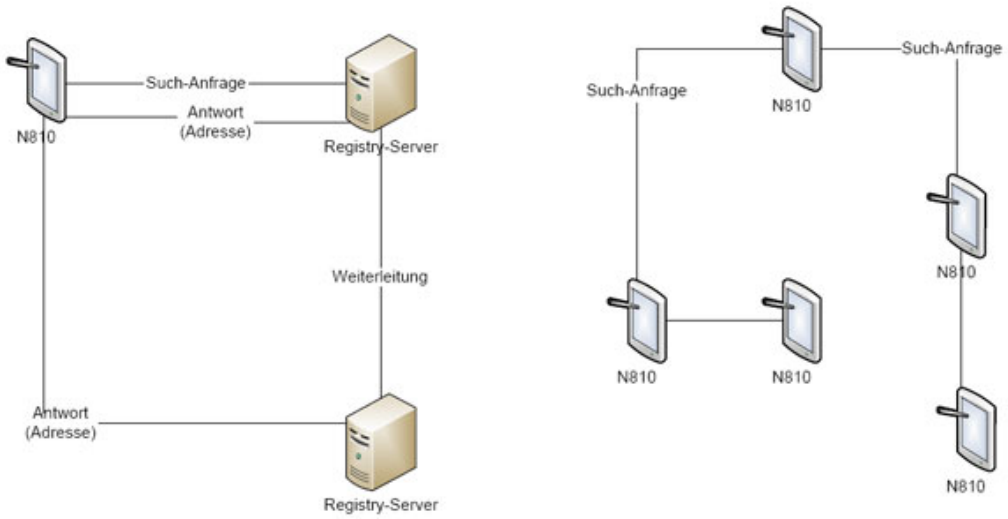

Abbildung 6: Anwendungsfall: Inhaltssuche

Handelt es sich um ein Netz, welches über das Internet aufgebaut wurde, so muss der Client seine Anfrage an den Server schicken, welcher diese entweder beantwortet oder an andere Maschinen weiterleitet. Sofern mehrere Server eingesetzt werden, sollten diese untereinander eine Verbindung haben und einen Index über bei sich registrierte Peers führen. Auf diese Weise können Anfragen schnell beantwortet und weitergeleitet werden. Inwiefern die Server auch wissen müssen, welche Dateien welche Peers zur Verfügung stellen, wurde offen gelassen.

Bei einem Ad-Hoc-Netz gestaltet sich die Suche nach Inhalten etwas anders: Suchanfragen werden nicht an eine zentrale Instanz zur Bearbeitung geleitet, sondern direkt an bekannte Peers gestellt. Diese können die eingehenden Anfragen mit ihrem Angebot abgleichen und entsprechend direkt beantworten, oder aber auch an andere Peers weiterleiten. Eine bestehende Verbindung zur Datenübertragung ist dabei natürlich eine zwingende Voraussetzung. Es wurde bei diesem Szenario offen gelassen, wie oft eine Weiterleitung von Anfragen stattfinden darf. Eine Netzflutung durch zu häufige Weiterleitungen sollte jedoch unterbunden werden.

2.1.4 Veröffentlichung von Inhalten

Die Bereitstellung von allgemeinen Inhalten kann entsprechend zum gewählten Verbindungsmodell ebenfalls auf zwei Arten geschehen: Falls ein Server zur Verfügung steht, sollte sich ein Peer dort bekanntmachen und die von ihm bereitgestellten Inhalte dem Server mitteilen. Andere Peers können auf diese Weise direkt beim Server erfragen, welche Leistungen und Dienste angeboten werden. Abbildung 7 illustriert den Aufbau.

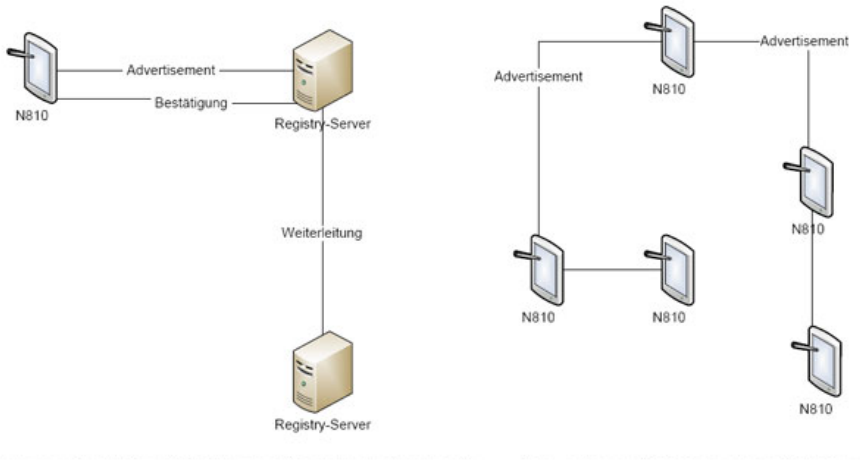

Abbildung 7: Anwendungsfall: Veröffentlichung von Inhalten

Im Kontext eines Ad-Hoc-Netzes gestaltet sich die Veröffentlichung etwas anders: Hier wäre es denkbar, dass ein Peer sowohl eigene Advertismentes an andere Peers sendet um ihnen sein Angebot darzulegen, ebenso können Peers die Angebote der anderen Netzteilnehmer erfragen. Um allerdings ein wahlloses Fluten des Netzes mit eigenen Angebots- oder Suchnachrichten zu unterbinden, sollte unter Umständen eine Schranke festgelegt werden, damit die Angebots/Suche-Anfragen nur eine gewisse Zeit oder Hopanzahl im Netz überleben können.

2.1.5 Weiterleitung von Daten

Wenn keine direkte Verbindung zwischen zwei Netzteilnehmern besteht, werden die Daten über mehrere Netzteilnehmer durch das Netz zum Ziel geleitet. Dabei werden die Daten verschlüsselt, mit Quelle und Ziel versehen und von Peer zu Peer geschickt, bis sie ihr Ziel erreicht haben. In Abb. 8 ist ein Beispiel einer Weiterleitung zu sehen. Bei der Datenübertragung ist jedoch nicht gesichert, dass eine konstante Dienstgüte vorhanden ist, da die zur Verfügung stehenden Datenraten zwischen den einzelnen Peers sehr unterschiedlich ausfallen können.

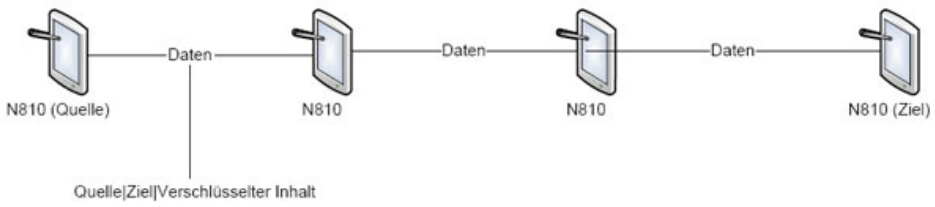

Abbildung 8: Anwendungsfall: Weiterleitung von Daten

2.2 Multimedia

Neben den bereits erwähnten Basisfunktionen soll die UbiMuC-Umgebung in der Lage sein, auch mit Audio- und Videoströmen adäquat umgehen zu können. Dabei soll es über die integrierte Webcam und das Mikrofon des N810 möglich sein, eine Videokonferenz zu einem anderen Netzteilnehmer aufzubauen. In Anlehnung daran steht auch das Anbieten eines reinen Streaming-Diensts im Raum, so dass Teilnehmer die multimedialen Inhalte von anderen Peers direkt abspielen können, ohne die Datei vorher vollständig herunterladen zu müssen.

2.2.1 AV-Konferenz

Für die AV-Anwendungen gibt es zwei Nutzungsszenarien. Zum einen das AV-Streaming, bei dem der Nutzer AV-Daten abrufen kann, die über das Netz gestreamt werden, oder er bietet selber AV-Daten zum Streamen an. Zum anderen die AV-Konferenz, bei der zwei Netzteilnehmer sich per Videokonferenz unterhalten können.

Bei der AV-Konferenz in Abb. 9 müssen zuerst die Parameter wie Auflösung, Codec und Bitrate ausgehandelt werden, bevor man starten kann. Dann werden die Daten von der Kamera und dem Mikrofon aufgenommen, codiert und zu einem Datenstrom verpackt, der an die anderen Teilnehmern geschickt wird. Der mögliche Durchsatz der Netzleitung beschränkt hier den Datenstrom genauso, wie die Anzahl der Hops, die zwischen den Teilnehmern liegen.

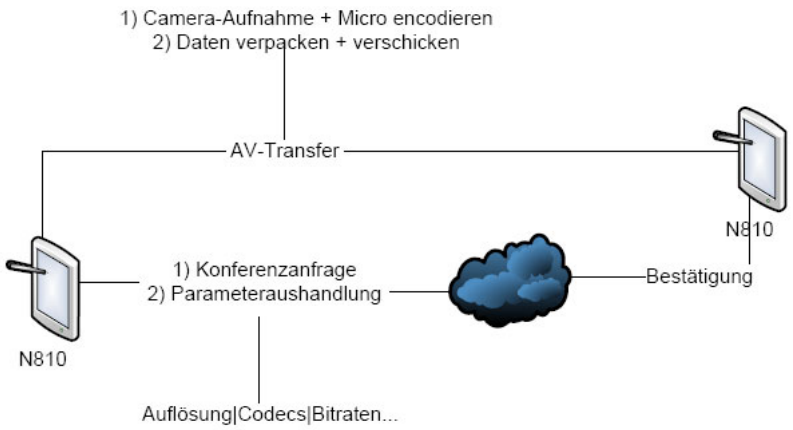

Abbildung 9: Anwendungsfall: AV-Konferenz

2.2.2 AV-Streaming

Für das AV-Streaming sieht das Szenario in Abb. 10 ähnlich aus. Ein Interessent stellt eine Streaming-Anfrage an einen anderen Peer, welcher den Stream bestätigt und mit der Aussendung des Datenstroms beginnt. Der Client hat nun die Möglichkeit über RTSP-Kommandos den Stream zu steuern. Darunter sollten sich unter anderem gängige Optionen wie Vorspulen, Zurückspulen und Pause befinden. Die Beschränkungen im Hinblick auf die nutzbare Übertragungsrate des Netzes sind hier genauso gelagert, wie bei der AV-Konferenz und hängen im Wesentlichen von der Hopanzahl zwischen den beiden Parteien ab.

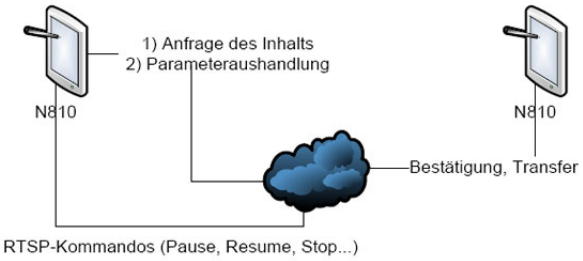

Abbildung 10: Anwendungsfall: AV-Streaming

3 „WIR“ – WIR Interaction Relay (Björn, Lutz)

3.1 Einleitung

Dieses Kapitel handelt von der Schicht zwischen GNUnet-Kern und der GUI, dem sogenannten WIR Interaction Relay, kurz *WIR*. Damit die Basisfunktionen von GNUnet von unserer Anwendung genutzt werden können, ist es erforderlich, die zentralen GNUnet-Programmteile zu kapseln und so für uns nutzbar zu machen. Durch die Kapselung sollen weitergehende Eingriffe in den GNUnet-Quellcode umgangen werden. Im Falle von späteren Weiterentwicklungen oder Codeveränderungen durch die GNUnet-Community ist es dann höchstens notwendig, unsere Schnittstellen anzupassen. Alle Anfragen unserer Anwendung an das GNUnet-Netzwerk sollen in diesen Zwischenklassen übersetzt und angepasst werden. Auf der anderen Seite müssen dort ebenfalls die Antworten von GNUnet interpretiert und unter Umständen aufbereitet werden. Für den Nutzer soll diese Schicht vollständig transparent sein. Die abgesetzten Kommandos werden von der GUI lediglich weitergeleitet. Die Antworten der Schicht werden im Anschluss von der GUI lediglich dargestellt, während die eigentliche Verarbeitungsintelligenz innerhalb der Adapterklasse liegt. Auf diese Weise sollte ein Ersetzen der GUI in der Zukunft deutlich vereinfacht werden, da lediglich die Kommandoweiterleitung und Darstellung der Ergebnisse angepasst werden müssen.

3.2 Aufgaben der WIR-Schicht

Neben der oben genannten Schnittstellen-Funktion der WIR-Schicht, die einzelnen Komponenten, wie GUI, *MPlayer* und *GNUnet* zu verbinden, hat die WIR-Schicht insbesondere noch die Aufgabe, dafür Sorge zu tragen, dass die aufgenommenen Informationen von Webcam und Mikrofon passend kodiert und für GNUnet verpackt werden.

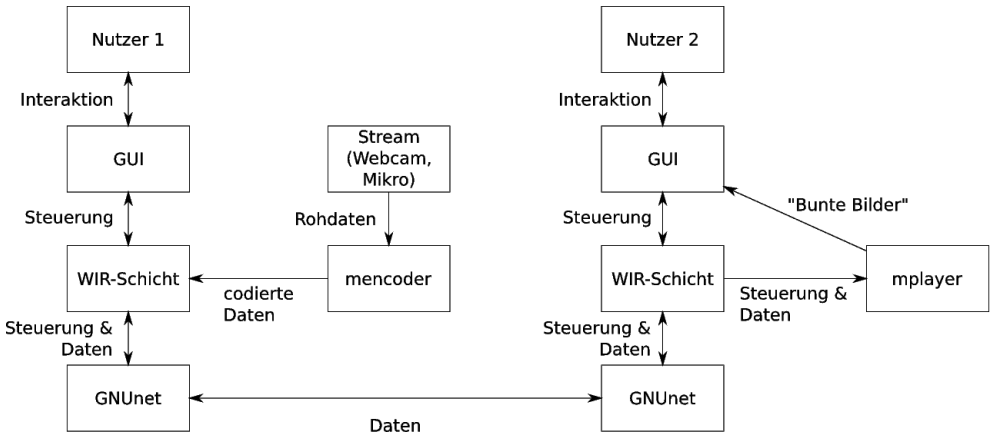

Abbildung 11: Schema der UbiMuC-Schichten

In Abbildung 11 wird das Zusammenspiel der einzelnen Komponenten der *UbiMuC*-Plattform graphisch dargestellt. Die Pfeilrichtung gibt dabei die Flussrichtung der Befehle und Daten an.

Wählt der Nutzer beispielsweise die Videotelefonie-Applikation in der GUI aus, wird die *WIR*-Schicht eine *MEncoder*-Instanz starten. Diese steuert selbstständig die im N810 eingebaute Webcam und das Mikrofon an, die über die standardisierten Linux-Multimediaschnittstellen *Video4Linux* und *ALSA* bereitgestellt werden. Die so gewonnenen Video- und Audiorohdaten werden vom *MEncoder*-Prozess in Echtzeit in ein Format konvertiert, das sich relativ leicht über ein Netzwerk streamen lässt (s. Kapitel 9.2). Die *WIR*-Schicht nimmt die nun komprimierten Daten wieder entgegen, um sie für die *GNUnet*-Ebene vorzubereiten. Dazu gehört u. a. das Zerstückeln des Datenstreams in Pakete geeigneter Größe oder das Einfügen von Sequenznummern für die spätere Wiederherstellung der korrekten Reihenfolge. Anschließend werden die Pakete der *GNUnet*-Schicht übergeben und damit über das Netzwerk übertragen.

Auf der Empfängerseite erhält die *WIR*-Schicht die Pakete von *GNUnet*, die entsprechend der Paketierung auf der Senderseite wieder zu einem kontinuierlichen Datenstrom zusammengesetzt werden. Die GUI wird nun angewiesen, eine leere Fläche zu erzeugen, die später für die Videoausgabe verwendet werden soll. Gleichzeitig wird eine *MPlayer*-Instanz im sogenannten *Slave*-Modus gestartet (s. Kapitel 10.4), die von der *WIR*-Schicht mit den kodierten Audio- und Videodaten versorgt wird. Dabei wird dem *MPlayer* zusätzlich noch die Identifikationsnummer der von der GUI erzeugten, aber noch leeren Fläche übergeben, um dem Player so direkten Zugriff auf die GUI-Fläche zu ermöglichen. Dieser wird dann dort die Videoausgabe anzeigen.

3.3 BiDi-Pipes

Zur Übertragung von Stream- bzw. Chat-Daten und für die Möglichkeit, Dateien aktiv zu senden, haben wir uns entschieden, bidirektionale Pipes zu benutzen. Die Einbindung solcher Pipes in GNUnet läuft über die Deklaration eines neuen Nutzdatentyps. Abhängig vom angegebenen Typ im Header eines GNUnet-Pakets wird eine darauf zugeschnittene Verarbeitungsfunktion innerhalb GNUnets aufgerufen. Durch die Deklaration eines neuen Nutzdatentyps ist es möglich, BiDi-Pipes durch die GNUnet-Schicht hindurch von WIR-Schicht zu WIR-Schicht aufzubauen. Diese Pipes sorgen dafür, dass wir eine anhaltende bidirektionale Verbindung zwischen den kommunizierenden Benutzern verwenden können, ohne das grundsätzliche Pull-Paradigma GNUnets, wonach Dateien nur nach Anforderung des Empfängers gesendet werden, brechen zu müssen.

3.3.1 Verbindungsaufbau

Da der Datentransfer innerhalb GNUnets nicht Datei, sondern Datei-Block-basiert ist, stehen wir vor dem Problem, dynamische Daten nicht im Voraus indizieren zu können.

Um trotzdem Kommunikationsstreams ermöglichen zu können, verwenden wir Dateideskriptoren, die über die normalen GNUnet-Mechanismen angefordert werden können. Diese Deskriptoren haben einen statischen Inhalt und sind demzufolge auch über den Hash-Wert identifizierbar. Der Aufbau einer Verbindung für ein Telefonat liefe wie folgt: Vom Anrufer aus wird der Dateideskriptor für ein Telefonat beim Angerufenen angefragt. Diese Anfrage wird von der WIR-Schicht des Angerufenen dahingehend verarbeitet, dass dieser die Gesprächsannahme über die GUI bestätigen muss. Wird diese Bestätigung gegeben, wird mit dem Audiostream geantwortet. Gleichzeitig wird seitens des Angerufenen der Dateideskriptor für ein Telefonat beim Anrufer angefragt, der dann seinerseits unmittelbar mit dem Audiostream antwortet. Die Nutzung der Pipe für einen Chat verläuft ähnlich. Allerdings werden nur dann Daten gesendet, wenn einer der Teilnehmer etwas absendet. Es wird kein kontinuierlicher Datenstrom verwendet.

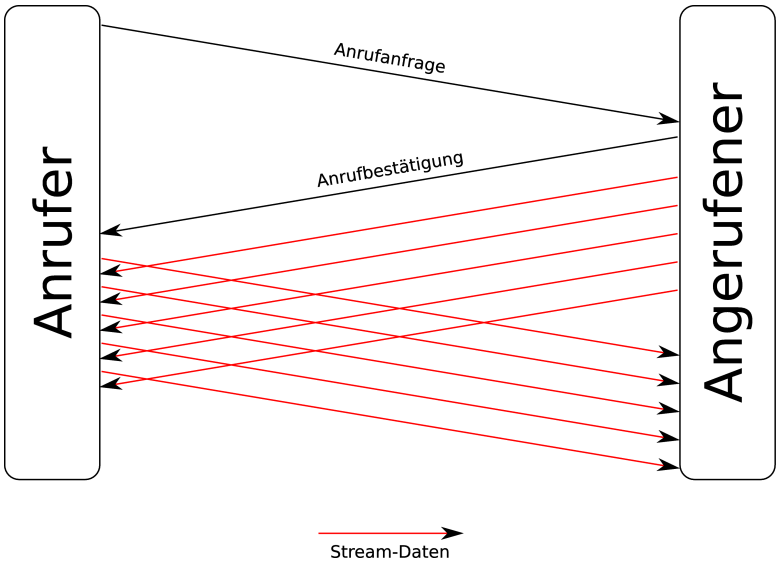

Abbildung 12: Ablauf eines Verbindungsaufbaus mit Datenstreaming

3.4 Datencontainer

Innerhalb eines GNUnet-Pakets mit dem BiDi-Pipe-Datentyp werden die eigentlichen Nutzdaten in einen weiteren Container gekapselt, dessen Nutzung insbesondere bei der Übertragung von Streams zum Tragen kommt. GNUnet hat von sich aus keine Mechanismen, Daten sequenziell zu versenden. Da dies für einen Multimedia-Stream absolut notwendig ist, erhalten unsere Datencontainer eine fortlaufende Nummer, die es dem Empfänger ermöglicht, die Daten auch bei unterschiedlicher Laufzeit korrekt sortieren zu können. Zusätzlich sind im Header unseres Datencontainers ein weiteres Flag für den Datentyp (Multimedia, Chat, Steuerung) und die Größe des Nutzdatenbereichs enthalten.

4 Netzwerk-Konzepte

Um sich über die Probleme der Netzwerk-Konzepte Gedanken zu machen, wurde dieses Kapitel eingefügt.

Im ersten Teil werden drei verschiedene Arten von Peer-to-Peer-Netzwerken erläutert. Bei der Vorstellung von serverbasierten, reinen, und hybriden Peer-to-Peer-Netzwerken werden zusätzlich noch die Stärken und Schwächen dieser Arten aufgezeigt.

Nachdem die drei Arten vorgestellt wurden, werden grundlegende Probleme diskutiert:

Zum einen stellt das Finden eines Peers im Peer-to-Peer-Netzwerk ein Problem dar. Wie kann ein Peer dem Netz beitreten? Wie wird ein Netz aufgebaut? Auf der anderen Seite stellt sich die Frage, was passiert, wenn zwei Netze aufeinander treffen. Was muss beachtet werden, wenn zwei Peer-to-Peer-Netze verschmolzen werden?

4.1 Peer-to-Peer-Netzwerke (Fabian, Markus G.)

Zur Realisierung eines Peer-to-Peer-Netzwerkes muss zunächst die gewünschte Struktur des Netzwerkes geklärt werden, da es verschiedene Arten von Peer-to-Peer-Netzwerken gibt. Diese werden im Folgenden mit ihren Eigenschaften, Vorteilen und Nachteilen kurz vorgestellt:

4.1.1 Serverbasierte Peer-to-Peer-Netzwerke

Eine Ausprägung von Peer-to-Peer-Netzen orientiert sich noch sehr stark an traditionellen Client-Server-Strukturen (z. B. Napster). Hierbei ist ein Server für die Verwaltung von Ressourcen und den Verbindungsaufbau zwischen zwei Peers zuständig. Demnach besteht dieses Netz aus heterogenen Knoten mit unterschiedlichen Aufgaben: Den Peers auf der einen Seite und dem Server auf der anderen Seite.

Serverbasierende Netze sind relativ leicht zu implementieren. Sie sind zudem leicht wartbar und einfach zu administrieren. Allerdings haben sie eine eklatante Schwachstelle: Fällt der Server aus, bricht das gesamte Netz zusammen. Dies macht das Netz extrem angreifbar, aber auch eine große Anzahl an Anfragen können den Server und somit das Netz in die Knie zwingen. Des Weiteren sind die Speicherressourcen des Servers begrenzt. Da auf diesem sowohl die Ressourcen der Peers als auch deren Adresse gespeichert werden müssen, ist die Größe des Netzes durch diesen beschränkt. Ein ebenfalls nicht zu unterschätzender Nachteil liegt in der permanent benötigten Erreichbarkeit und dem damit verbundenen Energiebedarf.

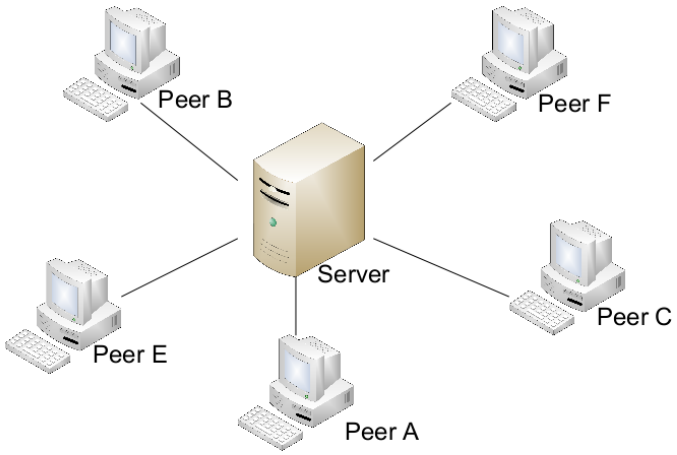

Abbildung 13: Aufbau eines serverbasierten P2P-Netzes

4.1.2 Reine Peer-to-Peer-Netzwerke

In reinen Peer-to-Peer-Netzwerken wird der Grundgedanke des Peer-to-Peer strikt verfolgt. Alle Peers im Netzwerk haben die gleichen Aufgaben, Pflichten und Rechte. Diese Topologie besteht demnach ausschließlich aus homogenen Knoten. Peers können Anfragen stellen und Ressourcen, Dienste oder ähnliches zur Verfügung stellen. Stellt ein nicht unmittelbar benachbarter Peer eine Ressource bereit, so werden Anfragen über die anderen Peers zu diesem weitergeleitet.

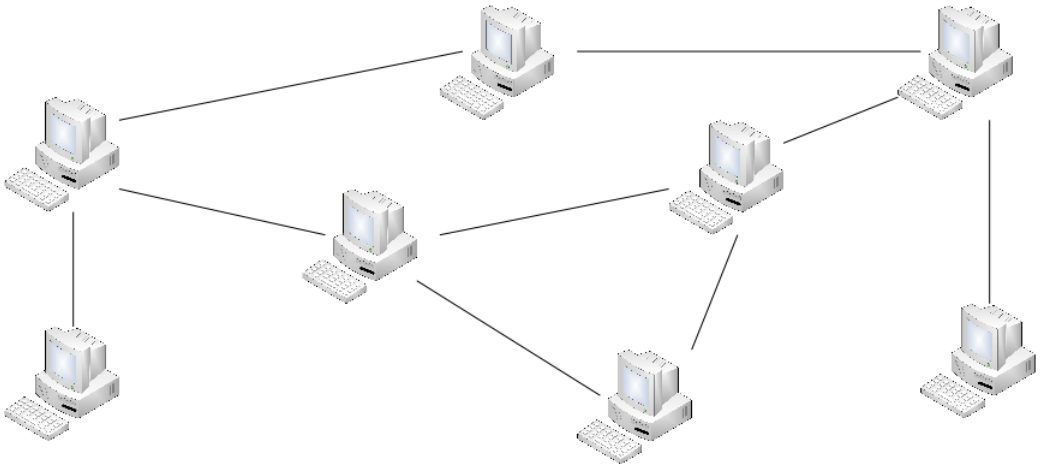

Abbildung 14: Aufbau eines reinen P2P-Netzes

Reine Peer-to-Peer-Netzwerke sind zwar weiterhin angreifbar, jedoch können durch die vollkommen dezentrale Struktur nur selten Schlüsselknoten ausgeschaltet werden. Das

Wegfallen eines Peers lässt somit das bestehende Netz nicht zusammenbrechen und eine hohe Ausfallsicherheit ist gewährleistet. Auch eine hohe Skalierbarkeit ist hierdurch gegeben. Es ist zudem energiesparend, da keine permanent betriebenen Server benötigt werden. Allerdings ist die Wartbarkeit des Netzes nahezu unmöglich. Die Suche nach Ressourcen ist durch die Time-to-live (kurz *TTL*) der Suchanfrage beschränkt. So sind Ressourcen einzelner Peers nicht im gesamten Peer-to-Peer-Netzwerk verfügbar. Auch das Finden vorhandener Peers ist eine Herausforderung, zu der bislang keine optimale Lösung gefunden wurde (siehe Kapitel 4.2).

4.1.3 Hybride Peer-to-Peer-Netzwerke

Bei hybriden Peer-to-Peer-Netzwerken wurde versucht, die Vorteile von reinen und serverbasierten Peer-to-Peer-Netzen zu vereinen und die Nachteile zu minimieren. Daraus ergaben sich zwei Arten von Peers: Endknoten und Superpeers. Die Superpeers bilden das „innere“ Netz eines hybriden Peer-to-Peer-Netzwerks, welches als reines Peer-to-Peer-Netzwerk aufgefasst werden kann. Jeder Endknoten hat eine Verbindung zu einem Superknoten. Dieser fungiert als eine Art Server für diesen Endknoten.

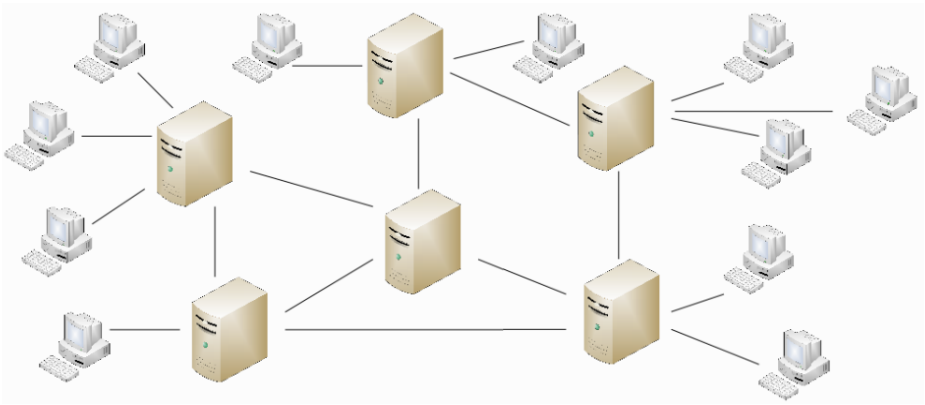

Abbildung 15: Aufbau eines hybriden P2P-Netzes

Ein großer Vorteil eines hybriden Peer-to-Peer-Netzwerks ist die verbesserte Skalierbarkeit im Vergleich zur Skalierung des server-basierten Peer-to-Peer-Netzwerks. Durch die multiplen Superknoten kann das Netz zwar weiterhin von außen angegriffen, jedoch nicht so leicht komplett zerstört werden. Bricht ein Superknoten weg, so bricht nicht das gesamte Netz zusammen. Zudem sind sie relativ leicht zu realisieren. Wie bei einem serverbasierten Peer-to-Peer-Netzwerk können an den Superpeers auch Engpässe entstehen. Weiterhin kann ein Superpeer durch die Vergabe von Prioritäten auch Peers bei der Bearbeitung von Suchergebnissen blockieren. Auch in hybriden Systemen müssen mehrere Superknoten permanent erreichbar sein, was wiederum einen nicht zu vernachlässigenden Energieverbrauch mit sich bringt.

4.1.4 Fazit

Durch Betrachtung der Vor- und Nachteile können folgende Überlegungen festgehalten werden:

Während sich für kleine, lokale Netze reine Peer-to-Peer-Netze gut eignen, sind für größere, globale Netze hybride Peer-to-Peer-Netze von Vorteil.

4.2 Finden vorhandener Knoten eines Peer-to-Peer-Netzwerkes (Fabian, Markus G.)

Alle oben aufgeführten Peer-to-Peer-Arten haben ihre Vor- und Nachteile. Ein Problem ist aber allen gemein: Das Finden des ersten Peers. Dieses hatte entscheidenden Einfluss auf die Entscheidung der Projektgruppe, welche Topologie für das Projekt verwendet werden sollte.

Es war bereits entschieden, dass dieses Projekt zwei autonome Systeme zur Erzeugung von Peer-to-Peer-Netzen beinhalten soll:

Zum einen sollten auf Zeroconf-Grundlage (siehe Kapitel 7) also durch ein reines Peer-to-Peer-Netz lokale Adhoc-Netze und zum anderen Netze über globale Infrastrukturen erzeugt werden können. Bei letzterem war allerdings noch nicht festgelegt, ob es sich um ein serverbasiertes, ein hybrides oder ein reines Peer-to-Peer-Netz handeln sollte.

Das Beitreten zu einem P2P-Netzwerk hängt von dessen Typ ab. Deshalb vergleichen wir in den folgenden Abschnitten die in Kapitel 4.1 aufgeführen Typen im Hinblick auf ihre mögliche Beitrittsmöglichkeiten.

4.2.1 Serverbasierte P2P-Variante und Hybride P2P-Variante

Bei serverbasierten und hybriden P2P-Netzwerken sind a priori die Adressen ständig erreichbarer Server oder Superpeers bekannt (respektive eine Adresse bei einem serverbasierten Netzwerk), zu dem ein Peer problemlos Kontakt aufnehmen kann, um dem betreffenden Netzwerk beizutreten. Somit stellt sich die Frage nach dem Finden des Netzes nur im Falle des Ausfalls des zu kontaktierenden Servers/Superpeers: Bei serverbasierten Netzen ist in diesem Fall schon definitorisch kein Netz vorhanden. In einem hybriden Netzwerk kann ein weiterer bekannter Superpeer angesprochen werden.

4.2.2 Reine P2P-Variante

Im Gegensatz zu serverbasierten und hybriden gibt es in reinen Peer-to-Peer-Netzen keine bekannte Adresse sich im P2P-Netz-befindlicher Peers. Hier gab es mehrere Überlegungen, wie ein erster Peer gefunden werden könnte:

- **Broadcast:** In kleinen, internen Netzen, wie beispielsweise bei unserem lokalen, durch Zeroconf/Avahi-realisierten Netz, bietet eine Broadcastanfrage über das gesamte Netz eine adäquate und legitime Möglichkeit, Peers zu finden. Jedoch ist es nicht unbedingt eine gute Idee, Broadcast-Anfragen in weltweite Netze zu versenden.
- **Dynamische Domain/dynamische „feste“ Adresse:** Die Idee hierzu war, dass es eine feste Adresse oder Domain gibt, unter der ein Peer zu finden ist. Bricht dieser Peer weg, muss entweder ein schon vorhandener Netzteilnehmer oder ein beitretender Peer diese Adresse übernehmen. Daraus resultieren allerdings folgende Probleme:
  - *Generelle Probleme:*
    - Abhängigkeit von externen „dynamischen Domain-Diensten“ - fällt der Dienst aus, ist das Netz nicht zu erreichen.
    - Fällt der Peer, der die dynamische Domain belegt aus, zerfällt das Netz eventuell zeitweise in disjunkte Netze.
  - *Probleme, wenn vorhandener Peer die Domain allokiert:*
    - Thrashing, d.h. die Domain würde gegebenenfalls permanent zwischen den Peers des Netzes wandern.
  - *Probleme, wenn neu beitretender Peer die Domain allokiert:*
    - Tritt kein Peer, nachdem die dynamische Domain frei geworden ist, dem Netz bei, so bleibt dieses Netz, falls es in disjunkte Teilnetze zerfallen ist, nicht-zusammenhängend. Dies ist gerade bei wenigen Teilnehmern ein großes Problem, da bis zum Beitritt kein Peer die bekannte Domain allokiert.

4.2.3 Fazit

Mit unseren sechs Internet-Tablets und einigen, wenigen PCs wollen wir unter anderem ein globales Peer-to-Peer-Netzwerk aufbauen. Demnach war der Aufwand zur Realisierung eines reinen Peer-to-Peer-Netzwerkes zu groß. Die maximale Anzahl der teilnehmenden Peers sprach gegen ein hybrides System. Nach ausführlicher Diskussion und Erörterung von Vorteilen, Nachteilen und Problemen, sowie Klärung mit unseren Betreuern, ob die Entscheidung der Gruppe PG-Ziel-konform sei, wurde somit eine server-basierte Peer-to-Peer-Struktur zur Realisierung eines Peer-to-Peer-Netzes über die globale, bestehende Infrastruktur festgelegt. Im lokalen Netz stellte sich die Frage nach der Findung des ersten Peers nicht, da die von Zeroconf/Avahi (siehe Kapitel 7) zur Verfügung gestellten Dienste dieses bewerkstelligen.

4.3 Verschmelzen von Netzen (Thomas, Michael P.)

4.3.1 Allgemeines zum Netmerge

Die primäre Schnittstelle des Internet Tables N810 von Nokia ist WLAN. Sie basiert auf dem IEEE Standard 802.11, wodurch für unser Projekt viele Vorteile entstanden sind. Zum Beispiel ermöglicht uns dieser Standard den Aufbau von sogenannten Ad-hoc-Netzen. Es handelt sich dabei um Netze, die spontan aufgebaut werden können und keine zentrale Verwaltung benötigen. Es ist also möglich, dass ein existierendes Netz mit mehreren Knoten durch einen weiteren Knoten oder durch ein weiteres Netz erweitert werden kann.

Jedoch ergeben sich aus der dynamischen Struktur weitere Probleme, die abgefangen werden müssen, damit ein Netmerge problemlos ablaufen kann. Eines dieser Probleme, ist die eindeutige Zuweisung von Adressen. Betrachtet man als Beispiel das World Wide Web, so werden die Internet Adressen durch die „Internet Assigned Numbers Authority“, kurz *IANA*, zugewiesen. Diese kümmern sich dann um die weitere Vergabe der einzelnen Adressen, wie beim Einrichten des lokalen Netzwerks oder bei der Einwahl ins Internet. In einem privaten Netzwerk zu Hause sorgt dann ein Router mit einem integrierten DHCP-Server für die eindeutige Adressvergabe. Allerdings gibt es im unserem Projekt Anwendungsfälle, in denen es nicht möglich ist, bzw. der Aufwand zu groß ist, solche eindeutigen Adressen zuzuordnen. Deshalb setzen wir zur Lösung dieses Problems das Protokoll „Zeroconf“ ein, das von der 1999 gegründeten „Zeroconf Working Group“ spezifiziert worden ist. Durch die Verwendung dieses Protokolls stehen uns wichtige Mechanismen zur Verfügung, wie zum Beispiel die automatische Erzeugung einer IP-Adresse, sowie die Überprüfung und die Behandlung von Adresskonflikten.

Im nächsten Abschnitt zeigen wir ein paar Netmerge-Szenarien und gehen dabei auf die Probleme und ihre Lösungen ein. Des Weiteren behandeln wir zuerst die Ad-hoc- und später die Peer-to-Peer-Ebene.

4.3.2 Szenarien beim Netmerge

Wir befinden uns bei der Ausgangssituation in der Ad-hoc-Ebene, siehe Abbildung 16. Es existieren zwei Ad-hoc Netze, Ad-hoc-Netz1 und Ad-hoc-Netz2 mit jeweils einem N810, dargestellt als Knoten A bzw. Knoten B. Der orange gefüllte Kreis symbolisiert jeweils ein einzelnes Netz. Jedem Gerät wurde in der Abbildung eine gültige IP-Adresse durch Zeroconf zugewiesen. Weiterhin wird angenommen, dass sich die Ad-hoc-Netze aufeinander zubewegen.

Zu einem Zeitpunkt X befindet sich der Knoten B in der Reichweite von Knoten A und umgekehrt. In diesem Augenblick sorgt Zeroconf von Knoten B dafür, dass B eine gültige IP-Adresse zugewiesen wird. (B schickt ein spezielles ARP-Paket an A und A leitet es an alle, die er kennt, oder umgekehrt.) Im nächsten Schritt antwortet Knoten A dem

Abbildung 16: Netmerge: Ausgangssituation

Knoten B, dass seine IP-Adresse eindeutig im neuem Netzwerk ist. Sollte dies nicht der Fall sein, generiert Zeroconf des Knotens B eine neue IP-Adresse. Danach werden wieder, wie oben beschrieben, die ARP-Pakete an den Knoten A verschickt und durch Knoten A beantwortet. Am Ende dieses Vorgangs haben die Knoten A und B jeweils eine gültige IP-Adresse und die beiden Ad-hoc-Netze wurden zu einen Netz vereinigt, siehe Abbildung 17.

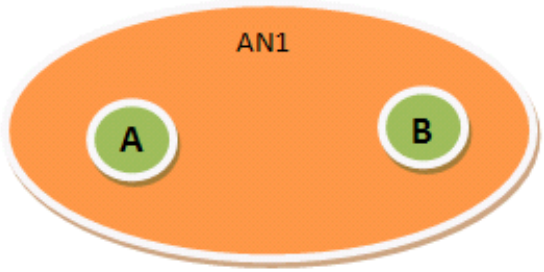

Abbildung 17: Netmerge: Knoten A sieht Knoten B

4.3.3 Weiteres Szenario

Kommen wir nun zu einem weiteren Szenario, bei dem die Ausgangssituation in der Abbildung 18 dargestellt wird.

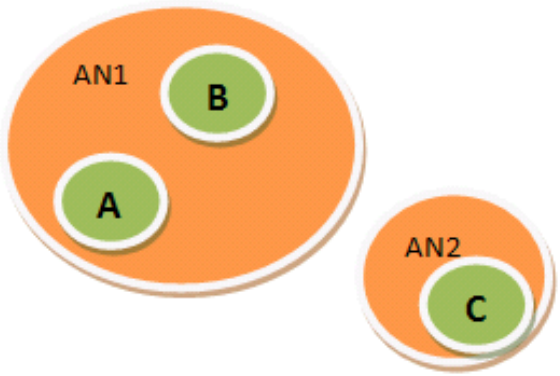

Abbildung 18: Netmerge: Ad-hoc-Netz1 (Knoten A und B) und Ad-hoc-Netz2 (Knoten C)

Es existiert ein Ad-hoc-Netz mit jeweils zwei Knoten A und B. In diesem Netz sind bereits eindeutige IP-Adressen durch Zeroconf vergeben worden. Zusätzlich existiert ein weiteres Ad-hoc-Netz, das aber nur einen Knoten C enthält. Weiterhin nehmen wir an, dass sich die Netze aufeinander zubewegen, und dass der Knoten C in die Reichweite des Knotens B kommt. Im nächsten Schritt wird ein neues Ad-hoc-Netz aufgebaut, welches die Knoten B und C enthält: siehe Abbildung 19. Folgend existieren zwei Ad-hoc-Netzwerke, Ad-hoc-Netz1 mit den Knoten A und B und Ad-hoc-Netz2 mit den Knoten B und C, wobei Knoten A und C sich nicht gegenseitig sehen können. Somit hat Knoten B die Aufgabe, die Nachrichten, die von Knoten A an C (oder umgekehrt) gesendet werden, weiterzuleiten.

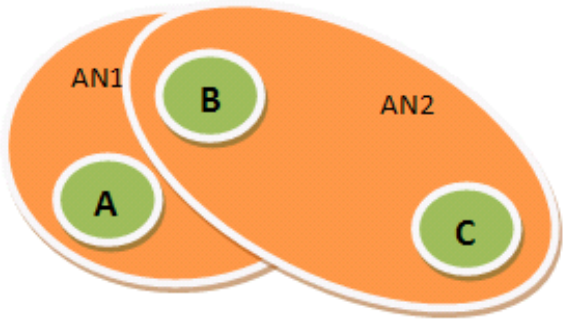

Abbildung 19: Netmerge: Zwei Ad-hoc-Netze

In Abbildung 20 wird ein P2P-Netz betrachtet, wo Peer A und Peer D die gleiche IP-Adresse besitzen. In diesem Fall gibt es keinen Konflikt, da kein Netzteilnehmer existiert, der beide Peers direkt erreichen kann, und somit auf unterschiedliche IP-Adressen angewiesen ist. Die eigentliche Nutzdatenübertragung inkl. Routing findet auf der Applikationsebene statt. Dort werden Hashwerte zur Identifizierung der Peers verwendet. Diese sollten mit hoher Wahrscheinlichkeit auch auf den Peers A und D unterschiedlich sein.

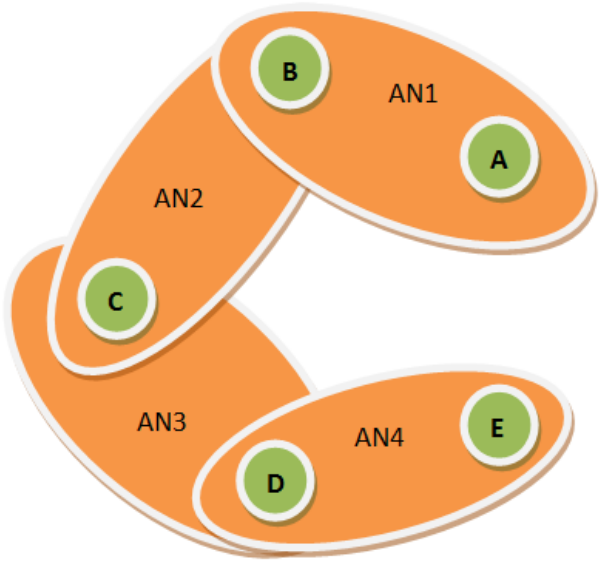

Abbildung 20: Netmerge: P2P Netz

5 P2P Frameworks

Nachdem bisher die theoretischen Netzwerk-Konzepte betrachtet wurden, muss eine Implementierung mittels eines geeigneten Frameworks gefunden werden, welches unseren Anforderungen entspricht. In diesem Kapitel werden verschiedene Frameworks betrachtet und analysiert. Das Hauptaugenmerk liegt hierbei auf JXTA und GNUnet. Es wird ausführlich geklärt welche Vor- und Nachteile beide besitzen. Schlussendlich werden beide direkt gegenübergestellt und es wird eine Entscheidung getroffen mit welchem Framework die Arbeit der Projektgruppe UbiMuC fortgesetzt wird.

5.1 Detaillierter Blick auf JXTA (Michael, Markus S.)

Aufbauend auf der zuvor erstellten Seminararbeit zu JXTA[6] blieben immer noch einige weitere Fragen in Bezug auf JXTA offen. Die Seminararbeit beschränkte sich hauptsächlich auf den theoretischen Hintergrund, den es nun zu vertiefen und zu evaluieren galt. Wichtige Eckpunkte waren hierbei, Erfahrungen in Bezug auf die C-Implementierung zu sammeln.

Bei ersten Recherchen haben wir auf der Projektseite von JXTA-C[7] zwei verschiedene Implementierungen gefunden. Dabei handelte es sich um eine Windows- und eine Linux-Version. Zum Zweck schneller Testmöglichkeiten haben wir zunächst die Windows-Version betrachtet. Diese war zwar leicht zu installieren, stellte sich jedoch als stark veraltet und fehlerbehaftet heraus. Aus diesem Grund war es nicht möglich, die mitgelieferten Testszenarien durchzuspielen. Da UbiMuC ohnehin auf einem Linuxsystem laufen soll, haben wir die Windows-Implementierung nicht weiter betrachtet.

Der Fokus wurde nun also auf die unter Linux lauffähige Version gelegt. Da für JXTA-C keine fertigen Pakete im Debian Paketmanager vorhanden sind, mussten wir den Quellcode selber kompilieren. Dafür müssen zunächst einige Abhängigkeiten aufgelöst werden:

- OpenSSL
- zlib
- sqlite3 (mit Quellcodepatch)
- apr
- apr-utils
- libxml2

OpenSSL, zlib und libxml2 sollten standardmäßig bei jeder Linuxdistribution enthalten sein. SQLite implementiert eine einfache Datenbankengine. APR steht für *Apache Portable Runtime* und ist eine plattformunabhägige Bibliothek, die Standardfunktionalitäten für darunterliegende plattformspezifische Implementierungen bereitstellt. Insgesamt

brauchen alle Abhängigkeiten zusammen ca. 15 MB an zusätzlichem Speicher, allerdings sind diese Abhängigkeiten auch alle unverzichtbar.

Nachdem JXTA-C erfolgreich in das System eingebunden wurde, versuchten wir mit dem mitgelieferten Kommandozeilenprogramm JXTAshell, die Testszenarien nochmals auszuführen. Das erste Szenario umfasste dabei das Erstellen und Finden von zwei Peers in einem JXTA-Netzwerk. Hierbei hat die JXTAshell ständig die Fehlermeldung „Deprecated API“ ausgegeben. Dennoch erzielte der Test mit zwei Peers die gewünschten Ergebnisse. Bei der Erweiterung auf mehr als 2 Peers stellten sich grundlegende Probleme mit der Funktionalität heraus. Zwar konnte der Rendezvous-Peer mit allen mit ihm verbundenen Peers eine Verbindung aufnehmen, weitergeleitete Anfragen über den Rendezvous-Peer waren jedoch nicht möglich. Auch eine Neuinstallation, ein Update auf die neuste Version und eine Änderung der Konfigurationsdateien haben weder die Fehlermeldungen beseitigt noch die Probleme mit fehlerhaften Funktionalitäten behoben. Ein fortführendes Testszenario ermöglichte direkte Kommunikation mittels einer Talk-Session zwischen zwei Peers. Auf Grund der beschriebenen Fehler war auch hier eine Erweiterung auf mehrere Peers nicht möglich. Vor allem die Fehlermeldungen bezüglich „Deprecated API“ deuteten bereits auf eine Vernachlässigung des Quellcodes seitens der Entwickler hin. Der Eindruck wurde durch völliges Fehlen von Programmier-Guides auf der Projektseite, Kommentaren im Quellcode u. ä. bestärkt. Dies fiel vor allem deshalb auf, da für die Java-Implementierung eine Vielzahl von Dokumentationen und Programmier-Hilfen bereitgestellt werden.

Unser Fazit aus den Experimenten zeigte deutlich, dass der Fokus der Entwickler auf der Java-Version von JXTA liegt. Da JXTA-C insgesamt sehr unfertig erschien und auch einige Features der Java-Version vermissen lies, wurden für die Implementierung in UbiMuC weitere P2P-Frameworks in Betracht gezogen.

5.2 Detaillierter Blick auf GNUnet (Fabian, Markus G.)

Neben JXTA wurden alternative Peer-to-Peer-Frameworks als Fundament des Projektes gesucht. Hierbei fand sich GNUnet [28], ein Framework, das vor allem anonymes und vor Zensur geschütztes Filesharing als Ziel hat.

Es ist ein quelloffenes Projekt und steht unter der GNU Public License (GPL). Somit war es von der Lizenz her möglich, UbiMuC auf GNUnet-Grundlage zu entwickeln. Als Programmiersprache wird C verwendet. Es fiel zudem durch seine geringen Systemanforderungen auf, die eine mögliche Portierung auf die ARM-Architektur des N810 und die verhältnismäßig geringen Ressourcen des Internet-Tablets suggerierten. Allerdings gab es einige Abhängigkeiten von Paketen, die nicht Out-of-the-Box vom N810 unterstützt wurden.

Technisch gesehen ist GNUnet ein modular aufgebautes Projekt, was den Einsatz auf einem eingebetteten System durch Entfernung und Deaktivierung nicht benötigter Komponenten und dem damit einhergehenden sinkenden Ressourcenverbrauch erleichtert. Auch

Dienste können einzeln aktiviert und deaktiviert werden. Der modulare Aufbau bietet ebenfalls die Möglichkeit eigene Protokolle neben den bereits implementierten (UDP, TCP, HTTP, SMTP) einzubinden. Die direkte Verbindung zwischen zwei Peers wird nur unidirektional hergestellt. Dabei kommt ausschließlich ein Pull-Protokoll zum Einsatz, so dass nur Daten von anderen Peers abgerufen, jedoch nicht absichtlich an andere Peers gesendet werden können (Push-Protokoll).

Das GNUnet-Paket besteht aus einzelnen, unabhängigen Programmen, zum Beispiel zum Freigeben und Suchen von freigegebenen Inhalten. Dateien wird beim Freigeben ein Hashcode zugewiesen, der die Datei höchstwahrscheinlich eindeutig identifiziert. Dieser wird aus verschiedenen Attributen der Datei, wie zum Beispiel der Größe, berechnet. Das eigentliche Hauptprogramm ist ein im Hintergrund ausgeführter Dienst, der schon bei Systemstart geladen werden kann. Die grafische Benutzeroberfläche ist optional und somit leicht austauschbar. Herzstück von GNUnet ist jedoch das Routingprotokoll GAP, das zugleich den Netzwerkverkehr der Peers verschleiert.

Dieses Protokoll wird im folgenden Kapitel 5.2.1 behandelt. Hierauf folgt eine kurze Abhandlung zum in GNUnet implementierten Austausch von Hostlisten (Kapitel 5.2.2). Der detailierte Blick auf GNUnet wird danach mit einem Abschnitt über die Suche nach freigegebenen Inhalten (Kapitel 5.2.3) abgerundet.

5.2.1 Routingfunktionen von GNUnet (Jens, Michael, Markus S.)

Generelles zum Routing

GNUnet setzt zur Kommunikation auf ein Query/Response-artiges System, in welchem Peers unterschiedliche Anfragen (Queries) an andere Peers absenden können. Diese Anfragen werden wiederum mit Responses beantwortet. Dabei spielt das Routing eine zentrale Rolle, da die Peers keine vollständige Sicht auf das Netzwerk haben, sondern nur eine Liste von bekannten Nachbarn besitzen. Konkrete Pfade zu diesen kennen die Peers allerdings nicht. Um dem Netzwerk beizutreten, muss man die Adresse eines bekannten Peers haben, der dem neuen Peer seine Hostliste zukommen lässt. Die Hostliste enthält dabei bekannte Peers und dient der anfänglichen Adressverteilung.

Erhält ein Peer eine Query-Anfrage, so hat er unterschiedliche Möglichkeiten zu reagieren: Er kann die Query direkt beantworten, beispielsweise durch Bereitstellung einer angefragten Datei, oder er kann die Anfrage an ihm bekannte Peers weiterleiten. Im Falle der Weiterleitung muss er sich jedoch eine gewisse Zeit lang merken, wessen Query er an welchen Peer geleitet hat. Dies ist nötig, damit die Antwort am Ende den ursprünglichen Peer erreichen kann. Jeder Peer baut also mit zunehmenden Weiterleitungen von Queries eine Art lokale Routingtabelle auf. Sollte er jedoch in einem vorgeschriebenen Zeitfenster keine Antwort auf von ihm weitergeleitete Queries erhalten, so wird der Eintrag verworfen und entfernt. Routenänderungen werden ebenfalls berücksichtigt, da mittels eines Timeouts erkannt werden kann, wenn Peers nicht mehr erreichbar sind. In solch einem Fall wird der Peer einfach aus der Tabelle gestrichen und ist damit faktisch nicht mehr erreichbar.

Durch diese Vorgehensweise wird ein hochgradig anonymes Routing realisiert, da zu Beginn einer Query-Absendung weder die genaue Route noch der eigentliche Endpunkt bekannt sind. Zentral ist dabei vor allen Dingen der Routing-Algorithmus des GAP-Protokolls.

GNUnet Anonymity Protocol

Zur Realisierung eines Routings wurde in GNUnet ein auf den Namen „GAP“ getauftes Protokoll entwickelt. Es handelt sich dabei um einen verschlüsselten Datenaustausch zwischen einzelnen Peers, der dabei zusätzliche Funktionen für anonymen Datenverkehr enthält.

Die Verschlüsselung des Datenverkehrs basiert sowohl auf symmetrischen wie auch asymmetrischen Verfahren. Jeder Peer erhält von GNUnet ein Schlüsselpaar bestehend aus öffentlichem und privatem Schlüssel. Der private Schlüssel ist dabei nur dem Peer selbst bekannt, während der öffentliche Schlüssel zur Verteilung periodisch an andere Peers über das Netz gesendet wird. Damit können nun auch diese Peers in Kontakt mit dem

ursprünglichen Absender treten. Um den Datenaustausch mit einem zweiten Peer zu realisieren, muss also dessen öffentlicher Schlüssel bekannt sein. Mithilfe dieses Schlüssels wird nun erstmalig ein neuer Sitzungsschlüssel für ein symmetrisches Verfahren ausgetauscht. Dieser symmetrische Schlüssel wird im Anschluss für die Verschlüsselung des gesamten Datenverkehrs benutzt und ist so wesentlich performanter als die Nutzung eines asymmetrischen Verfahrens für die komplette Verkehrsverschlüsselung.

Message Padding
Neben der erwähnten Verschlüsselung des Datenverkehrs werden noch weitere Funktionen implementiert, die dafür sorgen, dass inhaltlich verschiedene Datenpakete äußerlich nicht unterscheidbar sind. GAP sorgt durch ein automatisches „Message Padding“ dafür, dass alle Datenpakete auf eine festgelegte Größe aufgestockt werden. Einem Betrachter ist es demnach nicht möglich, anhand der Paketgröße auf die Paketart oder den Inhalt zu schließen.

Covertraffic
Darüber hinaus findet eine Maskierung der Pakete statt, indem Daten in Form von sog. „Covertraffic“ von den Peers über das Netz ausgesandt werden. Dieser Covertraffic dient auf der einen Seite dazu, bestehende Query/Response-Daten hinter einem festgelegten Volumen von „Hintergrundrauschen“ (durch den Covertraffic) zu verstecken. Es soll einem Betrachter auf diese Weise nicht möglich sein, die tatsächliche Funktion eines Peers (als Routing-Teilnehmer oder Bereitsteller von Daten) zu identifizieren. Als zweite Funktion kann der Covertraffic allerdings auch für eine bessere Verfügbarkeit von Inhalten sorgen. Wenn ein Peer Dateien bereitstellt, kann der Covertraffic mit diesen Inhalten gefüllt werden und von anderen Peers übernommen werden. Das Ausmaß des Covertraffics richtet sich dabei immer nach der von Nutzer eingestellten maximalen verfügbaren Bandbreite und dem Anonymitätsgrad der Inhalte. Es können also Suchanfragen oder Dateidownloads mit unterschiedlich hohen Anforderungen an den Covertraffic ausgesendet werden. Jeder Peer verfügt über einen spezifischen Migrations-Cache, dessen Größe vom Nutzer variiert werden kann. In diesem Cache können Daten von anderen Peers gespeichert werden, so dass die Verbreitung und die Datenraten bei Ladevorgängen erhöht werden können.

Message Delay
Zu guter Letzt kommt als weitere Absicherung noch eine Nachrichteverzögerung hinzu, so dass erhaltene Queries nicht sofort weiterleitet oder beantwortet werden, sondern zufällig innerhalb eines festen Zeitrahmens bearbeitet werden.

Absender-Verschleierung
Um sämtliche Wege und Routen innerhalb des GNUnet-Netzwerkes zu verschleiern, ändert jeder Peer die Absenderadresse eines von ihm „gerouteten“ Pakets. Ein erhaltenes Paket von Peer A wird durch den Empfänger B bearbeitet und erneut abgesendet, so dass es für weitere Peers so aussieht, als hätte B die Nachricht abgesendet. Erhält B nun Antworten auf die Anfrage, muss in der internen Routingtabelle nachgeschaut werden, damit die Antwort A zugeordnet und entsprechend editiert/weitergeleitet werden kann.

Anpassung von GAP
Um GNUnet möglichst reibungslos für das UbiMuC-Projekt einzusetzen, sind einige Anpassungen nötig. Dabei ist es vorteilhaft, dass bereits eine gut geeignete Krypto-Infrastuktur durch GNUnet gegeben ist. Als nachteilig dürfte sich allerdings der Covertraffic ebenso wie das Message-Padding und die Absender-Verschleierung erweisen, da diese Funktionen eine Anonymität realisieren, die für UbiMuC eigentlich gar nicht notwendig ist. Um daher Netzwerkbandbreite, Rechenzeit und auch Energie zu sparen, sind einige Anpassungen des Routings geplant.

Darunter fällt eine Verringerung des Message-Delays, damit Querys möglichst schnell von den Peers beantwortet oder weitergeleitet werden können. Um den Covertraffic und die Verschleierung des Absenders zu unterdrücken, kann eine hartkodierte Anpassung des Anonymitätslevels vorgenommen werden. Als Reaktion auf einen festgelegten Anonymitätsgrad von 0 verschleiert GNUnet weder den Absender, noch wird für die Ausführung der Query gesonderter Covertraffic generiert. In der Konfiguration von GNUnet kann auch das Message-Padding abgeschaltet werden, so dass keine Homogenisierung der Datenpakete stattfindet.

5.2.2 Der Austausch von Hostlisten (Björn, Lutz, Stefan, Nils)

Einleitung

Thema dieses Kapitels ist das hinter dem Hostlistenaustausch stehende Verfahren. Dabei wird auf das Verhalten beim ersten Beitreten des Netzes sowie auf den späteren Austausch von Hostlisten eigegangen. Das generelle Verhalten ist auch im Paper zu GNUnet[31] im Kapitel „3.4 Joining GNet“ erklärt. Es ist zu beachten, dass die Nachrichten in dem Paper als 'HELO' Nachrichten bezeichnet werden, obwohl sie im Sourcecode von GNUnet als HELLO Nachrichten Erwähnung finden.

Allgemein

In GNUnet werden für die Bekanntmachung von sämtlichen Inhalten spezielle Pakete genutzt, die sogenannten „Advertisements“. Diese enthalten sämtliche relevanten Informationen, um die entsprechenden Daten erhalten zu können. Im allgemeinen erfolgt der Austausch der Hostlisten über sogenannte 'HELLO' Nachrichten. Auch diese sind normale Advertisements. HELLO Nachrichten können sowohl verschlüsselt als auch unverschlüsselt sein. Im allgemeinen enthalten sie die normalen Verbindungsinformationen, wie die IP-Adresse und den zugehörigen Port von anderen Clients. Die Nachrichten sind mit einem Timestamp (Ablauf des Eintrags) versehen und müssen signiert werden. Die Standard-Gültigkeitsdauer ist auf 10 Tage festgelegt und kann in der *gnunet_identity_service.h* geändert werden.

Theoretisch erfolgt der Austausch wie folgt (Auszug aus „7.1 UDP layer: confidentiality and authentication protocols“ dem Hauptpapers zu GNUnet[31]):

Der Sender A signiert seine Nachricht mit seinem privatem Schlüssel und baut sich ein Paket mit der signierten Nachricht (S_A) , seinem öffentlichen Schlüssel (A), seiner Netzwerkadresse (@A) und einer Gültigkeitsdauer (t). Dieses Paket wird mit dem öffentlichen Schlüssel von dem Empfänger B verschlüsselt (E_B) und an B verschickt. Wenn der Empfänger das Paket empfangen hat, entpackt er es und überprüft die Signatur des Pakets. Danach fügt er Sender A zusammen mit dessen Adresse, seiner Signatur und dem Timestamp seiner Hostliste hinzu.
Eine Nachricht hat dabei folgendes Format: „E_B(A,S_A(@A,t))“.

Die Funktionen zum Austausch der HELLO-Nachrichten in GNUnet sind in der Datei `src/applications/advertising/advertising.c` definiert.

Beitreten des Netzes

Nach dem Beitreten des Netzes wird der neue Host in die Hostliste des kontaktierten Host aufgenommen. In der Hostliste des beitretenden Clients steht zu der Zeit nur der direkt kontaktierte Client. Weitere Clients werden erst über den späteren Hostlistenaustausch bekannt gemacht.

Späterer Hostlistenaustausch

Alle X Zeiteinheiten wählt der Client aus seiner Hostliste zufällig zwei andere Hosts aus. Dem einen wird dabei nun eine HELLO Nachricht über den anderen Knoten weitergeleitet (sie wird *nicht* neu generiert, es wird nur die alte, vom anderen Knoten empfangene, HELLO Nachricht weitergereicht), so dass sich diese beiden bekannt machen. Allerdings werden die HELLO-Nachrichten nur so lange weitergeleitet, wie der Timestamp noch gültig ist. Wenn der Eintrag abgelaufen ist, darf die HELLO Nachricht nicht weitergeleitet werden.

5.2.3 Die Suche nach Inhalten im Netzwerk mithilfe von Metadaten (Stephan, Fabian, Markus G.)

Was ist der Libextractor?

Der Libextractor ist ein Parser, der Metadaten aus verschiedenen Dateiformaten extrahieren kann. Da er essentieller Bestandteil des GNUnet Paketes ist, haben wir uns mit seiner Funktionsweise, seinem Aufbau und der möglichen Portierung auf das N810 beschäftigt.

Der modulare Aufbau des Libextractors ermöglicht eine anwendungsspezifische Anpassung der zu unterstützenden Dateiformate. Die verschiedenen Dateiformate werden durch Integration von Paketen realisiert, die einzeln aktiviert und deaktiviert werden können. Im Urzustand besitzt der Libextractor eine große Anzahl Abhängigkeiten, die aus der Vielzahl unterstützter Dateiformate resultiert.

Da unser Studienobjekt ein eingebettetes System ist und somit wenige Ressourcen zur Verfügung stehen, ist es unsere Aufgabe, den durch die Abhängigkeiten resultierenden Ressourcenverbrauch zu minimieren.

Im folgenden werden Beispiele von Metadaten gezeigt, die durch Libextractor extrahiert wurden:

```
creation date - 20080930235959
keywords -
producer - pdfTeX-1.10b
creator - LaTeX with beamer class version 3.06
subject -
title - PG 625 - Der Zwischenbericht - Die Präsentation
author - Alle PG-Mitglieder
format - PDF 1,0
mimetype - application/pdf
```

Abbildung 21: Metadaten eines PDF Dokuments

```
format - 128 kbps, 44100 hz, 4m55 stereo
resource-type - MPEG V1
mimetype - audio/mpeg
description - Heiko ist schuld ...
genre - Punk Rock
year - 2008
album - ... und Constantin ist im Urlaub (in der Südsee)
artist - PG UbiMuC
title - Stoßdämpfer
track number - 23/42
content type - Punk Rock
```

Abbildung 22: Metadaten einer Audio Datei

Inhaltssuche

Mit dem resultierenden Wissen über die Funktionsweise des Libextractors wurde der Fokus auf die praktische Implementierung der Inhaltssuche im GNUnet-Framework gelegt.

Dazu wurde eine Testumgebung mit zwei über ein ZeroConf-Netzwerk (siehe Kapitel 7) verbundenen Desktop Rechnern erstellt, auf denen jeweils ein GNUnet-Client lief. Dabei agierte der eine Rechner als Daten zur Verfügung stellender Peer, der andere als suchender Peer.

Im Folgenden beziehen wir uns auf das oben erwähnte Beispiel (Abbildung 22). Dazu wurden eine Reihe von Suchanfragen gestellt. Die folgende Tabelle 1 spiegelt einen repräsentativen Querschnitt unserer Tests sowie unsere Ergebnisse wieder. Ihr ist zu entnehmen, dass die in GNUnet implementierte Suchen nicht nur case-sensitive ist, sondern *nur* nach exakt dem gesuchten Ausdruck sucht. Auch wenn der gesuchte Ausdruck Teil einer Zeichenkette ist, wird er nicht gefunden. Auch Ausdrücke mit Umlauten und Sonderzeichen können nicht gefunden werden.

Anfrage	Ergebnis	vermutliche Ursache für Misserfolg
Heiko ist schuld ...	Datei gefunden	
Stoßdämpfer	Datei nicht gefunden	Umlaute („ä“) und Sonderzeichen („ß“)
pg ubimuc	Datei nicht gefunden	Kontextsensitivität nicht beachtet
PG	Datei nicht gefunden	keine Suche nach Substrings

Tabelle 1: Ergebnisse der Testreihe zur Inhaltssuche

Fazit

Wie oben bereits erwähnt verbraucht der Libextractor durch seine Abhängigkeiten ein immenses Maß an Ressourcen in Form von Speicher, was ihn für unsere Zwecke in seinem Urzustand nur bedingt geeignet erscheinen lässt. Daraus resultiert, dass wir seinen Funktionsumfang auf ein ressourcenfreundliches - aber für unsere Zwecke ausreichendes - Maß reduzieren sollten. Hierbei ist allerdings noch nicht entschieden, welche Dateiformate letztendlich unterstützt werden sollen.

Ein weiteres Problem stellt die Suche dar, welche im aktuellen Zustand nicht die üblich erwartete Funktionalität einer Suche realisiert (Beispiel: Suche nach Substrings). Die Anpassung auf einen allgemein erwarteten Standard (wie zum Beispiel Unterstützung von Teilwortsuche) würde gegebenenfalls den Rahmen der Möglichkeiten der Projektgruppe überschreiten.

Da die Suche nach Metadaten nicht essentiell für das Erreichen des PG Ziels notwendig ist und die oben genannten Probleme nicht trivial lösbar sind, wurde dieses Thema bis auf weiteres zurückgestellt und mit einer niedrigen Priorität versehen.

5.3 Entropy (Björn)

5.3.1 Überblick

ENTROPY steht für *Emerging Network To Reduce Orwellian Potency Yield* und ist, wie der Name bereits vermuten lässt, ein *P2P*-Netz, das mit dem Ziel entwickelt wurde, sich einer möglichen Überwachung des Internets zu entziehen.

Das Funktionsprinzip kann man als „verteilten Webserver“ verstehen. Jeder Teilnehmer des *ENTROPY*-Netzes wird automatisch Teil des Webservers und speichert auch verschlüsselte Datenpakete anderer Benutzer. Jedes Datenpaket ist zwecks Redundanz mehrfach im Netz vorhanden, um auch nach dem Entfernen eines oder mehrerer Hosts noch ein vollständiges Datum zur Verfügung stellen zu können. Da alle Daten verteilt gespeichert werden, kann nun nicht mehr nachvollzogen werden, welcher Teilnehmer eine Datei zum Download anbietet. Wie in *P2P*-Netzen üblich, fallen bei *ENTROPY* *Server*, *Client* und *Router* zusammen, so dass man auch nicht mehr ohne Weiteres feststellen kann, wer eine vollständige Datei heruntergeladen hat, da als *Router* auch Daten für andere Benutzer weitergeleitet werden.

Die Datenpakete werden auf den einzelnen Hosts nicht separat gespeichert, sondern in einer großen, verschlüsselten Containterdatei aufbewahrt, auf die mit anderen Programmen nicht ohne Weiteres zugegriffen werden kann.

5.3.2 Fazit

ENTROPY eignet sich nur bedingt als *P2P*-Framework für unser Projekt, da zum einen die Hauptaufgabe von *ENTROPY* das anonyme Übertragen von Daten ist, was bei uns keine Rolle spielt. Zum anderen ist der Einsatz der Containerdatei sehr problematisch, da sie immer konstant viel Speicherplatz benötigt und man nur bedingt Zugriff auf den Inhalt hat. Desweiteren scheint das Projekt, das sich mit der Weiterentwicklung von *ENTROPY* beschäftigt, eingeschlafen zu sein, da die Projekthomepage vor kurzem Offline gegangen ist.

Das sind die wesentlichen Gründe, warum wir uns gegen den Einsatz von *ENTROPY* als *P2P*-Framework entschieden haben.

5.4 GiFT (Markus S.)

5.4.1 Spezifikationen

GiFT ist eine Daemon-basierte Software für verteiltes File-Sharing, welche die Möglichkeit bietet, verschiedene File-Sharing-Protokolle einzubinden inkl. des für GiFT entwickelten OpenFT-Protokolls. In beiden Abkürzungen steht das FT für FastTrack, da beide Komponenten in Anlehnung an das FastTrack-Protokoll designt wurden, welches z. B. von

Kazaa genutzt wird. GiFT ist hauptsächlich als Client zu bereits existierenden, großen P2P-Netzwerken gedacht. Es ist stark modular aufgebaut und bietet eine klare Trennung in 3 Komponenten: GUI, Programmkern und Protokoll-Plugins. Diese Trennung ermöglicht die Nutzung auf einer Vielzahl von Plattformen. Die Kernkomponente von giFT übernimmt dabei rein die Verwaltung des File-Sharings. Alle Kommunikation zwischen verschiedenen Instanzen wird über die verschiedenen Plugins realisiert. Daher ist zur Nutzung von giFT auch immer mindestens ein aktives Protokoll-Plugin notwendig. Es ist dadurch aber auch möglich, mehrere Protokolle parallel in giFT zu nutzen und damit auf mehrere P2P-Netzwerke gleichzeitig mit demselben Client zuzugreifen. Als GUI gibt es eine Vielzahl von grafischen und Konsolen-basieren Frontends für Windows und Unix bzw. Linux.

5.4.2 Protokoll-Plugins

Die Entwickler von giFT tragen keine Verantwortung für das fehlerfreie Funktionieren der Plugins (abgesehen von ihrem eigens entwickelten openFT). Daher sind viele Protokolle für giFT im Alpha oder Beta-Status und teils auch gar nicht mehr in Entwicklung.

Stabile Protokolle:

- openFT (gift-openFT)
- Fasttrack (gift-Fasttrack)
- Gnutella (gift-gnutella)

Protokolle mit Alpha/Beta-Status:

- OpenNap
- eDonkey
- Soulseek

Details zu den bedeutsamen Protokollen:

OpenFT

OpenFT ist das zu giFT parallel entwickelte Protokoll und bietet Zugang zu dem projekteigenen P2P-Netzwerk. Es ist an die Funktionalität des FastTrack-Protokolls angelehnt, jedoch völlig eigenständig implementiert, da FastTrack vollständig undokumentiert ist. Das Netzwerk basiert auf drei Arten von Knoten: User-, Search- und Index-Knoten. User-Knoten könnten dabei an bis zu drei Search-Knoten Anfragen für Dateien stellen. Die Search-Knoten nehmen Suchanfragen entgegen und leiten diese rekursiv an andere Search-Knoten weiter. Sie sind für bis zu 500 User ausgelegt und erfordern nach Angabe der Entwickler mindestens 128 MB zur Verwaltung der Daten auf dem lokalen Rechner. Index-Knoten verwalten Listen über Such-Knoten und Statistiken innerhalb des Netzwerks.

Fasttrack

Dieses Plugin ist zur Kommunikation mit FastTrack-basierten Netzwerken wie Kazaa gedacht. FastTrack wurde vom gleichen Team entwickelt wie Kazaa, ist jedoch in keinster Weise dokumentiert. Daher implementiert dieses Plugin nur die Basisfunktionalität und Kommunikation zwischen Clients und Superknoten, die per Reverse Engineering rekonsturiert wurde. Gerade der von FastTrack genutzte Hash-Algorithmus UUHash wird oft stark kritisiert, da er zwar sehr schnell arbeitet, jedoch unzureichende Fehlerdetektion bietet.

Gnutella

Dieses Plugin ist zur Kommunikation mit Gnutella-Netzwerken gedacht und ist zusammen mit den Komponenten des giFT-Projekts bei Sourceforge erhältlich. Jedoch war zu diesem Plugin keine Dokumentation auffindbar.

5.4.3 Bewertung

Einer der großen Vorteile dieser Software ist ihre große Flexibilität, die sich aus den austauschbaren, kombinierbaren Protokolle ergibt. Auch der daraus resultierende Multiplattform-Support würde eine Anpassung an unsere Umgebung sehr erleichtern. Letztlich ist es sehr hilfreich, dass die API der giFT Kernkomponenten auf der offiziellen Seite sehr gut dokumentiert ist.

Der große Nachteil ist das sich die Funktionalität - wie bei den meisten betrachteten Programmen - auf reines File-Sharing beschränkt und daher keine von den benötigten Streaming-Funktionalitäten bereitgestellt werden. Die Software ist auch deshalb minder geeignet, weil sie vorrangig darauf ausgelegt ist, als Client für größere, bestehende P2P-Netze auf Computern ohne Ressourcen-Beschränkungen zu laufen. Weiterhin besteht eine große Abhängigkeit des Client von den vielen Third-Party-Plugins, da die Kernkomponente ausschließlich die Verwaltung der Software übernimmt. In diesem Kontext fehlen gerade für open-giFT viele Dokumentationen und man findet viele tote Links auf der Website des Projekts. Auch die letzte Releases und News zum Projekt liegen im Jahre 2004 bzw. 2006 und daher ist davon auszugehen, dass viele Komponenten stark veraltet sind. Somit ist keine aktiver Community-Support zu erwarten. Letztlich wirkt die gesamte offizielle Projektseite an vielen Stellen sehr unfertig und wenig gepflegt.

Fazit

Im Vergleich zu den Alternativen wie GNUnet und JXTA wirkt giFT weitaus weniger nutzbar für unsere Aufgabenstellung, da es nur reines File-Sharing bietet und für eingebettete System schlecht einsetzbar scheint. Außerdem erscheint das Projekt wenig gepflegt, schlecht dokumentiert und die benötigten Plugins sehr unausgereift. Daher wären massive Anpassungen erforderlich, um giFT in unser Projekt einzubinden. Auf Grund besserer Alternativen wurde daher giFT nicht weiter von uns betrachtet.

5.5 Mute (Stephan)

5.5.1 Beschreibung

Bei Mute handelt es sich um ein P2P Netz, welches den Fokus auf den anonymen Austausch von Dateien legt. Dabei kommen einige interessante Techniken zum Einsatz. Ein Beispiel hierfür ist der Routenfindungsalgorithmus, welcher - angelehnt an die Schwarmintelligenz - sich am Verhalten von Ameisen orientiert. Das Projekt basiert stark auf Forschungsergebnissen und stellt auf seiner Internetseite [30] einige sehr anschauliche Dokumente zu Verfügung.

5.5.2 Test

Zusätzlich zum eigentlichen „Programm“ stehen noch verschiedene GUIs zu Verfügung. Die Installation auf einem Debian basierten Linux System funktionierte problemlos. Auch das Einwählen in das Netzwerk, das Finden von Peers, das Suchen nach Dateien und der Download eben dieser funktionierten ohne die Notwendigkeit größerer Einstellungen. Eine genaue Untersuchung zum Datendurchsatz und Ressourcenverbrauch wurde nicht durchgeführt.

5.5.3 Fazit

Mute konzentriert sich auf den anonymen Dateienaustausch im Internet und benötigt hierfür - genauer gesagt für das Finden von Peers - Server, welche das Netz verwalten. Aus diesen beiden Gründen (Fokus auf Filesharing und die Nutzung von Servern) ist das Projekt für unsere Bedürfnisse nicht geeignet.

5.6 Vergleich von JXTA mit GNUnet (Fabian, Markus G., Markus S., Michael)

Nachdem Alternativen für das JXTA-Framework untersucht wurden, stellte sich heraus, dass letztendlich nur zwei Kandidaten als UbiMuC zu Grunde liegendes Peer-to-Peer-Framework in Betracht kamen. Bei diesen beiden Kandidaten handelte es sich um JXTA selbst und GNUnet.

Um eine endgültige Entscheidung für eines der beiden Frameworks zu treffen, wurden beide Aspiranten ausführlich untersucht und verglichen. Dies geschah unter Betrachtung verschiedener Anwendungsfälle und ausgewählter Kriterien, wie z.B. Dokumentation.

Auf Grund von technischen Problemen von JXTA ließen sich weite Teile der beschriebenen Anwendungsfälle nicht praktisch evaluieren. Unsere Ergebnisse beziehen sich daher nur auf die Dokumentation des allgemeinen JXTA-Frameworks.

Nr.	Anwendungsfall / Kriterium	JXTA	GNUnet
1	Daten durchleiten (Routing)	Endpoint Routing Protocol + Pipes	GAP + Pipes
2	Netzwerk beitreten	Rendevouz Peer + Peer Discovery Protocol	initialer Peer Adresse bekannt
3	Suche nach Inhalten	Advertisements	Metadaten
4	Veröffentlichen von Inhalten	Advertisements Peer Discovery	Advertisements
5	Datenstrom Unterstützung (Streaming)	eingeschränkt (mittels Pipes)	nein
6	Behandlung von Ausfällen	Endpoint Routing Protocol	GAP
7	Systemanforderungen	unbekannt	sehr gering
8	Dokumentation	sehr beschränkt	ausführlich
9	Support	nicht vorhanden	Kontakte zu den Entwicklern

	nicht vorhanden
	nicht bekannt
	theoretisch vorhanden
	praktisch überprüft

Tabelle 2: Vergleich JXTA - GNUnet

Im Gegensatz dazu war die Erzeugung fast aller Szenarien bei GNUnet möglich. Diese konnten zudem erfolgreich getestet werden. Die einzige Ausnahme bildete die Umsetzung von Streaming. Zwar konnten teilübertragene Audio-Dateien problemlos abgespielt werden, jedoch ist dieses, den Entwicklern nach, nicht immer gewährleistet. Weiterhin lässt sich die Steuerung des streamenden Geräts durch den Client nicht durch GNUnet realisieren, da die Implementierung der Pipes nur unidirektional vom Server zum Client vorhanden ist. Eine Pipe in Gegenrichtung lässt sich nicht nativ öffnen. Dieses war zum Zeitpunkt der Entscheidung der Projektgruppe für ein bestimmtes Framework nicht bekannt. Vielmehr suggerierten mitgelieferte Hilfsprogramme einen vorhandenen Push-Service, bis nähere Untersuchungen des GAP-Protokolls (siehe Kapitel 5.2.1) dies widerlegten.

Neben diesen Anwendungsfällen wurden - wie der Tabelle 2 zu entnehmen - weitere Kriterien in die Entscheidungsfindung einbezogen. Sowohl die Dokumentation als auch der mögliche Support durch die Entwickler oder Communities bieten eine wertvolle und

unerlässliche Hilfe für die Einarbeitung in die beiden verbleibenden, komplexen Frameworks.

JXTA bot nicht direkt die beschriebenen Funktionalitäten. Auf Grund von fehlendem Support und fehlender Dokumention konnten die Fehler nicht in angemessener Zeit behoben werden. GNUnet hingegen lieferte einen Großteil der gewünschten Funktionalitäten. Auch Dokumentation und Support waren mehr als zufriedenstellend. Daher wurde ein Wechsel von JXTA zu GNUnet durch die Projektgruppenteilnehmer beschlossen.

6 Die Seminarphase und deren Auswertung

Zur Beginn der Projektgruppe wurden insgesamt zwölf Seminare gehalten, um einen groben Überblick über alle Teilgebiete einer mobilen, multimedialen Platform zu gewinnen. Ziel war es insbesondere, die Anwendbarkeit von bestehenden Protokollen und Algorithmen auf die UbiMuC-Umgebung zu erörtern. Dabei wurde besonders viel Aufmerksamkeit auf verschiedene Kommunikations- und Netzwerktechniken sowie deren Absicherung gelegt. So wurden einerseits allgemeine Techniken wie *WLAN* oder *Peer-to-Peer* behandelt. Andererseits wurden aber auch konkrete Implementierungen, beispielsweise das *JXTA P2P-Framework*, die *Zeroconf Adressvergabe* oder die *Secure Socket Layer* getestet.

Ein anderes wichtiges Thema während der Seminarphase waren Multimedia-Frameworks. Hierbei wurden neben Video- und Audio-Codecs und Streaming-Protokollen auch bereits komplette Streaming-Lösungen, wie z. B. der *VideoLAN Media-Player*, samt ihrer Vor- und Nachteile vorgestellt.

Nicht zuletzt wurden aber auch ganz allgemeine Themen, wie die *Programmierung in C und C++*, abgehandelt. Auch die Zusammenfassung der Features des mobilen Endgeräts, das in dieser Projektgruppe verwendet werden soll, des *Nokia N810*, wurde in einem Seminar behandelt.

In diesem Kapitel werden alle zwölf Seminare mit jeweils einem kurzen Fazit zu einer möglichen Verwendbarkeit der Ergebnisse im Rahmen der Projektgruppe vorgestellt.

6.1 P2P (Fabian)

In der Seminararbeit „P2P“ geht es um Arten von Peer-to-Peer-Netzen und deren Umsetzungen. Sowohl die drei verschiedenen Arten („serverbasierte“, „reine“ und „hybride“ Peer-to-Peer-Netze), als auch die Realisierung durch DHT, CAN und andere wurden kurz vorgestellt.

Egal für welche Art von Peer-to-Peer-Netzen wir uns entscheiden, alle haben ihre Vor- und Nachteile. Größere Probleme könnten bei der Handhabbarkeit, der Informationskohärenz, der Fehlertoleranz, der Sicherheit und der Skalierbarkeit auftreten. Auf diese Dinge sollten bei der Implementierung besonders geachtet werden. Sonst bleibt nur festzuhalten, dass Peer-to-Peer-Netze eine gute Struktur für unser Projekt bieten.

6.2 JXTA (Michael K.)

JXTA implementiert die Basisfunktionalitäten eines P2P-Netzwerks. Es besteht aus einer Menge allgemeiner Peer-to-Peer-Protokolle, die es erlauben, dass verschiedene Geräte im Netzwerk miteinander kommunizieren können. Diese Geräte müssen nicht zwangsläufig

handelsübliche Personal Computer sein. Die JXTA-Protokolle sind der Kern des Projektes, da sie unabhängig von Programmiersprache und darunterliegenden Transportprotokollen arbeiten.

Als Fazit der Seminararbeit ist zu erwähnen, dass man auf JXTA hinsichtlich der praktischen Einsetzbarkeit der C-Implementierung einen genaueren Blick werfen sollte. Alle theoretischen Protokolle wirken auf den ersten Blick sehr stimmig und sollten bei der Umsetzung von UbiMuC eine große Hilfe sein. Die Evaluation des Laufzeitverhaltens zwischen JXTA und einer spezialangefertigten P2P-Lösung ergab, dass die Leistungseinbußen marginal und für ein Streaming mittels P2P durchaus geeignet sind. Jedoch ergaben schon die ersten Recherchen den Eindruck, dass JXTA hauptsächlich nur in seiner Java-Version weiterentwickelt wird und die C-Version in seiner Aktualität immer hinterher hängt.

6.3 State-of-the-art Multimedia Codecs und -Standards (Stefan)

Das Seminarthema „State of the Art Multimedia-Codecs und Standards“ beschäftigt sich mit den grundlegenden Codecs im Audio und Videobereich. Zusätzlich wird allgemein auf die Datenkompressionsverfahren eingegangen, die im Multimediabereich angewendet werden.

Im Hinblick auf die Projektgruppe gibt die Seminararbeit einen Einblick in die Materie der Multimedia-Codecs und Standards. Dies wird insofern benötigt, da die Bandbreite für die Kommunikation zwischen den einzelnen Teilnehmern begrenzt ist und daher bei Audio- und Videoübertragung effiziente Codierungsverfahren zum Einsatz kommen müssen.

6.4 Streaming Protokolle (Michael P.)

Das Seminarthema „Streaming Protokolle“ beschäftigt sich mit den grundlegenden Protokollen, die für die Übertragung von audiovisuellen Datenströmen eingesetzt werden. Zu diesen gehören Protokolle wie der Real-time Protokoll (RTP), Real-time Streaming Protokoll (RTSP) und der Hyper Text Transfer Protokoll (HTTP). Es wird zum Einen auf die Funktionsweise der jeweiligen Protokolle und zum Anderen auf die Anwendungsgebiete derselben eingegangen.

Im Rahmen der Projektgruppe spielen diese Protokolle eine wichtige Rolle, weil diese die Übertragung von Audio- und Videodaten zwischen Netzwerkteilnehmern ermöglichen.

6.5 Vergleich, Test und Evaluierung von Streaming-Servern und -Clients (Markus G.)

Im Seminarthema „Vergleich, Test und Evaluierung von Streaming-Servern und -Clients“ wurden die Programme Darwin Streaming Server [8], LScube [9], VLC VideoLan Media-Player [10] und MPlayer [11] auf ihren möglichen Einsatz als Streaming-Server oder -Client in UbiMuC betrachtet. Dazu wurden verschiedene Kriterien wie Speicherverbrauch, Lizenzkompatibilität und Stabilität des Programms betrachtet.

Letztendlich lieferte das Seminarthema mit seinen Tests keinen präferierten Kandidaten, auf dessen Grundlage das UbiMuC-Streaming zu realisieren sei: Da der VLC VideoLan Media-Player schon auf dem Testsystem für einen erheblichen Speicherverbrauch sorgte, erschien der Player für die Verwendung auf einem eingebetteten System und in UbiMuC ungeeignet. Auch die intern verwendeten FFmpeg-Codecs erschienen zumindest beim Einsatz in einigen Ländern aufgrund von Softwarepatenten rechtlich problematisch. Dasselbe Problem existiert bei MPlayer, da auch dieser intern FFmpeg verwendet. Der Darwin Streaming Server erlaubte nur eine Weiterentwicklung unter einer nicht GPL-kompatiblen, Apple-eigenen Open-Source-Lizenz und ist somit nicht für die Verwendung in UbiMuC geeignet. LScube hingegen erwies sich als instabil, was bei einer Software im frühen Entwicklungsstadium nicht ungewöhnlich ist. Die Frage nach der UbiMuC zugrunde liegenden Streamingsoftware musste demnach später im Plenum mit der gesamten Gruppe erörtert werden und führte zu weiterer Recherchearbeit.

6.6 Allgemeine Sicherheitsalgorithmen (Jens)

Die Seminararbeit über „Allgemeine Cryptoverfahren“ beschäftigt sich mit den Unterschieden zwischen symmetrischen und asymmetrischen Verschlüsselungsverfahren und geht dabei auf einige wichtige Algorithmen detailliert ein. Nebem dem häufig benutzten AES wird auch ein Blick auf den Vorgänger DES geworfen. Zusätzlich werden Grundlagen von Public-Key-Systemen angeschnitten und die dortigen Verfahren DSA und RSA betrachtet.

Im Hinblick auf die Projektgruppe gibt die Seminararbeit einen Einblick in die Materie der Verschlüsselungen und Sicherheitskonzepte. Dies ist insofern von Bedeutung, da die Kommunikation zwischen den einzelnen Teilnehmern der UbiMuC-Anwendung geschützt und vertraulich ablaufen soll. Zu diesem Zweck bietet sich die Nutzung der erwähnten Verschlüsselungsmethoden und Konzepte an.

6.7 Sicherheitsalgorithmen für eingebettete Systeme (Markus S.)

Diese Seminararbeit geht auf spezielle Verschlüsselungsverfahren für eingebettete Systeme ein, da man es in diesem Bereich mit anderen Anforderungen und Einschränkungen zu tun bekommt, als bei üblichen Rechensystemen, die von den zuvor dargestellten Verfahren

nur schlecht behandelt werden können. Insbesondere wird auf Public-Key-Verschlüsselung auf Basis von elliptischen Kurven und die zugrundeliegenden mathematischen Konzepte eingegangen.

Durch die genaue Gegenüberstellung der angestrebten Vorteile dieser Verfahren und der mit ihnen verbundenen Einschränkungen stellte sich recht schnell heraus, dass wir sie in unserem Projekt nicht integrieren können. Zwar zielen die Ansätze dieser Verfahren auf eine Reduktion der benötigten Datenbandbreiten für Schlüssel bei gleich bleibender Sicherheit ab. Der Preis, den man dafür zahlt, ist jedoch eine erhöhte Rechenlast auf dem Endgerät. Nach erster Betrachtung des N810 in der zugehörigen Seminararbeit zeichnete sich bereits ab, dass wir schon durch die Umsetzung unser Basisfunktionalitäten einer Multimedia-Streaming-Applikation die Möglichkeiten der Hardware zu einem großen Teil ausreizen. Folglich sind Verschlüsselungsverfahren, die einen besonders hohen Berechnungsaufwand erzeugen, für uns nicht tragbar. Ein weiteres Problem ergab sich durch die rechtlich unklare Lage. Viele Konzepte dieser Verschlüsselungsverfahren sind patentiert, sodass sich die Integration von bereits bestehenden Lösungen unter Einhaltung unserer Software-Lizenzen stellenweise als unmöglich herausstellt.

6.8 Sicherheitsprotokolle und Implementation (Björn)

Die Seminararbeit zum Thema „Sicherheitsprotokolle und Implementationen“ beschäftigt sich mit den *Secure Socket Layer*-Protokollen.

Im ersten Teil wird das allgemeine Verfahren von *SSL* zur Absicherung von streamorientierten Verbindungen mit Hilfe der in den vorherigen Kapiteln beschriebenen Algorithmen zur symmetrischen und asymmetrischen Verschlüsselung beschrieben. Dabei wird auch genauer auf die verschiedenen Schlüsselaustauschverfahren, die *SSL* bietet, eingegangen. Anschließend werden ein paar Einsatzgebiete dieses Sicherheitsprotokolls, wie *HTTP Over TLS* oder *Secure SMTP over Transport Layer Security* vorgestellt.

Der zweite Teil der Arbeit geht auf die *OpenSSL*-Implementierung ein und erläutert die *API* anhand eines kurzen Beispielprogramms. Es werden die wichtigsten Funktionen beschrieben, die zum Aufbau einer *SSL*-Verbindung unbedingt benötigt werden. Anschließend wird kurz auf das `openssl`-Kommandozeilentool eingeganen, dass u. a. zur Schlüsselverwaltung, also beispielsweise dem Erzeugen, Konvertieren oder Widerrufen von Schlüsseln, verwendet werden kann.

Das *OpenSSL-Toolkit* bietet also grundsätzlich alle Funktionen, die wir im Rahmen der PG benötigen, um Verbindungen zwischen zwei Endgeräten abzusichern. Je nach Algorithmenwahl hält sich auch der Ressourcenverauch in Grenzen, was gerade im mobilen Bereich von Vorteil ist. Der wohl entscheidene Nachteil der *SSL*-Protokollfamilie ist die fehlende Unterstützung von paketorientierten Verbindungen, die aber gerade beim Streamen von Multimediadaten eine wichtige Rolle spielt.

6.9 C/C++ (Stephan)

Dieses Seminar hatte die Programmiersprachen C und C++ zum Inhalt. Der Fokus dabei sollte vor allem auf den Unterschieden zu der - an der Uni gelehrten - Programmiersprache Java liegen, da eine Kenntnis eben dieser vorausgesetzt wurde.

Neben den grundlegenden Sprachelementen - welche sich nur unwesentlich von Java unterscheiden - wurde dabei vor allem auf Zeiger, die Speicherverwaltung und die Sichtbarkeit bzw. Gültigkeit von Variablen eingegangen, da dies die - wahrscheinlich - größten Probleme im Umstieg von Java auf C/C++ darstellen, für ein effizientes Arbeiten aber unerlässlich sind.

Unterstützend wurden mehrere Codeschnippsel gezeigt, um den Zuhörern die Sprache näher zu bringen.

6.10 N810 (Nils)

Dieses Seminar beschäftigte sich mit der im Laufe der PG genutzten Hardwareplattform und den auf ihr zur Verfügung stehenden Möglichkeiten. Es wurde die Hardware- und Software-Plattform N810 zuerst grundlegend vorgestellt. Dabei wurde aber auch immer wieder auf für die Projektgruppe relevante Punkte, wie zum Beispiel die verwendete CPU, die Speicherausstattung und bereits vorhandene Software, eingegangen.

Als Fazit dieses Seminarvortrags bleibt zu sagen, dass es sich um eine vielversprechende Plattform handelt, die schon jetzt viele Möglichkeiten auch im Bereich der Kommunikation bietet. Viele gängige Programme sind bereits für das N810 verfügbar, und das Portieren von weiteren unter Linux lauffähigen Bibliotheken und Programmen sollte ohne große Probleme möglich sein. Allerdings ist während der Projektgruppe auch immer zu bedenken, dass es sich um ein eingebettetes System handelt, weshalb entsprechend darauf geachtet werden muss, ob die Hardware genug Rechenleistung und Speicherplatz für die angedachten Anwendungsszenarien bietet. Bei der zu erstellenden grafischen Oberfläche muss außerdem auf die besonderen Eingabegeräte des „Internet-Tablet“ geachtet werden.

6.11 WLAN-Netze (Thomas)

Die Seminararbeit beschäftigte sich mit WLAN-Netzen im Hinblick auf die Nutzbarkeit mit dem Internet Tablet N810 von Nokia. Dabei wurde das Wireless Local Area Network (WLAN) und Bluetooth vorgestellt, wobei der Schwerpunkt mehr auf WLAN lag. Der Grund für das Interesse an WLAN lag erstens an der primären Schnittstelle des N810, die WLAN unterstützt, und zweitens an den wesentlich besseren Vorteilen im Vergleich zu Bluetooth, für unser Projekt. Weiterhin wurde der WLAN Standard IEEE 802.11 und seine Übertragungstechnik, die Funkwellen mit ihren Vor- und Nachteilen, gezeigt. Wichtige Eigenschaften des Standards, wie die Systemarchitektur, Verfahren zur

Übertragung von Signalen und die Sicherheitsmechanismen wurden auch vorgestellt. Des weiteren wurde auf die Bildung von Adhoc-Netzen mit diesem Standard eingegangen.

6.12 Konfigurationsfreie Vernetzung - Zeroconf, Bonjour, mDNS (Lutz)

Die Seminararbeit beschäftigte sich mit den Standards für konfigurationsfreie Netzwerke im Hinblick auf die Nutzbarkeit auf den N810-Geräten. Ein Ziel der Projektgruppe ist es, Netze auch unabhängig von vorhandenen Internet-Zugangspunkten aufbauen zu können. Die Standards für konfigurationsfreie Netzwerke, kurz Zeroconf-Netzwerke, sehen folgende Funktionen vor: Eineindeutige Adresszuweisungen ohne DHCP-Server, bijektive Namenszuweisung und -auflösung, Anbieten und Finden spezieller Netzwerkdienste ohne zentrale Instanz, sowie die Zuweisung von Multicast-Adressen.

Die Notwendigkeit von Adress- und Namenszuweisungen ist offensichtlich gegeben. Die Veröffentlichung und Abfrage von Netzwerk-Diensten kann so genutzt werden, dass die einzelnen P2P-Clients sich als Teilnehmer ausweisen und durch die vorgesehene Suchfunktion mit geringem Overhead gefunden werden können. Da in unseren Planungen lediglich One-To-One-Kommunikation vorgesehen ist, ist die Zuweisung von Multicast-Adressen nicht erforderlich.

Basierend auf den Standards für Zeroconf-Netzwerke existiert mit Avahi eine lizenzkompatible Open-Source-Implementierung, die auch als fertiges Paket für das N810 zur Verfügung steht. Mit dieser können alle von uns benötigten Funktionen der Standards, im Einzelnen die Adress- und Namenszuweisung und die Veröffentlichung und Abfrage von Netzwerk-Diensten, ohne spezielle Anpassungen genutzt werden. Die Nutzung dieses Pakets erscheint sehr aussichtsreich, da neben der benötigten Funktionalität auch eine hohe Flexibilität im Hinblick auf Änderungen in der Netzwerktopologie gegeben ist.

6.13 Abwägung der Bedeutung der einzelnen Seminare für den weiteren Verlauf (Stefan)

Insgesamt haben die unterschiedlichen Themen einen guten Überblick über die verschiedenen Aspekte der Projektarbeit gegeben und zum Teil auch Punkte aufgezeigt, die im Laufe der Projektarbeit näher betrachtet werden müssen. Ebenso haben die Seminararbeiten Themen aufgezeigt, die aufgrund von Hardwareanforderungen nicht durchführbar sind.

Mit den Themen 6.1 (P2P), 6.4 (Streaming-Protokolle), 6.9 (C, C++), 6.10 (N810), 6.11 (Wlan) wurden die technischen und theoretischen Grundlagen gelegt, auf denen die Projektgruppe aufbauen kann.

Die beschränkte Prozessorleistung und der beschränkte Speicherplatz verhindert so den Einsatz der in 6.7 vorgestellten Kryptoverfahren und lenkt das Augenmerk der Projektgruppe eher auf die in 6.6 vorgestellen Verfahren.

Der beschränkte Speicherplatz spielt auch für die Multimediainhalte eine wichtige Rolle und zeigt, dass die in dem Kapitel 6.3 vorgestellen Multimediacodecs unverzichtbar für die Projektarbeit sind, um den hohen Datenstrom bei AV-Inhalten zu reduzieren.

Einschränkungen kommen ebenfalls bei den Streaminganwendungen zu tragen, bei denen sich kein bevorzugtes Programm hervortun konnte, welches die Grundlage für das Streaming liefern kann. Hier wird man auch noch weiter forschen und testen müssen.

Eine weitere Einschränkung ergibt sich in dem Themengebiet von JXTA, welches hauptsächlich auf Java entwickelt wurde. Hier wird das Ziel sein, herauszufinden ob die C Portierung unseren Ansprüchen genügt oder ob eine Alternative gefunden werden muss.

Zum Schluss haben die Themen 6.8 (Sicherheitsprotokolle) und 6.12 (Zeroconf) zwei wichtige Themenbereiche hervorgetan, die im Verlauf der Projektgruppe noch eine wichtige Rolle spielen werden und auf jeden Fall näher betrachtet werden müssen.

7 ZeroConf Netzwerke mithilfe von Avahi

In diesem Kapitel wird das Protokoll Zeroconf (Zero Configuration Networking, auch Automatic Private IP Addressing, kurz APIPA, oder Auto-IP) vorgestellt. Zeroconf ist eine Technik zur konfigurationsfreien Vernetzung von Geräten in lokalen Rechnernetzen. Somit ist eine dezentrale automatische Konfiguration von Netzwerken möglich. Die IETF hat sich mit Zeroconf zum Ziel gesetzt, aufbauend auf dem proprietär von Apple vorhandenen Rendezvous-Protokoll, einen offenen und gut dokumentierenden Standard zu realisieren. Mit Zeroconf annoncieren Programme ihre Dienste im Netzwerk. Normalerweise muss jeder Anwender, der einen Dienst im Netz nutzen möchte, einen Server oder ein zentrales Verzeichnis kennen, wo die Daten aufgelistet werden. Zeroconf bietet jeden Anbieter eines Dienstes an, seinen eigenen Dienst im Netz bekannt zu machen. Multimedia-, Instant-Messaging und Telefonie-Software ist damit einfacher zu bedienen. Zeroconf verfügt über folgende Mechanismen:

- automatische Zuweisung von IP-Adressen ohne DHCP-Server
- übersetzen von Hostnamen in IP-Adressen ohne DNS-Server
- automatisches Finden von Diensten im lokalen Netzwerk ohne einen zentralen Directory-Server

Im Rahmen der Projektgruppe wurde Avahi als eine freie Zeroconf-Implementierung, die heutzutage in allen Linux-Distributionen Standard ist, ausgewählt.

7.1 ZeroConf Netzwerke und P2P-Frameworks (Lutz, Thomas)

Eines unserer Ziele ist es, eine mobile Plattform zu erstellen, auf der man spontan Daten austauschen kann. Auf Netzwerk-Ebene bieten sich hierfür Ad-Hoc-Netze an. Hier besteht allerdings das Problem, dass es keine zentrale Verwaltungsinstanz für IP-Adressen und für das Routing gibt. Deswegen sind wir auf eine Möglichkeit angewiesen, die eine Kommunikation ohne eine solche zentrale Instanz möglich macht. Zeroconf bietet uns diese Möglichkeit. Es existieren eine Reihe von Zeroconf-Implementierungen. Angefangen von der Referenzimplementierung von Apple (Bonjour) bis zur GPL-lizensierten Implementierung Avahi. Diese kommt für unser Projekt in Frage, da sie

- lizenzkompatibel,
- in den Maemo-Repositories verfügbar und
- im Quelltext verfügbar

ist. Zur Installation des Avahi-Pakets musste das N810 in den Entwicklermodus (Red-Pill-Modus) versetzt werden, damit das notwendige Repository verfügbar gemacht werden konnte.

Durch die Mechanismen zur Vergabe von IP-Adressen besteht für ein Peer-to-Peer-Framework eine einfache Möglichkeit, verfügbare Peers zu verwalten und die Kontaktdaten

dauerhaft konsistent zu halten. Für ein Peer-to-Peer-Netzwerk ist die Möglichkeit der Dienstenutzung sehr interessant, da hierüber der Austausch von Nachrichten und/oder Metadaten der zu verteilenden Dateien schnell und unkompliziert möglich ist.

Details zur Implementierung von Avahi in unser Framework finden sich in Kapitel 8.2.

7.2 ZeroConf Netzwerke mit dem N810 nutzen (Lutz, Thomas)

Erste Gehversuche

Für erste Verbindungstests wurde neben einem N810 ein Notebook herangezogen, auf dem ebenfalls eine Avahi-Implementierung installiert wurde. Zunächst wurde auf manuelle Weise ein Ad-Hoc-Netzwerk zwischen beiden Geräten aufgebaut, indem beide Geräte auf die selbe SSID und den selben Kanal eingestellt wurden. Anschließend wurden die Avahi-Clients gestartet. Die IP-Zuweisung verlief dem Standard entsprechend ohne Probleme. Allerdings war ein Verbindungsaufbau zunächst nicht möglich. Eine Analyse mit Wireshark ergab, dass die Pakete an ein nicht vorhandenes Gateway geschickt worden sind, weil die Routingtabelle auf dem Notebook nicht korrekt war. Nach der Korrektur der Routingtabelle war eine Kommunikation zwischen den beiden Geräten möglich und vom N810 aus konnte zu Testzwecken eine ssh-Sitzung am Notebook gestartet werden.

Reine N810-Umgebungen

Als nächsten Schritt haben wir dann Avahi in reinen N810-Umgebungen getestet. Ein erster Versuch mit zwei Geräten verlief erfolgreich. Die Geräte konnten nach Aufbau des Ad-Hoc-Netzes und Start des Avahi-Clients unmittelbar miteinander kommunizieren. Auch ein erster Test mit der Veröffentlichung und Suche von Diensten verlief erfolgreich. Erwähnenswert ist auch, dass alle Avahi-Programme im User Space laufen, das heißt, es sind keine root-Rechte zur Ausführung notwendig.

Eine Skalierung des Versuches auf größere Umgebungen ergab, dass die einzelnen Geräte innerhalb der jeweiligen physikalischen Reichweite sein müssen, damit eine Kommunikation möglich ist. Verlässt ein Client das Netz, bzw. die Reichweite des Netzes, wird es von den anderen Clients dadurch erkannt, dass entweder keine Antworten auf Anfragen mehr kommen, oder dass die Lebensdauer des Kontaktes im Cache abläuft. Eine Weiterleitung über eine Zwischenstation ist nicht vorgesehen, um die Netzstabilität zu gewährleisten. Würden Pakete weitergeleitet und wäre der Zielclient nicht mehr im Netz, würde das Netz mit hoher Wahrscheinlichkeit mit Nachrichten, die kein Ziel finden, überflutet werden.

Für die weitere Arbeit müssen Konzepte gefunden werden, wie Daten über die physikalische Reichweite der Geräte hinaus verteilt werden können.

8 Unsere Arbeiten an GNUnet

Nachdem im vorherigen Kapitel die zur Auswahl stehenden P2P-Frameworks detailliert behandelt wurden, befassen sich die folgenden Seiten ausschließlich mit GNUnet. Darunter fällt neben der anfänglichen Portierung der GNUnet-Umgebung auf das N810 auch die Einbindung in unser Repository. Die erste wesentliche Anpassung von GNUnet findet später durch die Verbindung mit Avahi statt, so dass GNUnet nun auch mit Ad-Hoc-Netzen arbeiten kann. Um einen Einblick in die Arbeitsweise von GNUnet zu gewähren, werfen wir außerdem einen Blick auf dessen Routing-Funktionen und diverse Mechanismen zur Gewährleistung der Anonymität innerhalb des Netzwerkes. Die letzten Unterkapitel behandeln abschließend die Verbindungsaufbau-Phase von GNUnet sowie Schilderung einiger Basisfunktionen wie Inhaltssuche im Netzwerk und unser Konzept einer eigenen Personensuche.

8.1 GNUnet auf dem N810 (Björn)

Nachdem wir uns für *GNUnet* als P2P-Framework entschieden haben, begannen die Portierungsarbeiten der *GNUnet*-Programme. Zum einen benötigten wir eine auf der Maemo-Architektur des N810s lauffähige Version von *GNUnet*. Zum anderen mussten wir aber auch die beschränkten Ressourcen im Auge behalten.

Portierung von *GNUnet*

Die Portierung begann damit, sämtliche Abhängigkeiten von *GNUnet* zu portieren. Für die meisten Pakete existierte im offiziellen Maemo-Repository keine Version für das N810. Doch glücklicherweise waren fast alle Pakete im Debian-Repository vorhanden, was eine Portierung erleichterte. Bei den meisten Debian-Paketen reichte es aus, die Anweisungen für die Buildtools (*dpkg-buildpackage*) an die Scratchbox anzupassen. Einige wenige Pakete (z. B. *libextractor*) hatten allerdings so viele weitere Abhängigkeiten, dass zusätzlich noch Übersetzungszeitoptionen modifiziert werden mussten, um beispielsweise nicht benötigte Features zu entfernen, was die Zahl der Abhängigkeiten erheblich reduzierte.

Die Portierung von *GNUnet* war deutlich aufwändiger. Wir haben uns dagegen entschieden, auf das bereits existierende Debian-Paket aufzusetzen, da es sich zum einen um eine zu alte Version handelte und zum anderen noch eine Konfiguration zur Installationszeit benötigte, die auf dem N810 nicht erwünscht ist. Unser von Grund auf neu erzeugtes Paket installiert jetzt eine für unsere Zwecke geeignete Standardkonfiguration, die im Debian-Paket nicht vorhanden war. So konnten wir auch auf das GTK-basierte Konfigurationstool von *GNUnet* verzichten.

Das UbiMuC-Repository

Da allein für *GNUnet* schon dutzende Pakete benötigt werden, wird der Aufwand für das manuelle Installieren der *DEB*-Pakete auf Dauer zu groß. Vor allem ist das während der aktiven Entwicklung der Fall, wo ständig neue Pakete erstellt und installiert werden müssen. Da die Maemo-Architektur auch *apt-get*-basiert ist, bot es sich an, ein eigenes *apt-get*-Repository zu erstellen, um die Installation von Paketen inkl. aller Abhängigkeiten zu vereinfachen. Die fertigen Pakete werden dazu in eine definierte Verzeichnisstruktur kopiert. Anschließend werden mit dem Tool *apt-ftparchive* die Repositoy-Metadaten erzeugt.

Um den Zugriff auf das Repository weiter zu vereinfachen, wurde die gesamte Verzeichnisstruktur unter *http://ls12-www.cs.tu-dortmund.de/ubimuc/repository/* freigegeben, so dass der Paketmanager des N810s direkt darauf zugreifen kann. Die Architektur des *apt-get*-Repositories ermöglicht es, dass unser Repository nahtlos ins System integriert wird. Es ist also ohne Weiteres möglich, dass die Installation unserer eigenen Pakete die Installation von Abhängigkeiten aus dem offiziellen Maemo-Repository anstoßen kann.

8.2 Die Einbindung von Avahi in GNUnet (Björn, Lutz)

Einleitung

Dieses Kapitel beschäftigt sich mit der Nutzung von Avahi-Funktionalitäten im Rahmen des GNUnet-Frameworks. Es werden im einzelnen die Avahi-Elemente beschrieben, die von uns genutzt werden, und an welchen Stellen sie eingebunden werden. Der Aufbau eines GNUnet-Netzwerkes in einem konfigurationsfreien Netz wird durch die Mittel, die Avahi zur Verfügung stellt, erheblich vereinfacht und beschleunigt. Die Avahi-Funktionen werden als neue Teilapplikation in das Programm eingebunden und basieren auf Beispieldateien des Avahi-Projektes.

Motivation

GNUnet bietet von Haus aus keine Mittel, Peers zu finden, ohne einen zuvor definierten Server ansprechen zu müssen. Dieser Server muss vor der Laufzeit in einer Konfigurationsdatei eingetragen werden. Dies ist zu statisch für die Dynamik und Flexibilität konfigurationsfreier Netzwerke.

Avahi als Linux-Implementierung von Zeroconf enthält sowohl die Adress- und Namenszuweisung, als auch die Verwaltung von Netzwerkdiensten. Da wir Avahi für Ersteres bereits nutzen, liegt die Einbindung von Netzwerkdiensten nahe, da keine neuen Abhängigkeiten mit weiteren Paketen hinzukommen.

Ablauf

Im aktuellen Entwicklungsstand gehen wir davon aus, dass das Netzwerk bereits aufgebaut wurde und die Adresszuweisung bereits stattgefunden hat. Zentrales Element des GNUnet-Netzaufbaus sind nun die Netzwerkdienste aus dem Zeroconf-Standard. Durch die Möglichkeit, Netzwerkdienste schnell und mit wenig Overhead finden zu können, wird das Problem, den ersten Peer zu finden, wesentlich vereinfacht. An Stelle einer Anfrage an einen zentralen Server wird nun einfach nach einem spezifischen Netzwerkdienst gesucht. Gemäß Spezifikation beantworten nun alle Netzteilnehmer diese Anfrage, sofern sie diesen Dienst anbieten. Damit bekommt der neu hinzugekommene Peer direkt eine komplette Liste aller unmittelbar erreichbaren Peers, die er kontaktieren kann.

Vor der Suche nach anderen GNUnet-Peers wird zunächst ein eigener Dienst erstellt, damit er selbst auch von den übrigen Geräten gefunden werden kann. Durch die Möglichkeit, mittels Avahi das Netzwerk permanent nach vorhandenen Peers durchsuchen zu können, kann eine sehr hohe Netzstabilität gewährleistet werden.

Technischer Ablauf

Um GNUnet so flexibel wie möglich zu halten, haben wir uns dazu entschieden, Avahi als Plugin zu realisieren. Dies hat den Vorteil, dass es erst bei Bedarf zur Laufzeit geladen und verwendet wird.

In der Initialisierungsphase von GNUnet wird auch die Avahi-Applikation initialisiert und ein Callback-Thread gestartet, der dann die einzelnen Funktionen aufruft.

Zunächst erstellt der Callback-Thread einen Netzwerkdienst, der aus dem Gerätenamen, der Protokollfolge `_gnunet._tcp`, sowie der verwendeten Portnummer 8080 besteht. Die *libavahi* kümmert sich um die Veröffentlichung des Dienstes im Netzwerk. Sollte es dabei zu einer Namenskollision gekommen sein, wovon bei unserem Setup ab dem zweiten Netzteilnehmer auszugehen ist, wird dieser Fehler abgefangen und ein neuer Versuch mit einem abgewandelten Gerätenamen so lange durchgeführt, bis ein eindeutiger Name gefunden wurde.

Nach Abschluss der Initialisierung wird der Service-Browser gestartet, der das Netz anhaltend nach GNUnet-Diensten absucht, indem er nach der Protokollfolge `_gnunet._tcp` sucht. Die teilnehmenden Peers liefern nun eine Datenstruktur zurück, die einen Verbindungsaufbau ermöglicht. Diese Datenstruktur enthält einerseits den kompletten Gerätenamen, was die eindeutige Identifikation der übrigen Peers während einer Sitzung ermöglicht. Weiterhin werden die IP-Adresse und der für unseren Dienst vorgesehenen TCP-Port 8080 übermittelt.

Ausgehend von diesen Daten kann dann von allen Peers der Download der Hostliste angestoßen werden. Dieser Download ist ein gewöhnlicher HTTP-Download, der mit der *libcurl*-Bibliothek durchgeführt wird. Auf der Seite des anderen Peers übernimmt der in

GNUnet enthaltene *libmicrohttpd*-Server die Kommunikation. Auch nach der Initialisierungsphase überwacht der Browser das Netzwerk und verarbeitet neu hinzugekommene Clients unmittelbar. Ebenso werden Peers, die das Netz verlassen, bzw. sich aus der Reichweite des Geräts entfernen, aus der eigenen Liste gestrichen.

Die eigentliche Kommunikation zwischen den Peers bezüglich Dateisuche, Datentransfer und Austausch weiterer Peers läuft dann unabhängig vom Avahi-Modul. Hierfür werden die gewöhnlichen GNUnet-Konstrukte verwendet, da eine weitere Nutzung von Avahi keine weiteren Vorteile brächte ohne massiv in die GNUnet-Strukturen eingreifen zu müssen.

8.3 Die Suche nach Personen mithilfe einer Distributed Hash Table (Markus S., Stefan, Nils)

8.3.1 Einleitung

Die Existenz einer Personensuche ist letztlich wohl die Grundlage für eine Möglichkeit, einen Chat innerhalb von UbiMuC anzubieten. Denn um explizit mit Personen kommunizieren zu können, muss man erst einmal befähigt sein, seine Gegenüber zu finden. In gebräuchlichen Chatanwendungen ist es dabei meist so, dass sich der Nutzer auf einem zentralen Server anmeldet und dieser als Ausgangspunkt genutzt wird, um andere Nutzer zu erreichen. Der Server stellt dann zum Beispiel Räume zur Verfügung (Beispiel: IRC), in denen die Kommunikation stattfindet, oder er hält zumindest vor, welcher (registrierte) Nutzer momentan angemeldet ist, so dass man mit diesem in Kontakt treten kann (Beispiel: ICQ). Dabei ist dem Gesprächspartner entweder eine genaue Nutzerkennung bekannt oder der Gesprächspartner hat einige Profildaten angegeben, nach denen man suchen kann. Auf eine serverbasierte Lösung können wir bei UbiMuC allerdings nicht zurückgreifen, da es sich ja um dedizierte, lokale Netze handeln soll. Entsprechend müssen Lösungen gefunden werden, wie man mit einem dezentralen Ansatz dennoch seine Gesprächspartner finden kann. Wir haben uns dabei für ein Vorgehen mit distributed Hashtables entschieden, welche im folgenden kurz beschrieben werden.

GNUnet benutzt als zentrale Datenstruktur eine verteile Hashtabelle (Distributed Hash Table, DHT). Diese bildet veröffentlichte Daten und deren Inhaber auf Hashwerte ab und verteilt sie automatisch unter allen verfügbaren Knoten. Auf diese Weise wird eine relativ robuste und weitestgehend Knoten-unabhängige Struktur realisiert, da kein Teilnehmer über exklusive Informationen verfügt. Diese Robustheit ist allerdings erst nach einiger Zeit gegeben, da zu Beginn die Informationen natürlich verteilt werden müssen. Auch bleibt es bei der Inhaltsverteilung weiterhin ein Problem, wenn der eigentliche Inhaber der Daten vor der vollständigen Verteilung das Netzwerk verlässt. Sollte aber eine gewisse Zeit vergangen sein, so kann relativ sicher davon ausgegangen werden, dass die nötigen Informationen für einen Dateidownload über die DHT im Netzwerk verteilt worden sind. Jeder Knoten kann dann Anfragen an die DHT stellen und erhält Informationen, welche Knoten er für gewisse Datenblöcke anfragen kann. Dabei muss natürlich berücksichtigt

werden, dass aufgrund der Verteilung im Netzwerk keine 100%ige Aktualität gewährleistet ist.

In diesem Kapitel des Zwischenberichts soll nun dargestellt werden, mit welchen Mechanismen die Personensuche auf der DHT funktionieren soll und welche grundsätzlichen Annahmen und Überlegungen dem zugrundeliegen. Dabei gehen die Überlegungen insbesondere in die Richtung, dass die Daten von gefundenen Kontakten über mehrere Sitzungen persistent sein sollen, so dass es möglich wird, eine Liste mit bekannten Personen aufzustellen, bei denen man dann, sobald man selbst einem Netz beitritt, auch feststellen kann, ob sie gerade verfügbar sind. Des weiteren haben wir als erstes zu erreichendes Ziel eine eins-zu-eins Kommunikation gesetzt, bei der immer genau zwei Kommunikationteilnehmer involviert sind. Dennoch sollte eine spätere Erweiterung auf Mehrnutzerkonferenzen möglich sein.

8.3.2 Grundlagen

Zuerst ist das Problem der persistenten Nutzerkennung zu lösen. Wir haben uns überlegt, dass wir hierfür einfach einen Hash über den durch GNUnet ohnehin vorhandenen public key des Client verwenden. Zwar kann es theoretisch vorkommen, dass zwei Nutzer identische Hashwerte erhalten, doch ist die Wahrscheinlichkeit dafür in der Praxis derart gering, dass sie vernachlässigbar ist, solange man dem Hashwert genug Bits zugesteht. Dies sollte zum Beispiel mit einem 64 Bit Hash schon sichergestellt sein. Vorteil der Nutzung des ohnehin vorhandenen Schlüssels ist, dass kein extra Schlüssel nur für die Personensuche und die Kontaktliste erstellt werden muss, da die Erstellung auf dem N810 einige Zeit beansprucht. Letztlich ist dieser Hash für den Anwender allerdings nicht direkt einsehbar, da er nur für interne Operationen genutzt wird.

Da es letztlich unpraktisch ist, anderen Nutzern den Hash des eigenen Key zu sagen, muss jeder Nutzer noch einen Nickname auswählen, welcher zusammen mit dem Hash des public key und der aktuellen Adresse in der DHT abgelegt wird. So kann der Nutzer anderen seinen Nickname verraten und diese können nach diesem einfach im Netz suchen. Hierbei kommt natürlich das Problem auf, dass es leicht passieren kann, dass mehrere Nutzer den gleichen Nickname wählen wollen. Da wir keinen zentralen Registrierungsserver haben, lässt sich das auch nicht verhindern. Allerdings stellt dies kein Problem dar, da letztlich nur der Hash des public key eindeutig einem Nutzer zuordbar sein muss. Sollte später noch eine Raumfunktion gefragt sein, ist das kein Problem, da GNUnet selbst eine Funktion für „unique namespaces“ zur Verfügung stellt, mit der es möglich ist, jenen Clients mit identisch gewählten Nicknamen leicht andere Namen in der Anzeige zuzuordnen.

Um Nutzern bei der Suche nach anderen Anwendern zu helfen, haben wir uns noch dazu überlegt, freiwillige Profildaten anzubieten. Da kann der Nutzer dann freiwillige Daten, wie zum Beispiel den echten Namen, das Geburtsdatum, die email-Adresse und ähnliches angeben, die sich andere Anwender auf expliziten Wunsch anzeigen lassen können. Diese

werden allerdings nicht in der DHT gespeichert, und nach ihnen kann nicht gesucht werden. Der Grund hierfür ist, dass die DHT nicht mit nur selten benötigten Daten unnötig gefüllt werden soll. Allerdings ist es möglich, dass wenn man nach der Suche nach einem Nickname mehrere Treffer erhält, sich für jeden der Treffer die genaueren Profildaten ansehen kann, um so möglicherweise herauszufiltern, welcher nun der gesuchte Gesprächspartner ist.

Um nun noch eine über mehrere Sitzungen persistente Kontaktliste zu ermöglichen, muss diese natürlich auch auf dem N810 des Anwenders abgelegt werden. In dieser Liste müssen allerdings nur wenige Informationen abgelegt werden. Für jeden Kontakt muss auf jeden Fall der dem Nutzer zugehörige Hash gespeichert werden. Des weiteren kann noch ein vom Nutzer gewählter Nickname gespeichert werden, der in der eigenen Anzeige verwendet wird und gegebenenfalls von dem Nickname, den der Kontakt selbst gewählt hat, abweicht. Dadurch ist es dem Anwender möglich, die Kontakte einfacher den dahinterstehenden Personen zuzuordnen. Andere Informationen müssen für die Kontaktliste nicht abgelegt werden, da ein Nutzer durch den Hash eindeutig identifiziert sein sollte.

8.3.3 Anwendungsszenarien

Hier werden die generellen Überlegungen zu den vorliegenden Anwendungsszenarien dargestellt. In den Grafiken ist dabei zu beachten, dass die DHT, welche ja über alle N810 verteilt ist, abstrakt als Server dargestellt ist. Dies erfolgt einzig, um die Darstellung klarer zu machen und nicht, um irgendwelche Details zu suggerieren.

Änderung des Online/Offline Status

Aufgrund der Funktionsweise der DHT ist es nicht möglich, direkt die entsprechenden Tripel zurück zu bekommen. Es werden von den Teilnehmern der DHT letztlich sämtliche Einträge vom Typ 'Buddylist' zurückgeliefert. Diese muss der Client nun selbst zerlegen und verwalten. Entsprechend wird immer lokal eine (recht) aktuelle Kopie des entsprechenden Teiles der DHT, welcher für die Kontaktliste genutzt wird, vorgehalten. Dadurch sind für sämtliche Suchoperationen keine gesonderten Zugriffe auf die DHT nötig. Es muss nur regelmäßig eine Kopie der DHT geholt werden, zum Beispiel alle 5min. Die genaueren Werte hierfür müssen bei Praxistests ermittelt werden, um sicherzustellen, dass sowohl die auf das Netz ausgeübte Last als auch die lokale Nutzbarkeit der DHT gegeben sind.

Letztlich muss man jeweils bei der lokalen Kopie der 'Buddylist'-Einträge der DHT dann prüfen, wer aus der eigenen Buddyliste auch in der aktuellen DHT aufgeführt ist und entsprechend Online ist. Sämtliche Kontakte, die nicht auftauchen, sind folglich Offline. Problematisch würde es dabei werden, wenn sich in dem Netzwerk mehrere tausend Nutzer aufhalten, da dadurch dann auch der Speicherbedarf für die lokale Kopie des DHT Bereiches entsprechend ansteigt. Solange die Netze allerdings eine Größe von weniger als

einhundert Nutzer haben, stellt dies überhaupt kein Problem dar, da pro Nutzer weniger als 1KB Speicher belegt wird.

Das Verlassen des Netzes wird nicht direkt bemerkt. Allerdings weisen DHT Einträge jeweils einen Timeout Wert auf. So verfallen die Einträge nach einiger Zeit, wenn sie nicht aktualisiert werden. Deshalb schickt jeder Client regelmäßig Nachrichten an die DHT, dass er noch immer verfügbar ist. Dabei sollten durchaus auch größere Zeiträume, wie zum Beispiel 2 Minuten, für das Senden dieser Put Nachrichten an die DHT ausreichend sein, wenn man nur die Verfallszeit der Einträge entsprechend wählt (zum Beispiel 5 Minuten).

Client betritt das Netz

Wenn ein Client dem Netz beitritt, muss die DHT aktualisiert werden und andere Nutzer müssen erfahren, dass der Kontakt nun online ist. Eine mögliche Lösung dafür ist in der Abbildung 23 dargestellt. Die erste Aktion beim Betreten des Netzes ist, dass der Nutzer an die DHT eine Anfrage sendet, so dass er mit dem eigenen Hash, dem gewählten Nickname sowie der Adresse in der Hash Table eingetragen wird. Dadurch können andere Anwender, sobald sie ihre eigenen Online/Offline Daten aktualisiert haben, ihn bei ihren Suchen auf der DHT finden. Da bei Suchanfragen auf der DHT immer sämtliche als Buddylist-Einträge klassifizierte Punkte geliefert werden, macht es kaum Sinn gesondert zu senden, dass der Nutzer nun online ist.

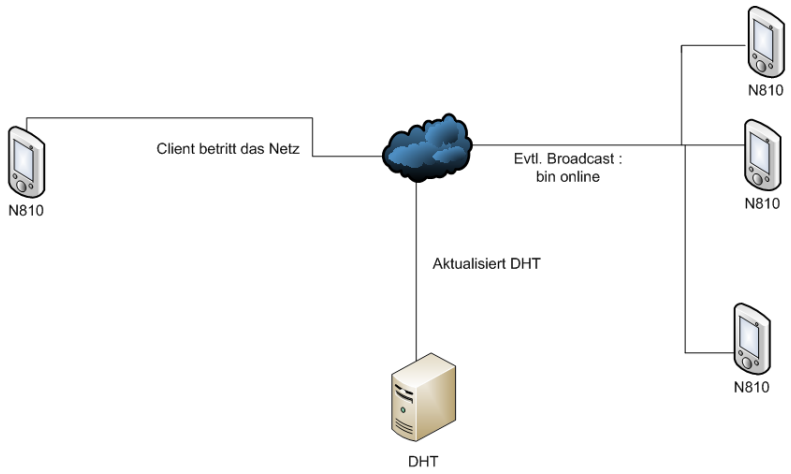

Abbildung 23: Personensuche in GNUnet: Client tritt Netz bei

Allerdings liegt hier aber auch wieder ein mögliches Problem im Zusammenhang mit der zuvor genannten Aktualisierung der lokalen Kopie der Daten. Ist das Aktualisierungsintervall zu groß, so muss beim Betreten des Netzes ein Broadcast erfolgen, so dass wir sicher sind, dass es nicht 10 Minuten dauert, bis bekannt ist, dass der Nutzer nun online ist. Dabei muss dann jeder Nutzer, der den Broadcast erhält, den neuen Client in seiner

Kopie des entsprechenden Teiles der DHT einfügen und den Broadcast auch weiterleiten. Dieses Broadcast ist rechts in Abbildung 23 als mögliche optionale Operation dargestellt. Solch ein Broadcast ist allerdings eine teure Operation, weshalb wir das genaue Vorgehen erst später in der Feinabstimmung festlegen können.

Client sucht nach Personen

Um eine Kommunikation mit bisher noch nicht in der Kontaktliste aufgeführten Personen zu ermöglichen, ist es zuerst nötig, nach den Kontakten zu suchen. Wie bereits erwähnt ist es hier einfach möglich, die lokale Kopie der 'Buddylist'-Einträge der DHT zu nutzen. So wird in der lokalen Kopie gesucht (in Abbildung 24 nur als 'DHT' dargestellt) und die Werte für zu der Suchanfrage passende Ergebnisse werden zurückgeliefert und dem Nutzer angezeigt. Nun kann der Anwender einzeln für jeden angezeigten Eintrag weitere Profildaten anzeigen lassen und auf Wunsch den gefundenen Nutzer in der eigenen Kontaktliste einfügen.

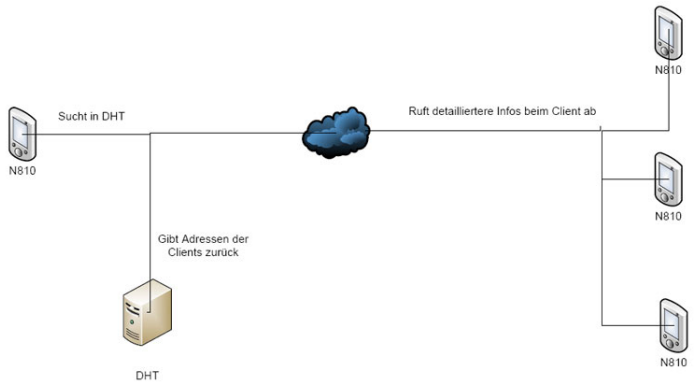

Abbildung 24: Personensuche in GNUnet: Client sucht nach Personen

8.3.4 Fazit

Letztlich wird die DHT einfach nur als verteilte Speichermöglichkeit genutzt, in der die aktuell anwesenden Nutzer abgelegt sind. Was in gebräuchlichen Netzen von einem dedizierten Server erledigt wird, wird hier zum Teil auf die DHT zum anderen Teil auf die Nutzer verteilt. Aufgrund der Funktionsweise, der von GNUnet zur Verfügung gestellten DHT, ist es nicht möglich, sämtliche Operationen immer an die DHT zu schicken. Man braucht letztlich eine lokale Kopie der zum erfragten Typ entsprechenden Einträge, auf der dann die Arbeit erledigt wird. Dadurch ist es allerdings jedem Nutzer möglich, die Daten ohne zentralen Server abzurufen. Das Arbeiten auf der lokalen Kopie der Daten verringert dabei die Last auf das Netzwerk, da wie zuvor beschreiben bei den meisten Anwendungsszenarien keine expliziten Anfragen an die DHT und damit das Netz gehen, sondern einzig auf der lokalen Kopie gearbeitet wird.

9 Audio-/Video-Codecs

Zur Realisierung einer entsprechend performanten Umgebung für Audio/Video-Konferenzen und Streaming-Angebote ist es essentiel notwendig, eine Abschätzung des Rechenaufwandes für die unterschiedlichen Codecs durchzuführen. Das folgende Kapitel beschäftigt sich ausschließlich mit Messungen zur CPU- und Arbeitsspeicherbelastung des N810 unter Nutzung verschiedener Codecs wie beispielsweise DivX, MP3 und MPEG mit jeweils unterschiedlichen Bitraten. Als Testbasis wurden Film- und Tonausschnitte verwendet, um eine möglichst realitätsnahe Situation zu konstruieren.

9.1 Skalierung von Mediaplayern bei verschiedenen Codecs und Bitraten (Björn, Stephan, Nils)

9.1.1 Einleitung

Da wir mit dem N810 über ein eingebettetes System verfügen, bei dem die Hardwareressourcen alles andere als unbegrenzt sind, ist es erforderlich, von vorne herein zu prüfen, mit welchen Multimediaformaten die Hardware umgehen kann und wieviel Last diese Formate bei verschiedenen Einstellungen der Bitrate erzeugen. Denn die Auslastung des Systems beim Dekodieren von Audio- und Videostreams hängt sowohl von den gewählten Codecs als auch von der gewählten Bitrate ab. In diesem Kapitel soll nun darauf eingegangen werden, wie die Multimediaprogramme *GStreamer* und *MPlayer* das N810 bei der Wiedergabe von Audio- und Videodateien belasten. Ziel der Tests war es, signifikante Verhaltensänderungen festzustellen, um dadurch Grundlagen für die Auswahl der zu unterstützenden Videoformate und deren Parameter zu finden.

Es wurden dabei insgesamt drei Testreihen durchgeführt: Die erste mit einem Ausschnitt der Serie „The IT Crowd“, wo nur wenige schnelle Bewegungen vorkommen, die zweite mit einer Szene aus dem Kinofilm „Der Transporter“, welche von schnellen Bewegungen dominiert wird, und die dritte Testreihe mit einem Musikstück.

Dabei ist zu beachten, dass vor der Durchführung von Tests die Stromsparmechanismen des Systems deaktiviert werden mussten, so dass sich die CPU nicht automatisch heruntertaktete, wodurch Daten bezüglich der CPU-Last nicht nutzbar würden. Die Last haben wir durch Beobachten des Programmes „top“ über den Verlauf der Testdateien ermittelt. Dabei lagen immer recht große Schwankungen in der CPU-Auslastung vor, so dass es nicht möglich war, einen genauen Wert festzuhalten. Deshalb haben wir immer versucht eine obere und untere Schranke für die vorliegende Last zu bestimmen, wobei auch hier zu beachten ist, dass das Programm „top“ nur ungefähr einmal pro Sekunde die Ausgabe aktualisiert und entsprechend nur diskrete Werte vorlagen. Entsprechend sind die angegebenen Prozentwerte für die Bereiche, in denen die Last lag, auch nicht auf extrem hohe Genauigkeit ausgelegt, sondern als grobe obere und untere Schranke zu sehen, die auf einige Prozentpunkte genau ist.

9.1.2 Erste Testreihe: Serie mit wenigen Bewegungen

Die getestete Szene hatte eine Auflösung von 496x252 Pixeln und lief mit 25fps. Die Tonspur war in MPEG-1 Layer-3, Stereo mit 48kHz und variabler Bitrate codiert. Die Bildspur haben wir als DivX-5, XviD (beides sind mpeg4-asp basierte Formate) und als h.264 kodiert. Die Dateien waren mit dem integrierten GStreamer-basierten Mediaplayer nicht abspielbar. Einzig die Tonspur konnte man wiedergeben, nachdem wir das Video in Audio und Videoteil aufgesplittet hatten. In dieser Testreihe haben wir auch untersucht, wie die verschiedenen vom MPlayer unterstützen Backends für die Ausgabe von Audio und Video die Last beeinflussen.

Bei der Wiedergabe mithilfe des MPlayers stellte sich heraus, dass das mit h.264 encodierte Video nicht flüssig abspielbar war, die Belastung des Systems war zu hoch. Die Wiedergabe als XviD und DivX-5 encodiertes Video klappte dagegen problemlos. Dabei war die CPU-Last bei der XviD kodierten Version mit 60% bis 70% durchgehend höher als bei der DivX-5 Version, wo die Last im Bereich von 40% bis 55% lag. Bei der Wiedergabe des Videos mit der Option -nosound lag die Last durchgehend um ca. 10 Prozentpunkte niedriger als zuvor. Der mit h.264 encodierte Videostrom war dennoch nicht flüssig abspielbar.

Im nächsten Schritt haben wir überprüft, wie sich verschiedene Videotreiber beim MPlayer und der DivX-5 encodierten Datei in Bezug auf die CPU-Last verhalten. Der Test mit dem Treiber 'xover' ist komplett gescheitet. Beim Test mit 'x11' wurde das Video nicht flüssig wiedergegeben. Die Last bei Nutzung von 'xv' lag bei 50% bis 60%, bei Nutzung von 'sdl' bei 55% bis 65% und bei Nutzung von 'omapfb' bei 40% bis 55%, dem Wert, den wir auch schon im vorherigen Durchlauf bemerkt haben. Entsprechend wird für die Videoausgabe standardmäßig wohl der 'omapfb' Treiber genutzt.

Nach dem Absplitten der Tonspur war es möglich, diese im integrierten Mediaplayer abzuspielen. Sie verursachte $<5\%$ Last, was nahelegt, dass dort der DSP genutzt wurde. Beim Test dieser Datei mit MPlayer zeigt sich, dass anscheinend nur bei Nutzung von 'sdl' als Audiotreiber der DSP verwendet wird, denn hier lag die Last bei $<5\%$. Bei Nutzung von 'alsa' lag die Last bei ungefähr 10%, bei Nutzung von 'esd' lag sie bei 15% bis 20%.

Ergebnis dieser Testreihe ist, dass XviD mehr Last verursacht als DivX-5 und dass die richtige Wahl des Ausgabetreibers signifikante Auswirkungen auf die CPU-Last bei der Wiedergabe hat.

9.1.3 Zweite Testreihe: Film mit schnellen Bewegungen

Für diesen Test haben wir eine Szene mit vielen schnellen Bewegungen genommen und uns, aufbauend auf den Resultaten der ersten Testreihe, dazu entschieden DivX-5 und XviD als Encoder zu nutzen. Dabei haben wir mit den Bitraten 1000kbit, 750kbit,

500kbit, 250kbit sowie einer qualitätsbasierten Einstellung mit 4 und 8 Quantizer gearbeitet, welche durchgehend eine ähnliche Qualität gewährleisten sollte. Des Weiteren haben wir bei der Tonspur MPEG1 Layer-3 mit einer konstanten Bitrate von 128kbit, mit variabler Bitrate und unkomprimierten PCM-Datenstrom als Formate für die Messreihen genutzt. Wir haben sämtliche Permutationen von Codec, Videobitrate und Tonspur im MPlayer auf Prozessorauslastung und subjektive Einschätzung hin getestet. Die Szene lag dabei in den Auflösungen 800x480 und 400x240 vor. Die eine ist bekanntlich die volle Auflösung des N810, die andere so gewählt, dass problemlos durch Verdopplung der Pixel hochskaliert werden kann.

In der Testreihe hat sich gezeigt, dass der Einfluss der Videobitrate auf die CPU Auslastung recht gering ist. Er bewegt sich sowohl bei XviD als auch bei DivX-5 im Bereich von nur 5 Prozentpunkten. Allerdings hat die Wahl des Audiocodecs durchaus einen Einfluss. Bei Nutzung eines unkompromierten PCM Datenstroms lag die CPU-Last ca. 10 Prozentpunkte niedriger, als bei Nutzung von MP3, egal ob mit konstanter oder variabler Bitrate. Eine Schlussfolgerung dieser Testreihe ist es, dass beim Abspielen von Videodateien der Ton anscheinend nicht über den DSP decodiert wird, weshalb diese Arbeit der ARM11 CPU zufällt, wodurch die beobachtete Last ansteigt.

Durch diese Resultate ermutigt, haben wir noch verglichen, wie sich die CPU-Last ändert, wenn man die Audioausgabe deaktiviert. Dabei kam heraus, dass der Mehraufwand für Audio, wenn es sich um einen unkomprimierten Datenstrom handelt, bei bis zu 4 Prozentpunkten CPU-Last lag.

Letztlich wollten wir noch testen, ob der integrierte GStreamer basierte Player bei diesen Testdateien andere Ergebnisse liefert. Er war in der Lage, die DivX-5 kodierten Dateien abzuspielen, bei den XviD kodierten Versionen kam es zu massiven Bildfehlern, so dass sie nicht nutzbar waren. Allerdings war die CPU Last bei den DivX-5 basierten Videos signifikant höher als bei Nutzung des MPlayer. Die Last lag um 5 bis 15 Prozentpunkte höher.

Schlussfolgerung dieser Testreihe ist, dass die Auslastung der Hardware beim Abspielen von Videos kaum von der genutzten Bitrate abhängt. Dabei hat sich gezeigt, dass die Formatunterstützung des integrierten Players wieder für Probleme sorgt und er eine höhere Last verursacht.

9.1.4 Dritte Testreihe: Musikstück

In der letzten Testreihe haben wir betrachtet, wie sich die CPU Auslastung beim Abspielen von Audiodateien verhält. Da wir wissen, dass bei Nutzung des DSP die Last meist in sehr niedrigen Bereichen liegt, haben wir die Wiedergabe über die CPU erzwungen. Hierfür haben wir explizit das Audioausgabegerät 'alsa' im MPlayer gewählt. Dies ist insofern praxisrelevant, als dass beim Abspielen von Videos anscheinend der DSP nicht für die Audioausgabe verwendet wird. Dies legen zumindest die Ergebnisse der ersten

Testreihe nahe. Entsprechend sind diese Ergebnisse insbesondere sinnvoll, um sie im Zusammenhang mit der Videowiedergabe und der verwendeten Tonspur zu betrachten. Da man die Verwendung des DSP im integrierten Player anscheinend nicht ohne weiteres deaktivieren kann, wurde diese Testreihe einzig mit dem MPlayer erstellt.

Für den Test haben wir ein Musikstück auf folgende Arten kodiert:

- MPEG1 Layer-3, konstante Bitrate von 64, 128, 192 und 320kbit
- MPEG1 Layer-3, variable Bitrate von im Durchschnitt 64, 128 und 192kbit
- Vorbis mit den Qualitätsstufen q0, q4 und q6
- Unkomprimierter PCM-Datenstrom

Die Unterschiede in der Auslastung bei Nutzung der verschiedenen MP3-kodierten Dateien waren dabei im Bereich der Messungenauigkeit. Die Last betrug 8% bis 12%. Die Nutzung von Vorbis ergab durchgehend eine ein bis zwei Prozentpunkte höhere Last als bei Nutzung von MP3. Beim unkomprimierten Datenstrom lag die CPU-Auslastung zwischen zwei und vier Prozent.

Auch in diesem Test zeigt sich wieder, dass die Wahl des Codecs den größten Unterschied ausmacht. Die Wahl der Bitrate dagegen hat nur einen sehr geringen, teilweise kaum messbaren Einfluss auf die CPU-Auslastung.

9.1.5 Fazit

Ergebnis unserer Tests bezüglich Skalierung mit verschiedenen Formaten und Bitraten ist, dass die Wahl des Codecs eine große Rolle spielt. Die Wahl der Bitrate hingegen, zumindest bei den hier untersuchten, vernünftigen Werten, verändern die Messung kaum. Entsprechend sollte man bei der Wahl der Bitrate vor allem die auf das Netzwerk ausgeübte Last im Auge behalten werden, da die vorhandene Bandbreite doch beschränkt ist. Auch die Auflösung sollte man beachten, da die CPU-Last mit der Auflösung entsprechend auch steigt. Bei einer Auflösung von 400x240 sollten auf dem N810, auch wenn UbiMuC noch im Hintergrund läuft, wohl kein größeres Problem auftreten.

Nach dieser Testreihe waren wir enttäuscht von der Kompatibilität des im N810 integrierten Mediaplayers. Bei vielen Dateien war er entweder gar nicht in der Lage, die Datei abzuspielen und beendete die Versuche mit der Meldung „unbekanntes Dateiformat“, oder er stellte das Bild mit starken Störungen, wie in der zweiten Testreihe beobachtet, dar. Im aktuellen Zustand ist der integrierte Player kaum für unsere Zwecke brauchbar. Hier ist zumindest Arbeit nötig, um die Kompatibilität entsprechend zu erhöhen, so dass die Wiedergabe weniger Probleme bereitet.

9.2 Betrachtung zu unterstützender Multimedia Codecs (Björn, Stephan, Nils)

9.2.1 Einleitung

Im vorherigen Kapitel wurde gezeigt, dass das N810 auch mit leistungsstarken Video- und Audiocodes umgehen kann, die selbst für Kinofilme mit sehr schnellen Bewegungsabläufen geeignet sind. Dennoch eignen sich die dort verwendeten Formate nicht für alle Einsatzgebiete. *MPEG*-kodierte Inhalte sind zwar beispielsweise verhältnismäßig platzsparend, was es zum bevorzugten Format zum Speichern von Video- und Audioinhalten macht, aber der Ressourcenverbrauch, der zum Erzeugen von *MPEG*-Dateien benötigt wird, ist enorm hoch und insbesondere bei der Video-Kodierung vom N810 nicht in Echtzeit zu bewältigen. Doch gerade die Echtzeitfähigkeit ist unverzichtbar bei der Videotelefonie.

Also ist es sinnvoll, sich nach speziell für dieses Einsatzgebiet entworfenen Formaten umzuschauen. Gerade im Bereich der Sprach- und Videotelefonie gibt es Codecs, die sowohl beim Kodieren als auch Dekodieren wenig Rechenleistung verbrauchen und dennoch einen gut komprimierten Datenstrom produzieren.

9.2.2 Geeignete Formate für Sprachübertragung

- **G.711**
  G.711 ist der Codec, der auch bei der Sprachübertragung in *ISDN*-Netzen verwendet wird. Er bietet daher eine relativ gute Sprachqualität und kommt mit 64 kBit/s Datenrate aus. Bemerkenswert ist außerdem, dass das Format bereits in den 60er Jahren vorgestellt wurde. Daher ist der Ressourcenbedarf auf beiden Seiten (Kodierer und Dekodierer) sehr gering.
- **GSM - Global System for Mobile Communications**
  Die Klasse der *GSM*-Audiocodecs umfasst mehrere Formate zur Audiokodierung und wird, wie der Name vermuten lässt, in Mobilfunknetzen eingesetzt. Die Bitrate des Datenstroms geht von 4,75 kBit/s bis 13,2 kBit/s, und ist damit noch deutlich geringer als bei *G.711*. Da im Mobilfunkbereich häufig mit Datenfehlern zu rechnen ist, wurde bei den *GSM*-Formaten viel Wert auf Datenkorrektur gelegt, d. h. auch bei Übertragungsfehlern sollte Sprache weiterhin verständlich bleiben, was dieses Format trotz des höheren Kodierungsaufwands interessant für die Sprachtelefonie auf dem N810 macht.

Sowohl *G.711* als auch *GSM* werden von der *libsndfile*-Bibliothek unterstützt, die unter der LGPL steht.

9.2.3 Geeignete Formate für die Videoübertragung

Für die Videoübertragung mit niedriger Bandbreite eignen sich *H.261* und dessen Nachfolger *H.263*.

H.261 wurde 1990 standardisiert und diente dazu, Videobilder über einen ISDN B-Kanal mit 64 kBit/s zu übertragen. Die Auflösung beträgt dabei 176x144 Pixel (*QCIF*). Das Format wurde einst für Bildtelefone verwendet. Der Nachfolger *H.263* ergänzt diesen Codec mit weiteren Bitraten und Auflösungen.

Beide Formate werden von der *libavcodec*-Bibliothek, die ebenfalls unter der LGPL steht, unterstützt.

10 Die Entwicklung der Benutzerschnittstelle

Dieses Kapitel befasst sich mit den Vorüberlegungen, dem Design und der letztendlichen Programmierung der GUI unseres Programms.

Für ein heutiges Programm ist eine GUI unentbehrlich. Sie erlaubt die Nutzung selbst für einen weniger versierten Personenkreis, da sie die Bedienung des Programms auf die für den Nutzer wichtigen Funktionen beschränkt. Dementsprechend mussten Überlegungen angestellt werden, welche Funktionen dem Nutzer zur Verfügung stehen, und wie diese innerhalb der GUI platziert werden sollen, damit die Bedienung leicht verständlich aber gleichzeitig komfortabel ist.

Dies wird zusätzlich durch die vorhandene Hardware - besonders durch die Bildschirmgröße, die Eingabegeräte und die beschränkten Ressourcen - erschwert.

10.1 Einführung in GTK (Stephan, Fabian, Markus G.)

GTK+[34] ist ein Akronym und steht für „GIMP Toolkit“. Es ist eine Bibliothek zur Programmierung graphischer Benutzeroberflächen. GTK basiert auf der imperativen Programmiersprache C und ist somit nicht objektorientiert. Die GTK zugrunde liegende Bibliothek glib realisiert allerdings ein objektorientiertes System. Objekte werden dabei in Form von Zeigern dargestellt und mit Hilfe von Funktionen modifiziert.

Sämtliche sichtbaren Objekte, wie z.B. Buttons oder Textfelder - aber auch nicht sichtbare Objekte wie z.B. Gruppierungen oder Ausrichtungen von Text - werden dabei als „Widgets“ bezeichnet. Diese Widgets werden hierarchisch ineinander verschachtelt und bilden zusammen die graphische Oberfläche eines Programms.

Da GTK ein fester Bestandteil des N810 ist und dessen Window-Manager ebenfalls auf dem Toolkit basiert, ist es möglich, ein Nutzerinterface zu realisieren, ohne zusätzlichen Overhead für das Einbinden weiterer Bibliotheken zu erzeugen. Des Weiteren nutzen wir das Hildon Toolkit, welches eine Erweiterung von GTK darstellt und eine auf das N810 zugeschnittene Bibliothek ist, die das Nokia-typische „look and feel“ realisiert. Da das Hildon Toolkit [35] speziell für eingebettete Systeme entwickelt wurde, ist es gegenüber dem üblichen GTK schlanker und ressourcenfreundlicher.

Auf diesen Grundlagen realisieren wir die Benutzeroberfläche unseres Programms.

10.2 Die grundlegende GUI von UbiMuC (Stephan, Fabian, Markus G.)

10.2.1 Theoretische Vorüberlegungen

Bei eingebetteten Systemen sind nicht nur Rechenressourcen, sondern auch mögliche optische Ausgabemöglichkeiten sehr eingeschränkt. Hierbei stellt zum Beispiel die Bildschirmauflösung [36] des Gerätes eine beschränkende Grenze dar: Schrift und Bedienelemente dürfen weder zu klein, noch zu groß gewählt werden. Auch stellen ungewöhnliche Eingabegeräte wie Stift und Touchscreen eine Herausforderung dar. Aus diesen Gründen sind theoretische Vorüberlegungen (siehe Bild 25) zur Gestaltung der Benutzeroberfläche eines eingebetteten Systems unerlässlich. Im Folgenden legen wir die theoretischen Überlegungen zur Gestaltung der graphischen Oberfläche unseres Projektes dar:

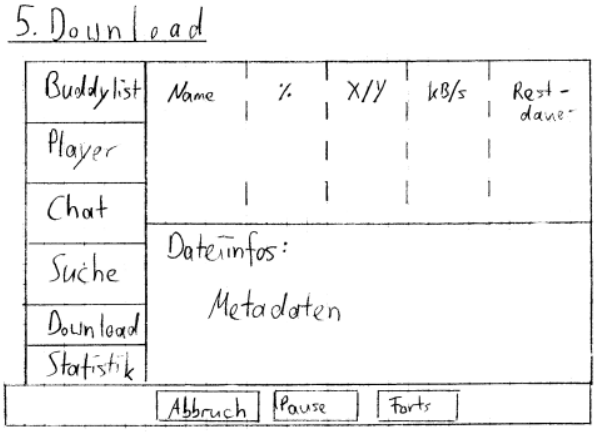

Abbildung 25: Erste Originalskizze der UbiMuC-GUI

Die Tabs am linken Bildschirmrand sorgen für einen schnellen Zugriff auf die wichtigsten Funktionen des Programms. Die Toolbar am unteren Bildschirmrand passt sich dem ausgewählten Tab an und stellt die benötigten Ein- und Ausgabe-Funktionen zur Verfügung. Die übrige Fläche des Bildschirmes wird zur Darstellung des eigentlichen Inhalts des ausgewählten Tabs benutzt. Weitere, seltener benutzte Funktionen sind über ein Menü in der Titelzeile verfügbar. Durch diese Aufteilung wird das „look-and-feel“ der übrigen Programme des N810 imitiert.

Eine weitere wesentliche Überlegung betraf die Implementierung der Funktionalität. Nach reiflicher Überlegung und Diskussionen im Plenum wurde beschlossen, die Funktionalität der GUI nicht direkt in ihren Quellcode zu integrieren, sondern die Funktionalität von der graphischen Oberfläche zu trennen. Es wurde ein Arbeitskreis zur Festlegung von Schnittstellen (siehe Kapitel 10.3) zwischen den beiden Programmteilen gebildet. Diese Entscheidung ermöglicht später ein problemloses Wechseln der GUI bei Portierung auf andere Systeme.

10.2.2 Praktische Implementierung

Generelle Probleme

Nach den theoretischen Überlegungen begann die Umsetzung dieser Ideen. Hierbei traten verschiedenste Probleme auf:

Wie oben bereits erwähnt, ist GTK eine C basierte Bibliothek. Dies bedeutete insbesondere die Einarbeitung der PG-Teilnehmer in das imperative Programmier-Paradigma sowie das Erlernen der Eigenheiten der Programmiersprache C. Dazu gehörten unter anderem der Umgang mit Zeigern und die selbst zu implementierende Speicherverwaltung. Aber nicht nur diese Eigenart, welche die Verwendung von Zeigern erfordert, sondern auch der schwer überschaubare Funktionsumfang der Bibliothek GTK, erforderten viel Aufwand bei der Einarbeitung. So war oft nicht klar, ob eigene Ideen selbst implementiert werden mussten oder es Vorlagen in Form von fertigen Widgets für diese Ideen gab.

Weitere Probleme stellten die unterschiedlichen - subjektiven - Ansichten zur Strukturierung des Projektcodes dar. Hierzu musste unter anderem geklärt werden, ob wiederverwendbare Codefragmente ausgelagert oder kopiert und in andere Dateien eingefügt werden sollten. Ebenso war die Namensgebung der Funktionen anfangs nicht kohärent, was gemeinsames aber auch verteiltes Arbeiten erschwerte. Aufgrund dieser Probleme wurden Coding Conventions eingeführt.

Implementierungsdetails

Zunächst wurde eine rein GTK basierte GUI erstellt. Diese wurde im weiteren Verlauf durch spezifische Hildon Elemente ergänzt und fortlaufend auf das N810 optimiert. Jedoch wurde die komplett GTK-basierende GUI nicht verworfen, um diese auf Geräten mit fehlender Hildon-Integration verwenden zu können. So stehen der Projektgruppe mehr als nur die gestellten sechs N810 zum Testen des Projektes zur Verfügung.

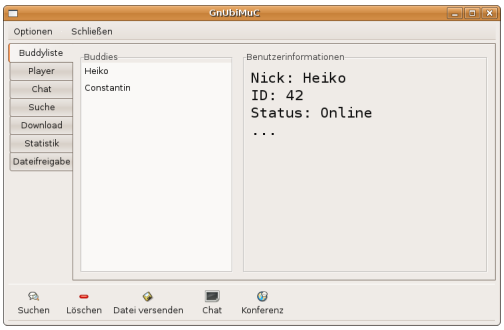

Abbildung 26: GTK-Version der UbiMuC-GUI

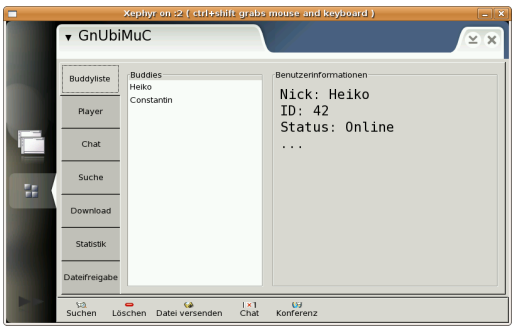

Abbildung 27: Hildon-Version der UbiMuC-GUI

Um den Funktionsumfang - insbesondere bei der Verarbeitung von Streams - bewältigen zu können, fand eine Konvertierung von C nach C++ und ein damit verbundener

Paradigmenwechsel statt. Dies bedeutete wiederum Einarbeitung in eine neue Programmiersprache, mit ihren Eigenheiten und Möglichkeiten.

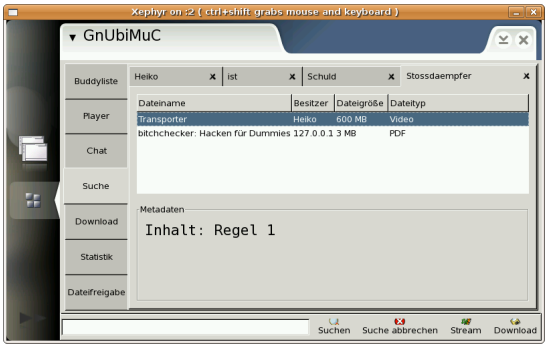

Abbildung 28: Tab-Version des Suche-Fensters der UbiMuC-GUI

Jedoch ergaben sich bei der Implementierung auch einige Probleme. So waren das Schließen von Tabs, das Einfügen von Textfeldern in die Toolbar oder auch die Integration von Icons jeweils eine Herausforderung.

Auch das Einbetten des „MPlayers“ in unser Programm erwies sich als nicht-triviale Aufgabe, mit der sich folglich eine eigene Arbeitsgruppe beschäftigt. Weitere Informationen hierzu finden sich in Kapitel 10.4.

10.2.3 Fazit

Obwohl ein erster, funktionierender Skelett-Entwurf steht, befindet sich die GUI immer noch in der Entwicklung. Diese wird auch weiterhin noch Zeit in Anspruch nehmen, zumal die Oberfläche iterativ nach Bedarf ergänzt wird.

10.3 Die Schnittstelle zu GNUnet (Jens, Thomas, Michael P., Michael K.)

Um die zentralen Funktionen von GNUnet zu benutzen, ist bislang geplant, eine umgebende C-Klasse zu schreiben, welche als Adapter zwischen der UbiMuC-GUI und dem GNUnet-Kern fungiert. Dort sollen alle Anfragen und Kommandos des Nutzers über die GUI interpretiert und weitergeleitet werden. Die Antworten des GNUnet-Kerns werden wiederum von der Adapter-Klasse umgearbeitet und an die GUI übergeben. Der GNUnet-Kern kann dabei einfach per Konsolenbefehl gestartet werden und ist im Anschluss ansprechbar.

Konkret sollen dabei die Szenarien der Anwendungsfälle realisiert werden, also Suche von Inhalten, Veröffentlichung von Inhalten, Download/Upload von Dateien und das

Streaming. Bei den ersteren Fällen ist im Wesentlichen nur eine Umformatierung und Weiterreichung der Anfragen bzw. Antworten nötig, damit GUI und GNUnet miteinander auf komfortable Weise über den Adapter interagieren können, ohne dass ausschweifende Änderungen an den eigentlichen GNUnet-Sourcecodes notwendig sind.

Die Realisierung einer Streaming-Funktion gestaltet sich allerdings etwas schwieriger, da GNUnet in der aktuellen Version keine bidirektionalen Datenverbindungen bereitstellen kann. Dies liegt dabei in erster Linie an dem bereits erwähnten GAP-Protokoll, sowie dem Pull-Paradigma des Query-Response-Designs. Ein Peer kann daher über GNUnet nicht „einfach Daten in das Netzwerk“ einschleusen. Dies ist nur in Form einer Reaktion auf eine Query möglich. Es ist also unumgänglich, ein eigenes Nachrichtenformat mit eigener Behandlung in GNUnet zu integrieren, damit eine Funktion wie das Streaming oder auch Videokonferenzen aufgebaut werden können. Bei Gesprächen mit einem der hauptverantwortlichen Entwickler stellte sich heraus, dass GNUnet auf praktische Art und Weise um neue Nachrichtentypen und Behandlungen erweitert werden kann. Daher ist geplant, ein eigenes Format zu entwickeln, sowie eine eigene Behandlung des Routings für derartige Nachrichten, um Streaming und Videokonferenzen zu ermöglichen.

10.4 Die Integration des MPlayer (Björn, Lutz)

10.4.1 Einleitung

Für eine Multimedia-Anwendung ist die Fähigkeit, Audio- und Videodaten abzuspielen zu können, obligatorisch. Allerdings ist das Implementieren einer eigenen Abspielsoftware nicht trivial, da hierbei zu viele verschiedene Aspekte zu beachten sind. Das Verwenden einer bereits existierenden Software ist normalerweise auch problematisch, da sich externe Applikationen oft nur schlecht in andere Anwendungen integrieren lassen.

MPlayer ist zwar auch ein eigenständiges Programm, allerdings kann man es in einem *Slave*-Modus betreiben, der verhindert, dass das Programm ein eigenes Benutzerinterface verwendet. Somit lässt es sich vollständig fernsteuern, was auch zwingend notwendig ist, um es in unserer Anwendung einbetten zu können. Des Weiteren kann man *MPlayer* anweisen, eine evtl. vorhandene Videoausgabe in ein bestehendes Fenster zu zeichnen. So hat man weiterhin die vollständige Kontrolle über das Videofenster, muss sich aber nicht selbst um die Videoausgabe kümmern.

10.4.2 Datenübertragung

MPlayer bleibt, trotz des *Slave*-Modus, ein eigenständiges Programm, was das Weiterleiten von Audio- und Videodaten erschwert. Um dennoch den Datenaustausch zwischen den Prozessen zu ermöglichen, werden *File-Deskriptoren* eingesetzt, die *MPlayer* wie Dateien verwenden kann, deren Inhalte allerdings in Echtzeit von einem anderen Programm

generiert werden. Dazu werden zunächst bidirektionale *Pipes* erzeugt, die auf einem *POSIX*-System dieselbe Schnittstelle besitzen wie Dateien. Statt nun eine reale Datei zu übergeben, wird die dem *File-Deskriptor* entsprechende virtuelle *Gerätedatei* übergeben. Der *MPlayer* nimmt nun Daten aus der *Pipe* entgegen, die auf der anderen Seite von der Hauptanwendung nach belieben generiert werden können.

Dieses Prinzip ermöglicht es unserer Anwendung, Multimediadaten unabhängig vom Herkunftsort mit dem *MPlayer* verarbeiten zu können.

10.4.3 Beispiel

Das Hauptprogramm muss zuerst ein leeres GUI-Element erzeugen. Diese noch leere Fläche hat, wie jedes andere GUI-Widget auch, eine eindeutige Identifikationsnummer. Diese Nummer kann der *MPlayer* verwenden, um die Videoausgabe auf genau dieses GUI-Element umzuleiten. So erfolgt die Videoausgabe in der GUI unserer Anwendung, obwohl die Dekodierung in einem externen Prozess statt findet.

Die Befehlszeile, die das Hauptprogramm dazu ausführen muss, um den *MPlayer* in diesem Modus zu starten, könnte so aussehen:

```
mplayer -slave -wid 1234 /dev/fd/42
```

wobei *1234* die ID des Fensters ist, wo die Videoausgabe erscheinen soll, und */dev/fd/42* die Gerätedatei des vom Hauptprogramm erzeugten File-Deskriptors. Über der Standardein- und ausgabe des neuen Prozesses kann man den *MPlayer* steuern.

11 Zusammenfassung und Ausblick

Die bisher erzielten Ergebnisse bieten uns eine gute Ausgangslage für die weiteren Entwicklungen. Auf der einen Seite steht bereits eine grafische Benutzeroberfläche, die die derzeit geplanten Programmteile enthält. Auf der anderen Seite steht das Basisgerüst für die Datenübertragung, gegeben durch die Nutzung des GNUnet-Frameworks. Zentrales Element des zweiten Projektgruppensemesters ist die Entwicklung einer Interaktionsschicht, der WIR-Schicht. Diese soll die Interaktion von GUI, GNUnet, weiteren eingebundenen Modulen, wie dem MPlayer, sowie dem Datei- bzw. Betriebssystem steuern.

11.1 Zusammenfassung der bisherigen Ergebnisse der Projektgruppe (Markus S)

Dieser Abschnitt soll noch einmal, nach den verschiedenen Themengebieten sortiert, zusammenfassen, was innerhalb des 1. Semesters in der PG an Ergebnissen erreicht wurden.

Hardware

Wir haben uns mit der durch das N810 gegebenen Hardware und Software auseinandergesetzt und darauf basierend einige softwarerelevante Entscheidungen treffen können, wie unser eigenes Programm gestaltet werden soll, welche Anwendungsfälle unterstützt werden sollen und welche Technologien wir ausschließen können. Auf Basis von Analysen der Performance weiter zu verwendender Teilkomponenten, wie Dateiformaten, Multimedia-Applikationen, P2P-Framework-Komponenten u.ä. konnten wir die Grenzen, die uns die Hardware bot, leichter einschätzen und die zu unterstützenden Funktionalitäten leichter voneinander abgrenzen.

Netzwerk-Topologie

Zum Aufbau eines P2P-Netzwerks haben wir uns mit den unterschiedlichen konzeptionellen Möglichkeiten auseinandergesetzt, wie unser P2P-Netzwerk strukturiert werden kann. Es musste eine Entscheidung getroffen werden, ob unser Netzwerk ein reines, serverbasiertes oder ein hybrides P2P-Netz darstellen soll. In diesem Zusammenhang wurden die Vor- und Nachteile aller Lösungen im Kontext unserer Aufgabenstellung und unserer Anwendungsfälle betrachtet und insbesondere das Problem des Findens des ersten Peers in einem solchen Netzwerk diskutiert. Letztlich wurde auf Grund der Größe unserer angestrebten Netze eine Entscheidung zugunsten eines serverbasierten Ansatzes getroffen. Die Möglichkeit, das Problem des ersten Peers durch unsere Software-Lösungen mit Hilfe von Avahi und ZeroConf zu umgehen, trug ebenfalls dazu bei. Weiterhin wurden Überlegungen angestellt, wie sich Probleme wie das Auflösen, Neubilden und Vereinigen von

Netzwerken innerhalb eines solchen P2P-Netzwerks konzeptionell verwalten lassen. Da unsere Software für mobile Endgeräte gedacht ist, stellt die Lösung dieses Problems eine wichtige zu implementierende Funktionalität dar. Es wurden verschiedene Szenarien konstruiert und Vorgehensweisen betrachtet, um mit diesen umzugehen.

Unterstützung von Ad-hoc-Netzen

Für das spontane Aufbauen von Netzverbindungen zwischen unseren mobilen Endgeräten wurde nach der Seminarphase Zeroconf als interessante Option festgehalten. Im Folgenden prüften wir, ob für unsere Hardware/Software passende Implementierungen verfügbar sind und testeten schrittweise die Nutzbarkeit von Zeroconf in Verbindung mit Avahi im Zusammenhang mit den N810-Netzen. Letztlich wurde es in unser P2P-Framework integriert.

Auswahl des P2P-Frameworks

Bereits zu Beginn der PG entschieden wir uns aus Zeitgründen dafür, bevorzugt ein bestehendes P2P-Framework anzupassen anstatt selbst eines zu programmieren. Das in der Seminarphase vorgestellte JXTA-Framework stellte sich in seiner C-Implementierung allerdings als sehr problematisch heraus, sodass weitere Frameworks in Betracht gezogen wurden. Dabei kristallisierte sich GNUnet als die beste Alternative zu JXTA heraus und wurde letztlich in einem Direktvergleich zwischen JXTA und GNUnet als der bessere Kompromiss und als Basis für unsere weitere Arbeit gewählt. Folglich mussten im Weiteren eine große Zahl von Anpassungen an GNUnet und an unseren anderen Software-Komponenten vorgenommen werden, da manche Anforderungen an unsere Software nicht von GNUnet direkt unterstützt wurden und manche Features von GNUnet zu Performance-Problemen führten oder unerwünschte Nebeneffekte mit sich brachten.

Arbeiten an GNUnet

Nachdem GNUnet als das von uns zu nutzende Framework festgelegt wurde, mussten wir erst sehr viel Zeit in die Analyse des Quellcodes investieren, um die genaue Funktionsweise des Frameworks zu verstehen. Dabei standen uns die Entwickler dieses Projekts mit vielen Antworten zur Seite. Außerdem wurde eine für das N810 nutzbare Version von GNUnet erstellt und als Pakete in unserem Repository abgelegt, um das weitere Arbeiten an GNUnet zu erleichtern. Im Weiteren werden in Kurzform all die speziellen Themenbereiche beschrieben, mit denen wir uns im Zusammenhang der Integration von GNUnet in unsere übrige Software-Struktur und im Rahmen der Analyse des GNUnet-Funktionsumfangs auseinandergesetzt haben:

- Einbindung von Avahi in GNUnet:

  Da GNUnet auf Basis von ad hoc aufgebauten Netzwerken arbeiten soll, wurde Avahi als unsere Lösung für den Aufbau dieser Netzwerke in GNUnet sehr frühzeitig integriert. Dieser Teil ist bereits funktionsfähig abgeschlossen.

- Routing-Funktionalitäten:

  Ein Verständnis für das von GNUnet bereitgestellte Routing zu gewinnnen war einer der zentralen Punkte unserer Arbeit, da nur dadurch eine effizient arbeitende, darauf aufsetzende, eigene Software entstehen kann. Die bereits integrierte Verschlüsselung stellte sich als sehr vorteilhaft heraus. Jedoch produzierten verschiedenste Anonymisierungs-Funktionen eine Menge unerwünschten Overheads. Daher beschäftigten wir uns damit, wie wir diese Funktionen abschalten oder umgehen können und welche Anpassungen deshalb am dazugehörigen Protokoll nötig werden.

- Austausch von Hostlisten:

  Unsere Software soll schließlich Anwendungsfälle zum Streaming und Chat innerhalb des Netzwerks unterstützen. Daher ist eine permanente Speicherung der aktuell verfügbaren Nutzer innerhalb des Netzes notwendig. Deshalb wurden die Erstellung, die Änderung und der Austausch der dafür genutzten Hostlisten genauer untersucht.

- Libextractor:

  Um die Meta-Daten von Multimedia-Dateien auslesen und innerhalb eines P2P-Programms nutzen zu können, werden spezielle Funktionalitäten benötigt. Diese werden in GNUnet von Libextractor bereitgestellt. Wir haben uns damit beschäftigt, mit welchem Funktionsumfang wir diese Bibliothek nutzen wollen und in wie weit sich dies mit unseren beschränkten Hardware-Ressourcen vereinbaren lässt. Da die Realisierung dieser Features nicht zwingend notwendig ist und die Nachteile der Integration noch nicht vollständig absehbar sind, wurde das Thema zurückgestellt und wird zu einem späteren Zeitpunkt erneut evaluiert.

- Personensuche über DHT:

  Um in einem P2P-Netz Dateien oder Streams abrufen zu können, muss zunächst klargestellt werden, wer zum aktuellen Zeitpunkt Netzteilnehmer ist. Da wir nicht immer von einem server-gestützten Netz ausgehen können, mussten wir eine Lösung finden, diese Informationen dezentral auf den Endgeräten zu verwalten. Die Tatsache, das innerhalb des GNUnet-Frameworks bereits „Distributed Hashtables“ implementiert sind, machten wir uns zu Nutze und beschlossen, diese um die für uns nötigen Funktionalitäten zu erweitern. Problemstellungen waren vor allem, wie eine persistente Nutzererkennung gewährleistet werden kann und welche Anwendungsszenarien in diesem Kontext auftreten können. Neben der Erstellung der Konzepte wurden Tests zum Umgang mit der DHT durchgeführt. Die Umsetzung der Ergebnisse ist jedoch erst in später entstehenden Programmteilen möglich.

Streaming und Codecs

Durch die starken Einschränkungen, die uns die Hardware bei der Verarbeitung von Multimedia-Inhalten auf dem N810 auferlegt, mussten wir sehr kritisch betrachten, welche Formate und Codecs wir in unserem fertigen Programm unterstützen wollen. Wir führten Testreihen durch, um die Performance der verschiedenen De- und Encoder auf dem N810 zu prüfen und wir entschieden, aufgrund der relevanten Anwendungsfälle und Kompatiblitätsproblemen einiger Formate beim integrierten Mediaplayer, uns auf eine kleine Anzahl von Formaten/Codecs zu beschränken.

Anwendungsfälle

Unsere selbst erstellte Software soll sehr verschiedene Aufgabenstellungen im Kontext von Streaming erfüllen. Daher beschäftigten wir uns recht frühzeitig damit, welche Anwendungsfälle letztlich unterstützt werden sollen. Im Kern handelt es sich dabei um vier Hauptfunktionalitäten: Audio-/Video-Konferenz, Dateiaustausch, Text-Chat und Audio-/Video-Streaming. Weiterhin betrachteten wir, welche Unterfälle diese Basis-Funktionalitäten jeweils mit sich führen.

GUI-Entwicklung

Um den Benutzern später eine möglichst einfache und intuitive Nutzung unseres Programms zu ermöglichen, begannen wir, eine GUI mit GTK+ an Hand unserer definierten Anwendungfälle zu konstruieren. Gerade durch das kleine Display und die spezielle grafische Oberfläche des N810 ergaben sich zusätzlicher Planungsaufwand und weitere Einarbeitungszeit. Die genauen Schnittstellen zu unserem P2P-Framework und zum Mediaplayer wurden ebenfalls untersucht. Zur Zeit steht eine Rohversion der GUI, jedoch muss diese mit dem Voranschreiten der Integration aller anderen Programmteile konstant weiterentwickelt werden.

11.2 Ausblick (Lutz)

Derzeit stellen sich zwei Schwerpunkte in der mittelfristigen Weiterentwicklung des Projekts. Da die grundsätzliche Konnektivität mehrerer Netzteilnehmer bereits implementiert und funktionsfähig ist, steht nun auf Netzebene die Implementierung von Routing-Algorithmen an. Derzeit können die Geräte in Ad-Hoc-Netzwerken nur dann miteinander kommunizieren, wenn sie sich innerhalb der physikalischen Reichweite der WLAN-Antennen befinden. Um darüber hinausgehende Konnektivität zu ermöglichen, müssen Routing-Algorithmen entworfen werden, die zum einen eine Kommunikation ermöglichen, sofern sie technisch machbar ist, und zum anderen sicherstellen, dass das Netz nicht mit Daten überflutet wird. Derartige Fluten können durch Pakete entstehen, die entweder bereits ihren Empfänger über einen anderen Weg gefunden haben oder der Zielteilnehmer nicht mehr im Netz vorhanden ist. Ein besonderes Augenmerk liegt hier auf der Propagierung von Netzteilnehmern über deren physikalische Reichweite hinaus.

Es ist hierbei noch offen, wie die Erweiterung des Netzes über die physikalische Reichweite hinaus realisiert werden soll. Eine Möglichkeit besteht in der simplen Weiterleitung der Daten durch erneutes Senden der Anfragen, Antworten und der Nutzdaten. Eine Alternative wäre ein virtuelles Zueigenmachen der Daten anderer Peers: Falls auf eine Anfrage hin der Peer, der die Daten anbietet, nicht antwortet, kann mit den zuvor zwischengespeicherten Daten geantwortet werden. Für den ursprünglichen Anfrager bliebe die Weiterleitung transparent, für ihn sähe es so aus, als wenn der Weiterleitende Peer die Daten anbieten würde. Hierfür würde der Routing-Peer als eine Art Proxy fungieren.

Der andere Schwerpunkt liegt auf der Implementierung der eigentlichen Funktionalität der Software, des Datenaustauschs. Bisher ist ein Datenaustausch nur manuell durch Ausführung bestimmter GNUnet-Applikationen auf der Kommandozeile möglich. Der Ausbau des Datenaustauschs soll also schrittweise durchgeführt und erweitert werden.

Zunächst ist geplant, einfache Textnachrichten zwischen den Teilnehmern austauschen zu können, um einen Chat zu ermöglichen. Hierbei besteht allerdings das Problem, dass das GNUnet-Framework lediglich den Abruf von Dateien vorsieht und ein Senden auf Push-Basis überhaupt nicht vorgesehen ist. Als Workaround könnte allenfalls eine fiktive „Chat“-Datei dienen, die zum Initiieren eines Chats vom Kommunikationspartner „heruntergeladen“ wird und dann den Chat durch den Transfer temporärer Textdateien zu ermöglichen.

Im weiteren Verlauf soll es dann möglich sein, freigegebene Dateien anderer Nutzer zu durchsuchen und herunterzuladen. Hier sind keine größeren Hürden zu erwarten, da dies die zentrale Funktionalität des GNUnet-Frameworks darstellt. Es ist lediglich die Kommunikation zwischen Framework und unserer GUI-Schicht zu implementieren, um Suchanfragen starten und deren Ergebnisse ansehnlich präsentieren zu können. An dieser Stelle steht die Überlegung, welche Metainformationen über die zu verteilenden Dateien bereitgestellt werden und in welchem Umfang diese genutzt werden. Die Nutzung aller von libextractor unterstützten Formate und Metadaten erscheint nicht vorteilhaft, weil

der alleinige Umfang der Bibliothek unsere Applikation übermäßig vergrößern würde. Bis hierhin ist es nicht zu erwarten, dass Performance-Probleme auftauchen werden.

Auf Basis dieses normalen Datei-Austauschs soll als dritter Schritt das dateibasierte Streaming von Multimedia-Inhalten ermöglicht werden. Hier kann es dazu kommen, dass durch rechenintensive Audio- bzw. Video-Codecs ein flüssiges Funktionieren des Gesamtsystems in Frage gestellt wird. Für die Darstellung der Multimedia-Inhalte wurde die Einbettung des MPlayer erwogen. Hier ist zu gewährleisten, dass eine eventuelle Systemüberlastung das Zusammenspiel der verschiedenen Softwareteile nicht verhindert und die Stabilität des Systems gesichert ist. Dieser dritte Schritt erfordert über die Funktionalität innerhalb unserer GUI hinaus auch noch Eingriffe in das GNUnet-Framework, da dort ein Dateistreaming nicht vorgesehen ist und keinerlei Mechanismen bestehen, die eine gewisse Bandbreite garantieren können. Weiter muss ein Weg gefunden werden, wie die eingehenden Stream-Daten von GNUnet an den MPlayer zur Anzeige weitergegeben werden, da der sendende Peer keinen Streaming-Server zur Verfügung stellt, der vom MPlayer direkt angesprochen werden könnte.

Über das dateibasierte Streaming hinaus ist noch geplant, auf dieser Grundlage eine direkte Audio- oder Videokommunikation zu ermöglichen. Hier muss dann beim sendenden Peer ebenfalls dafür gesorgt werden, dass die Daten vom Mikrofon bzw. von der Kamera an GNUnet zum Versand geleitet werden.

Allerdings erwarten wir hier im Bereich der Performance Probleme, da bereits das Anzeigen der Daten der geräteeigenen Webcam nicht ruckelfrei abläuft. Ob ein Auslesen der Webcam-Daten in geringerer Auflösung möglich ist, damit der Kodierungs-Prozess diese in Echtzeit bearbeiten kann, ist noch zu Testen und eine Grundvoraussetzung für das erfolgreiche Funktionieren dieses Teils.

Im Rahmen dieser direkten Kommunikation bietet sich auch an, Dateien auf Push-Basis zu übertragen. Ähnlich wie im Chat-Modus wird dann auch hier mit virtuellen Dateien gearbeitet. Ein Anfordern dieser Datei durch den Sender würde dann beim Empfänger ein Akzeptieren des Dateiempfangs aufrufen. Daraufhin kann der Empfänger dann die Datei mittels einer normalen Anfrage herunterladen.

Anhang

A Zusammenfassungen der Seminarthemen

Dieses Kapitel beschäftigt sich mit den Inhalten, die im Rahmen eines dreitägigen Seminars vor Beginn der eigentlichen Realisierungsarbeit vorgetragen wurden. In diesem Seminar wurde durch die PG-Mitglieder erarbeitetes Grundlagenwissen zur Realisierung von UbiMuC präsentiert. Dieses umfasste theoretische Grundlagen und die Vorstellung praktischer Implementierungen verschiedener Programme. Letzteres war vor allem nötig, da die komplette Implementierung aller für UbiMuC gewünschter Funktionen den Rahmen der PG weit überschritten hätte. Dieses Kapitel fasst nun sämtliche Seminarthemen jeweils kurz zusammen. Die Reihenfolge der Themen entspricht dabei der chronologischen Reihenfolge, in der die Themen während des Seminars abgehandelt wurden.

A.1 P2P (Fabian)

Da im Antrag der Projektgruppe verankert ist, dass als grundlegende Netzwerkstruktur Peer-to-Peer-Netze verwendet werden sollen, musste geklärt werden, welche Arten von Peer-to-Peer-Netzen es gibt [2]. Aufgabe dieses Seminarthemas war sowohl die Erläuterung der Unterschiede zwischen den einzelnen Arten, als auch erste grobe Überblicke über die mögliche Umsetzung zu schaffen.

Als Modelle für Peer-to-Peer-Netzwerke wurden folgende Varianten vorgestellt:

- reine Peer-to-Peer-Netzwerke
- serverbasierte Peer-to-Peer-Netzwerke
- hybride Peer-to-Peer-Netzwerke

Zusätzlich zu den Varianten wurden Umsetzungsmöglichkeiten zur Verwaltung von Informationen, Dateien und Ressourcen vorgestellt. Einige davon werden im weiteren Projekt benutzt und an gegebener Stelle detaillierter beschrieben. Vorgestellt wurden [2]:

- Distributed Hash Table (DHT)
- Content Addressable Network (CAN) [3]
- CHORD [4]
- Tapestry und Pastry
- Viceroy, Koorde und Distance Halving

Am Ende des Vortrages wurden noch ganz kurz die Probleme in Peer-to-Peer-Netzwerken angesprochen, um die sich bei der Implementierung Gedanken gemacht werden sollten. Diese Probleme sind die Folgenden [2]:

- Handhabbarkeit
- Informationskohärenz
- Fehlertoleranz
- Sicherheit
- Skalierbarkeit

Als endgültiges Fazit bleibt festzuhalten, dass die Umsetzung in ein Peer-to-Peer-Netzwerk sehr viel versprechend klingt, jedoch die oben genannten Dinge beachtet werden müssen.

A.2 JXTA (Michael K.)

UbiMuC soll mittels eines Peer-to-Peer-Netzwerkes die Kommunikation zwischen den einzelnen Teilnehmern während des Streamings ermöglichen. Dabei gibt es zwei Wege, die man einschlagen kann. Einerseits könnte man sich ein eigenes Kommunikationsprotokoll schreiben, andererseits könnte man auf bereits bestehende P2P-Frameworks zurückgreifen. Da die erste Wahl sicherlich viel unnötige Arbeit und Probleme hervorrufen würde, muss ein Framework gefunden werden, das unseren Ansprüchen genügt und einfach zu handhaben ist. Das wohl bekannteste open-source Framework dieser Art wurde von Sun Microsystems[5] ins Leben gerufen und nennt sich JXTA[6]. Dieses Seminarthema befasst sich mit den grundlegenden Eigenschaften von JXTA und evaluiert die Laufzeit- und Bandbreitenänderung gegenüber verschiedenen Netzwerkprotokollschichten.

JXTA implementiert die Basisfunktionalitäten eines P2P-Netzwerks. Es besteht aus einer Menge allgemeiner Peer-to-Peer Protokolle, die es erlauben, dass verschiedene Geräte im Netzwerk miteinander kommunizieren können. Diese Geräte müssen nicht zwangsläufig handelsübliche Personal Computer sein. Auch Mobiltelefone, Server, PDAs uvm. können als Peers teilnehmen. Die JXTA-Protokolle sind der Kern des Projektes, da sie unabhängig von Programmiersprache und darunterliegenden Transportprotokollen arbeiten. Die Protokolle sind:

- Peer Resolver Protocol
- Endpoint Routing Protocol
- Peer Discovery Protocol
- Rendezvous Protocol
- Pipe Information Protocol
- Pipe Binding Protocol

Die bisherige Vorarbeit im Laufe der Seminarphase behandelt JXTA im allgemeinen bzw. theoretischen Kontext. Dabei wird kein Unterschied zwischen der Java und der C-Version[7] gemacht. Für das weitere Vorgehen im Bezug auf das Internet Tablet müssen

genauere Informationen zur C Version erlangt werden. Dieses Wissen wurde im späteren Verlauf der Projektgruppe erlangt und ist in Kapitel 5.1 festgehalten.

A.3 State-of-the-art Multimedia Codecs und -Standards (Stefan)

Motivation

In den letzten Jahren sind diverse Datenkompressionsverfahren für Audio- und Videosignale als internationale Standards definiert worden. Den größten Anteil daran hatten die Internationale Fernmeldeunion (ITU) und die von der Internationalen Standardisierungsorganisation (ISO) eingesetzte Moving Picture Experts Group (MPEG). Durch die rasante Entwicklung des Internets steigt auch der Bedarf an multimedialen Diensten. Bild-, Audio- und Videodaten haben allerdings im Rohformat einen enormen Speicherbedarf, der mit den Internetbandbreiten nicht immer vereinbar ist. Auch in anderen Bereichen reichen die Bandbreiten nicht aus, um die Daten im Rohformat zu übertragen, wie z.B. bei dem Rundfunkstandard DVB oder beim Übertragen von Multimediainhalten über die Mobilfunknetze GSM und UMTS. Daher sind Datenkompressionsverfahren für multimediale Inhalte und somit auch für das Projekt UbiMuC wichtig.

Audio

Der MPEG-1 Audio Standard wurde Anfang der 90er Jahre von der „Moving Picture Experts Group“ (MPEG) verabschiedet, um einen einheitlichen Standard zur Speicherung und Übertragung von komprimierten Audiodateien zu gewährleisten. Es wurden 3 verschiedene Encoder/Decoder definiert, welche als Layer I - III bezeichnet werden. Die verschiedenen Layer sind abwärtskompatibel, d.h. ein Layer III Decoder kann auch die beiden vorherigen Layer dekodieren. Die Komprimierung basiert auf dem psychoakustischen Modell, bei dem vom Gehör nicht wahrnehmbare Anteile des Audiosignals auch nicht gespeichert werden.

OGG Vorbis befindet sich seit ca. 1993 in der Entwicklung. Die Xiphophorous Foundation (xiph.org) ist der Entwickler. Eine erste Version des lizenzfrei verfügbaren Encoders gibt es seit 2002. Während der MP3-Codec bei 64 kbit/s schlechte Qualität liefert, erreicht der Vorbis-Encoder bei praktisch gleicher Dateigröße eine erheblich bessere Signalqualität. Ogg Vorbis konnte sich aber nie richtig gegen den Gewohnheitsstandard MP3 durchsetzen, hat sich dahinter aber fest etabliert.

Video

Videocodecs gibt es recht viele, die meisten dienen aber dem Kodieren von Filmen in Dateien. Mit H.261 und H.263 hat die ITU zwei Videostandards herausgebracht, die

hauptsächlich für Videokonferenzen eingesetzt werden. Während die Codecs für Videodateien meist als fertige Kodier- und Dekodierprogramme existieren, gibt es für H.261 und H.263 sowohl Definitionen der Formate als auch Referenzdekoder. Welche Features eine Implementierung eines Enkoders oder Dekoders unterstützt, liegt in den Händen der Entwickler. Dies macht es möglich, dass es viele verschiedene Programme gibt, die miteinander kommunizieren können. Der Standard H.262 wurde 1994 zusammen mit der Motion Picture Experts Group eingeführt und ist besser bekannt als MPEG-2. Große Verbreitung hat dieser Standard durch die Einführung der DVD erhalten, die in MPEG-2 kodiert ist. Das MPEG-2 Format wird ebenfalls bei SVCDs und beim digitalen Fernsehen (DVB-S, DVB-T, DVB-C) genutzt. Das MPEG-2 Format setzt auf dem MPEG-1 Standard auf und ist somit auch abwärtskompatibel. H.264 ist ein Standard zur hocheffizienten Videokompression, der eine etwa dreimal so hohe Codiereffizienz wie MPEG-2 erreicht und gerade deshalb für hochauflösende Bilddaten ausgelegt ist. Der H.264 Standard ist als zehnter Teil in dem MPEG 4 Standard eingebettet (MPEG-4/Part 10, ISO/IEC 14496-10).

Fazit

MPEG-4 ist der erste Multimediastandard, der inhaltsorientiert arbeitet und viele Möglichkeiten zulässt, die durch die immer stärkere Verschmelzungen von Technologien gebraucht werden. Durch die Standardisierung mit der ISO-Norm finden Teile des MPEG-4 Standards bei vielen Endgeräten Verwendung, wie z.B MPEG-4 AVC/H.264 bei BluRay, PlayStation Portable und IPod. Ganz wichtig ist der Standard auch für kommende Systeme wie z.B für HDTV, DVB-H und DVB-S2. DVB-S2 ist der Nachfolgestandard zum digitalen Rundfunk per Satellit, der neben neuen Modulationen auch den Codec H.264 verwendet, um höhere Auflösungen und bessere Bildqualität zu liefern. Auch im Internet setzt man verstärkt auf den neuen Codec. So wurde H.264 mittlerweile auch im Flash Player von Adobe implementiert, welcher sich zum Standard für Videodarstellung auf Internetseiten entwickelt hat (z.B. bei Youtube.de oder bei Nachrichtenseiten wie Spiegel-Online.de und Tagesschau.de).

A.4 Streaming Protokolle (Michael P.)

Motivation

Die Netzwerke, die auf IP basieren sind erst einmal nicht auf Echtzeitkommunikation ausgelegt. In der heutigen Zeit wo es immer mehr Daten wie Auto und Video im Internet in Echtzeit übertragen werden, ermöglichen Streaming-Protokolle die Nutzung von multimedialen Echtzeit-Anwendungen. Dabei müssen Anforderungen wie Bandbreite, Latenzzeit oder gewisse Fehlertoleranz in diesem Zussamenhang beachtet werden. Als Streaming-Protokoll bezeichnet man einen speziellen Protokoll der für Übertragung von Streaming Media Daten über das Netzwerk verwendet wird. Dabei werden die Media-Daten mittels eines Kodierers vor dem Senden in einen Streaming-Format konvertiert.

Dabei handelt es sich in der Regel um Container für diverse mit Streaming-Codecs komprimierten Formate. Es gibt freie Streaming-Formate wie z.B. Ogg-Vorbis oder proprietäre Streaming-Formate wie Quicktime von Apple oder Windows Media Audio (WMA) oder Windows Media Video (WMV) von Microsoft.

Protokolle

Einige der wichtigsten Protokolle in Bezug auf Streaming in Netzwerken sind:

- Real-time Transport Protokoll (RTP)
- Real-Time Streaming Protokoll (RTSP)
- Hypertext Transfer Protokoll (HTTP)

Das Real-time Protokoll und das Real-time Streaming Protokoll bilden die Grundlage für die Übertragung von audiovisuellen Medien über das Web. RTP ist ein Paket-basiertes Protokoll und wird zur kontinuierlichen Übertragung von audiovisuellen Daten (genannt Streams) meinst unter der verwendung von UDP über ein IP-basiertes Netzwerk verwendet. Das Protokoll ist in der Lage, Multimedia-Datenströme wie Audio oder Video, zu übertragen. RTSP ist ähnlich wie RTP auf den Transport von audiovisuellen Daten ausgelegt. Die Hauptaufgabe des Protokolls ist die Steuerung des Datenstroms innerhalb von einer Session zwischen Sender und Empfänger. Das Protokoll kann sowohl über UDP als auch über TCP übertragen werden. RTSP ähnelt vom Aufbau dem HTTP. Im Gegensatz zu HTTP achtet RTSP aber auf die Zeitlinie von Streams, dadurch wird die Betrachtung und Steuerung von mehreren Streams zeitgleich ermöglicht. Das Hypertext Transfer Protokoll dagegen steht im Mittelpunkt des gesamten World Wide Webs. Das Protokoll spielt bei der Darstellung von Webpages im Internet eine entscheidende Rolle. HTTP ist in der Lage, nahezu jedes Objekt, das auf einer Webpage dargestellt werden sollte, zu verkapseln und dieses im Zuge dessen von einem Server zum Client zu transportieren. Außerdem liegen viele Netzwerke hinter Firewalls, so dass nicht alle Streaming Protokolle 'durchgelassen' werden. In diesen Fällen weicht man auf HTTP aus.

Fazit

RTP, RTSP und HTTP: auf diesen Protokollen basiert zum Teil das gesamte Internet. Im Rahmen der Projektgruppe wo es um die Übertragung von audiovisuellen Signalen und Echtzeit-Charakteristiken von Anwendungen geht werden die angesprochenen Protokolle von Bedeutung sein.

A.5 Vergleich, Test und Evaluierung von Streaming-Servern und -Clients (Markus G.)

Da allein vom Umfang der Funktionen, die das UbiMuC-Framework später unterstützen soll, absehbar war, dass nicht alle Komponenten von Grund auf neu implementiert werden können, befassten sich einige Seminararbeiten mit Open-Source-Software und deren Verwendbarkeit zur Realisierung des anstehenden Projektes. Gegenstand dieses Seminarthemas war der Test verschiedener quelloffener Streaming-Server und -Clients mit der Absicht, später auf ihrer Grundlage das angestrebte Streaming zwischen zwei Peers im UbiMuC-Framework zu realisieren. Getestet wurden dabei folgende Programme:

Server:

- Darwin Streaming Server [8]
- LScube [9]
- VLC VideoLan Media-Player [10]

Clients:

- MPlayer [11]
- VLC VideoLan Media-Player [10]

Diese Programme wurden zunächst auf ihre Eigenschaften wie Portierbarkeit, Lizenz, eingebundene Codecs, und spezielle Eigenheiten überprüft. Hierauf folgte ein praktischer Speicherbelegungstest, der die Speicherlast und das Verhalten der Programme unter verschiedenen Bedingungen (zum Beispiel mehrere verbundene Clients bei einem Test eines Server-Programms) evaluieren sollten. Bei einer zu hohen Last auf einem Desktop-System wäre das betroffene Programm nicht in die engere Wahl der Kandidaten zur Portierung auf das N810 gekommen.

Durch die Test ließ sich kein Programm ausmachen, das in allen für das Projekt relevanten Punkten den anderen Programmen überlegen war. Es bleibt jedoch festzuhalten, dass der Darwin Streaming Server aufgrund lizenztechnischer Probleme nicht weiter betrachtet werden muss, da seine Lizenz nicht kompatibel zur GPL ist, unter der UbiMuC entwickelt werden soll. Der VLC VideoLan Media-Player benötigte in den Tests schon so viel Speicher, dass effektiver Einsatz auf eingebetteten Systemen nicht zu erwarten ist. Der intern verwendete Codec FFmpeg verletzt gegebenenfalls Softwarepatente, wie sie in einigen Ländern vergeben werden. Da auch MPlayer FFmpeg verwendet besteht hier das gleiche Problem. LScube hingegen befindet sich noch in einer frühen Entwicklungsphase und lief teilweise erwartungsgemäß äußerst instabil. Somit war weitere Recherchearbeit notwendig, bevor im Plenum über die verwendete Streamingsoftware abgestimmt werden konnte.

A.6 Allgemeine Sicherheitsalgorithmen (Jens)

Motivation

Um im Zeitalter der digitalen Kommunikation und Geschäftsabwicklung über das Internet ein gewisses Maß an Vertraulichkeit und Integrität zu gewährleisten, sind Verschlüsselungsalgorithmen das nötige Rüstzeug: Auftragsdaten, Überweisungen und ähnliche brisante Informationen sollten keinesfalls im Klartext über offene Netzwerke versendet werden, sodass in jedem Fall eine Schutzfunktion nötig ist, welche die Lesbarkeit von unbefugten Dritten ausschließt. Je nach Anwendungsszenario realisieren Verschlüsselungsalgorithmen die Integrität (Verfälschungsschutz) von Nachrichten, ebenso wie sie die Authentizität (Absenderverbindlichkeit) sicherstellen. Dabei ist es besonders wichtig, dass passende Algorithmen und Verfahren für den jeweiligen Fall eingesetz werden, da es keinen „Universalalgorithmus“ gibt. Jede Verschlüsselungsmethode hat ihre eigenen Stärken und Schwächen und qualifiziert sich demnach mehr oder weniger für spezielle Szenarien. Neben der Auswahl von geeigneten Methoden zur Realisierung einer „sicheren“ Kommunikation steht auch die Reife der Algorithmen im Vordergrund. Es empfiehlt sich, wie zahlreiche Lehrbeispiele der Vergangenheit zeigen, grundsätzlich nur auf ausgereifte und weiträumig getestete und untersuchte Algorithmen zu setzen, da diese ein gutes Maß an Robustheit gegenüber vielschichtigen Angriffen bieten. Neuentwicklungen sind oft mit Vorsicht zu genießen, genauso, wie der Einsatz von geheim gehaltenen Verfahren nicht empfohlen wird. In den ersten Jahren ist hier zwar ein gewisser Schutz durch die fehlende inhaltliche Kenntnis des Algorithmus gegeben, mit zunehmendem Interesse am Verfahren wachsen jedoch die Chancen der Aufdeckung von Sicherheitsproblemen. Verschlüsselungsalgorithmen können nur dann qualifiziert beurteilt werden, wenn die zu Grunde liegenden mathematischen Verfahren für jedermann zugänglich sind.

Algorithmen

Im einzelnen geht die Seminararbeit auf zwei sehr bekannte Verfahren ein, den „Data Encryption Standard“ (DES) [12] und seinen Nachfolger „Advanced Encryption Standard“ (AES) [13]. Diese beiden Algorithmen sind symmetrisch aufgebaut, d.h. für die Operation der Ver- und Entschlüsselung werden identische Schlüssel verwendet. Aus diesem Grund entsteht hier das Problem des so genannten „Schlüsseltauschs“. Genau dort setzen asymmetrische Verfahren an, welche mit unterschiedlichen Schlüsseln (public/private keys) arbeiten. Weil aber asymmetrische Verschlüsselungen grundsätzlich langsamer sind als die symmetrischen Pendants, werden sie meistens nur zum Schlüsseltausch benutzt, während die eigentliche Verschlüsselung der Kommunikationssitzung dann mit einem symmetrischen Verfahren wie AES durchgeführt wird. Neben diesem Verwendungszweck finden Verfahren wie RSA [14] und DSA [15] allerdings auch ihren Einsatz in Signatursystemen: Dort gewährleisten sie die Verbindlichkeit von Nachrichten, ebenso wie deren Integrität.

Fazit

Aktuelle Lösungen zur Einrichtung einer sicheren Kommunikation zwischen mehreren Teilnehmern sollten unbedingt auf Kombinationen aus symmetrischen und asymmetrischen Verfahren setzen. Durch asymmetrische Verfahren lassen sich Sitzungsschlüssel für symmetrische Algorithmen austauschen, die anschließend für einen verschlüsselten Datenverkehr sorgen. Digitale Signaturen und Public-Key-Systeme steuern zusätzlich die Integrität und Verbindlichkeit der Nachrichten bei, sodass ein ausreichend abgeschotteter Kommunikationskanal über ein unsicheres Netzwerk eingerichtet werden kann. In Bezug zur Projektgruppe empfiehlt sich daher besagte Kombination beider Verfahren, um eine sichere Kommunikation über ein potentiell unsicheres Medium zu realisieren.

A.7 Sicherheitsalgorithmen für eingebettete Systeme (Markus S.)

Motivation

Im Bereich der eingebetteten Systeme sind, wie in so ziemlich allen Bereichen der Informatik, Verschlüsselungsverfahren eines der grundlegenden Mittel, um die Sicherheit und Vertraulichkeit von verarbeiteten und transportierten Daten zu gewährleisten. Jedoch hat man in diesem Bereich mit anderen Qualitätsanforderungen an Software zu tun, bedingt durch eingeschränktere Ressourcen und andere Zielsetzungen wie z. B. Minimierung des Stromverbrauchs von Geräten. 1976 wurde mit der Publizierung des Konzepts der Public-Key-Verschlüsselung ein wichtiger Schritt in der modernen Kryptographie getan. Diese Verfahren verwendet meist die Schwierigkeit der Lösung des diskreten Logarithmus, um die zur asynchronen Verschlüsslung nötigen Einweg-Funktionen zu erzeugen. Das Verfahren findet heutzutage Anwendung in einer Vielzahl von Standards und Protokollen, die zur Verschlüsselung und Verifikation genutzt werden. Übliche Varianten des Public-Key-Verschlüsselungsverfahrens erweisen sich im Kontext der eingebetteten Systeme auf Grund der eingeschränkten Ressourcen allerdings als recht problematisch. Gerade die Schlüssellängen von 1024 Bit und mehr, die für eine sichere Verschlüsselung benötigt werden, erzeugen bei üblichen Architekturen mit sehr geringen Befehlswortbreiten sehr viel Overhead. Die Ausarbeitung stellt zwei neuere Public-Key-Verfahren vor, die mit weit kürzeren Schlüsseln ähnlich hohe Sicherheit gewährleisten. Diese Verfahren basieren beide auf dem Konzept der elliptischen Kurven, daher geht die Ausarbeitung auf mathematische Eigenschaften dieser Kurven ein und stellt dar, wie diese zur Verschlüsselung genutzt werden können.

Verfahren

Elliptische Kurven sind Abbildungen von der Form $y^2 = x^3 + ax + b$ mit x, y, a und b aus einem Körper K. Die Punkte auf dem Graphen dieser Abbildung könnten zusammen mit einer auf geometrischen Eigenschaften basierenden definierten Addition als additive

Gruppe aufgefasst werden. Zu dieser Addition wird weiterhin eine skalare Multiplikation definiert, so dass wir mit einer abelschen Gruppe arbeiten können. Für die Verschlüsselung beschränken wir uns auf diskrete Mengen.

- **Elliptic Curves Cryptography** Dieses Verfahren verwendet, wie bereits dargestellt, die Berechnung des diskreten Logarithmus, angepasst an die beschriebenen Berechnungsverfahren der elliptischen Kurven. Dadurch ist der übliche Schlüsselaustausch der Public-Key-Verschlüsselung mit privatem und öffentlichem Schlüssel in leicht modifizierter Form umsetzbar. Auf Grund der höheren Komplexität der Basis-Operationen können bei diesen Verfahren Schlüssel weit geringerer Länge verwendet werden als bei anderen Public-Key-Verfahren bei ähnlich hoher Sicherheit.
- **Hyper Elliptic Curves Cryptography** Dieses Verfahren erweitert das vorherige auf elliptische Kurven höherer Ordnungen. Unter Ausnutzung spezieller mathematischer Strukturen kann auf diesen ebenfalls eine abelsche Gruppe gebildet werden und damit ein modifiziertes Public-Key-Verfahren aufgebaut werden.

Der Vorteil dieser Verfahren liegt darin, dass der Engpass, der bei vielen eingebetteten Systemen durch geringe Befehlswortbreiten und übermäßig lange Schlüssel entsteht, gemindert oder umgangen werden kann. Gerade mit dem 2. Verfahren ist es möglich, die Schlüssellängen stark zu senken ohne an Sicherheit im Vergleich zu üblichen Systemen zu verlieren. Der Trade-off besteht allerdings darin, dass Anzahl und Komplexität der Basis-Operationen stark ansteigen, und damit auch die Last auf das System bei der Verschlüsselung. Der effiziente Einsatz der Verfahren hängt daher stark von der Optimierung der verwendeten Datenstrukturen und mathematischen Algorithmen zur Berechnung der Basis-Operationen ab.

Nachteile dieser Ansätze liegen für uns ganz klar in der Zusatzlast, die beim Einsatz der Verfahren entstehen würde. Da unsere Anwendungsfälle von Streaming mit paralleler Verschlüsselung ausgehen, müssen wir schon ohne Verschlüsselung von einer starken Auslastung unser Hardware bei Normalbetrieb ausgehen. Auf Grund der mit dem Streaming verbundenen Echtzeitanforderungen stellen sich die Verfahren als recht problematisch dar. Ein weiteres Problem stellt die unklare rechtliche Lage dar. Verschiedene Firmen haben auf Teiloptimierungen für geeignete Datenstrukturen und Algorithmen Patente angemeldet, woraus bereits Klagen wegen unerlaubter Nutzung resultierten. Die Folge ist eine sehr geringe Akzeptanz und geringe Verbreitung der Verfahren.

Fazit

Auf Grund unserer Aufgabenstellung, Multimedia-Streaming mit paralleler Verschlüsselung auf Hardware mit geringer Prozessorleistung bzw. mit geringem Speichervolumen zu realisieren, wiegen die Nachteile, welche die vorgestellten Verschlüsselungsverfahren mit sich bringen, stärker als die Vorteile. Daher wurde entschieden, diese Verfahren nicht weiter für das Projekt zu berücksichtigen und andere etablierte Standards zu verwenden.

Die unklare Rechtslage war ein weiterer Grund, weshalb wir keine Möglichkeit sahen, diese Verfahren einzusetzen.

A.8 Sicherheitsprotokolle und Implementation (Björn)

Motivation

Wie bei allen Funkanwendungen, kann man das Übertragungsmedium, anders als bei kabelgebundenen Netzwerken, nur relativ schlecht überwachen. Ein sicherer Kommunikationskanal spielt aber gerade in (Video-) Telefonie-Anwendungen eine besonders wichtige Rolle. Da das Implementieren eigener Sicherheitsprotokolle allerdings aufwändig und fehlerträchtig ist, beschäftigte sich dieses Seminarthema mit bereits existierenden Sicherheitsprotokollen, die bei Bedarf relativ einfach eingebunden werden können.

Grundlagen

Verschlüsselung allein reicht für eine *sichere* Verbindung nicht aus, da der Aufbau eines verschlüsselten Kanals beispielsweise anfällig für *Man-in-the-Middle*-Angriffe ist [17]. Die Verschlüsselung verhindert nur das Abhören einer bestehenden Verbindung. Die *Authentifizierung* ist also nötig, um die Gefahren bei Verbindungsaufbau abwenden zu können. Damit wird sichergestellt, dass man nicht aus Versehen mit nicht-vertrauenswürdigen Personen kommuniziert, die den gesamten Aufwand der Verschlüsselung hinfällig machen würden. Beide Verfahren verhindern aber nicht, dass übertragene Daten von Dritten manipulierbar sind. Trotz Verschlüsselung ist man ohne Weiteres oft in der Lage, einzelne Bits gezielt kippen zu können, was beispielsweise Angriffe auf immer gleich aussehenden Formularen erleichtern würde. Die *Integrität* der Nachricht muss also zusätzlich mit *Digitalen Signaturen* sichergestellt werden.

OpenSSL

Eine Implementierung des wohl bekanntesten Sicherheitsprotokolls *SSL* (*Secure Socket Layer*), seit 1999 auch *TLS* (*Transport Layer Security* [16]) genannt, ist ohne Zweifel das *OpenSSL-Toolkit*, das einst für den *Netscape Navigator* entwickelt wurde und somit eine relativ lange Geschichte besitzt.

Das OpenSSL-Toolkit besteht im Grunde genommen aus zwei Teilen: der API, die insbesondere für Applikationsentwickler interessant sein sollte, und *openssl*, das vielseitige Kommandozeilentool u. a. für die Schlüsselverwaltung oder dem Testen von SSL-Anwendungen. Mit beiden Teilen sollten auch Entwickler mit nur wenig Erfahrung auf dem Gebiet der Kryptographie in der Lage sein, sichere Verbindungen zu nutzen, da die teils undurchsichtigen Interna der verwendeten Algorithmen von OpenSSL gekapselt werden.

Fazit

Mit OpenSSL hätten wir eine Möglichkeit, *TCP*-Verbindungen zuverlässig abzusichern. Je nach Algorithmenwahl ist der Ressourcenverbrauch verschwindend gering, so dass sich die Bibliothek auch auf mobilen Endgeräten einsetzen lässt.

Ein Nachteil ist allerdings, dass sich damit keine paketorientieren *UDP*-Verbindungen absichern lassen, die häufig im Bereich Multimedia-Streaming eingesetzt werden. Ein weiterer Punkt wäre die *OpenSSL-Lizenz*, die zwar grundsätzlich frei, aber wegen einer Reglung, dass man OpenSSL bei Verwendung in einem Produkt explizit erwähnen muss, nicht mit der *GPL* kompatibel ist. Dennoch lässt es sich ohne Bedenken mit einer *GPL*-Anwendung linken, falls OpenSSL, wie z. B. auch auf der Maemo-Platform, Teil des Betriebssystems ist.

Da sich *TCP*-Sockets mit nur wenig Aufwand durch *TLS*-Objekte ersetzen lassen, bleibt der Einsatz von OpenSSL auch später noch eine Option, die sich bei Bedarf noch umsetzen lässt.

A.9 C/C++ (Stephan)

In unserer bisherigen Laufbahn an der Universität wurde uns vor allem eine Programmiersprache beigebracht: Java. Warum also soll man sich nun mit einer anderen Sprache beschäftigen?

Dies liegt daran, dass Java - trotz einiger guter Ansätze - über einige Schwachstellen (bezogen auf unsere Aufgabenstellung) verfügt. „Komfortfunktionen“ wie der „garbage collector“ oder die Virtual Machine machen Java nicht nur langsam, sie trennen die laufenden Programme auch vom Rest des Betriebssystems. Für Anwendungen auf Desktop PCs und Servern, bei denen es nicht allzu stark auf Performance ankommt, mag dies die Programmierung vereinfachen. Für andere Aufgaben - wie z.B. die Programmierung eingebetteter Systeme - ist dies aber eher hinderlich.

Anstelle eines garbage collectors kann die Speicherverwaltung (also das Allokieren und Freigeben von Speicherplatz) durch den Programmierer geschehen, was die laufzeitintensive garbage collection überflüssig macht. Das Wegfallen der VM erlaubt direkten Zugriff auf das Betriebssystem und den Arbeitsspeicher.

Weitere Gründe für C/C++ sind die weite Verbreitung und damit große Verfügbarkeit von Bibliotheken und (Opensource-)Programmen und die Verfügbarkeit von Compilern für alle gängigen Architekturen und Geräten. All dies führt dazu, dass wir uns näher mit C/C++ beschäftigen.

A.10 N810 (Nils)

Einleitung

Thema dieses Abschnitts ist die Hardware- und Software-Plattform des N810. Insgesamt wird die Betrachtung des N810 in diesem Kapitel recht kurz gehalten, da im Folgenden in Kapitel B näher auf die der Projektgruppe zugrundeliegende Hardware eingegangen wird. Dabei wird dann auch das Verhalten in für die Projektgruppe relevanten Situationen behandelt.

Bei dem von der Firma Nokia entwickelten N810 handelt es sich um ein sogenanntes „Internet-Tablet“. Gedachter Einsatzzweck ist das mobile Surfen im Internet, sowie die problemlose Kommunikation über das Internet, ohne dafür extra ein Notebook oder gar einen Desktop-Computer verfügbar zu haben.

Hardware

Den Kern des N810 bildet der OMAP2420 Chip der Firma Texas Insturments [18]. Verbindung zur Außenwelt erhält man hauptsächlich über das dem IEEE 802.11 b/g Standard folgende WLAN Modul oder via Bluetooth. Da kein GSM Chip verbaut ist, kann es nicht als gewöhnliches Mobiltelefon verwendet werden. Die Eingabe erfolgt wahlweise über den Touchscreen oder über eine ausschiebbare Tastatur.

Software

Auf dem N810 kommt das von Nokia als „Internet Tablet OS 2008“ bezeichnete Betriebssystem zum Einsatz. Dieses basiert auf „Maemo-Linux“, welches selbst von Debian abstammt. Die verwendete Maemo Version heißt Maemo 4.0 und trägt den Codenamen „chinook“. Dem System liegt ein handelsüblicher Linux Kernel der 2.6er Reihe zugrunde.

Die grafische Oberfläche wird mithilfe von X11 dargestellt, und sowohl ein Mediaplayer als auch ein Browser und Kommunikationssoftware sind entweder schon vorinstalliert oder durch den Nutzer nachträglich installierbar.

Das gesamte System selbst baut dabei weitestgehend auf nur leicht modifizierten Standardkomponenten auf, wie sie auch bei vielen Linuxdistributionen zu finden sind. Entsprechend sind viele der zur nachträglichen Installation angebotenen Programme Portierungen von gebräuchlicher Software wie zum Beispiel dem MPlayer oder von Pidgin, einer Multiprotokoll-Chat-Software.

Fazit

Insgesamt bildet das N810 von Nokia eine vielversprechende Hardware-/Software-Plattform. Schon jetzt werden viele mögliche Anwendungen auf dem Tablet unterstützt und man kann recht leicht neue Anwendungen für das N810 entwickeln beziehungsweise bestehende Linux Software portieren. Allerdings handelt es sich um ein eingebettetes System, weshalb man entsprechend auf Geschwindigkeits- und Ressourcenoptimierungen sowie die besonderen Eingabegeräte achten muss.

A.11 WLAN-Netze (Thomas)

Motivation

Die wachsende Verbreitung drahtloser lokaler Netzwerke, so genannter WLANs (Wireless Local Area Network), zur breitbandigen Datenübertragung erfreut sich weltweit immer größerer Beliebtheit. Die kabellose Technik ist für jedermann erschwinglich geworden und ein weiterer Preisverfall ist für die Zukunft zu erwarten. Auch die Installation und Bedienung sind einfach gehalten, so dass die Zahl der WLAN Nutzer weiter schnell wächst. Für die Vergrößerung der WLAN-Infrastruktur sind aber nicht nur die privaten Nutzer zuständig. Auch Telekommunikationsanbieter, wie z.B. T-Mobile, sorgen mit ihren öffentlichen Netzwerken, den so genannten Hotspots für die Erweiterung der Struktur und einen weltweiten Zugang. Viele Mobilfunkanbieter reagieren auf diese Änderung, indem sie die Geräte mit einer WLAN-Schnittstelle ausstatten, wie es bei unserem Gerät, dem N810 von Nokia, der Fall ist. Durch diese Einbettung ergibt ein weites Spektrum von Einsatzmöglichkeiten für die Benutzer. Mit Hilfe von WLANs lassen sich schon bestehende drahtgebundene Netzwerke erweitern. Weiterhin ermöglicht die Schnittstelle eine schnelle Übertragung von Multimediadateien.

IEEE 802.11-Standard

Der IEEE-802.11-Standard wurde Ende 1997 verabschiedet und definiert eine Kommunikation in WLAN Netzen. Herausgeber ist das Institute of Electrical and Electronics Engineers. Dieser Standard gehört zu der Gruppe der 802.x-Standards für lokale Netze, in dem auch der Ethernet-Standard 802.3 festgelegt ist. Weiterhin spezifiziert dieser Standard die Bitübertragungsschicht und die Medienzugriffsverfahren, welche speziell für die drahtlose Übertragung notwendig sind. Für den Standard wurde das lizenzfreie ISM-Band (Industry, Science and Medical) bei 2,4 GHz gewählt, welches weltweit verfügbar ist. Die Übertragungsrate liegt bei dem 802.11-Standard bei 2Mbit/s.

Schon bei der Einführung des Standards war abzusehen, dass die Datenrate von 2Mbit/s nicht ausreicht, also wurde diese weiterentwickelt. Alle neuen Varianten des Standards werden mit Buchstaben gekennzeichnet, die an die Standardnummer angehängt werden. Auf unserer Zielumgebung, dem Internet Table N810 von Nokia ist der WLAN-Standard

IEEE 802.11g vorzufinden. Ebenfalls ist dieser Wireless LAN Standard von 2002/2003 im 2,4 GHz Bereich angesiedelt und ist zusätzlich abwärtskompatibel mit dem älteren IEEE 802.11b. Die Geschwindigkeit ist auf maximale 54 MBit/s beschränkt, die Sendeleistung und somit auch die Reichweite betragen bis zu 100m.

Weiterhin unterstützt der IEEE-802.11-Standard Infrastruktur- und Ad-hoc-Netzwerke. Bei einer Infrastrukturarchitektur werden mehrere Endgeräte drahtlos mit einer zentralen Verwaltung, dem Access Point verbunden. Dabei enthält jedes Endgerät alle Mechanismen für den Medienzugriff und eine Komponente für die drahtlose Verbindung mit dem Access Point. Bei einem Ad-hoc-Netz ist man dagegen nicht auf einen Access Point angewiesen. Aus dieser Eigenschaft folgt, dass Ad-hoc-Netze spontan gebildet werden können.

Um auch den Anforderungen der Datensicherheit gerecht zu werden, wurden im IEEE-802.11-Standard Mechanismen dafür integriert. Damit die Funkverbindung weder gestört noch abgehört werden kann, wird mit dem Bandspreizverfahren das Signal über ein möglichst breites Frequenzspektrum aufgeteilt. Jedem Funknetz kann ein eigener Netzwerkname, die so genannte SSID (Service Set Identity) zugewiesen werden. Somit können nur solche Clients am Netzwerk teilnehmen, die den Netzwerknamen (SSID) kennen. Weiterhin besteht auch die Möglichkeit, die Nutzdaten mittels des RC4-Algorithmus zu verschlüsseln. Und als letztes kann eine Liste von MAC-Adressen definiert werden. Diese Liste besteht aus den MAC-Adressen der Clients, die sich mit dem Netzwerk verbinden dürfen.

A.12 Konfigurationsfreie Vernetzung - Zeroconf, Bonjour, mDNS (Lutz)

Einleitung

Ein Ziel der Projektgruppe ist es, unabhängig von einem festen Internetzugangspunkt ein autarkes Netz zwischen einzelnen Netzwerkteilnehmern einrichten zu können. Hier auftretende Probleme sind unter anderem die fehlende Verfügbarkeit einer Kontrollinstanz, die für die Konsistenz des Netzes sorgen kann. Für eine solche Netzwerktopologie existiert bereits ein Standard, Zeroconf, und es existiert auch eine Implementierung für linuxbasierende Systeme, Avahi.

Zeroconf basiert auf einem von Apple in den 1980er Jahren eingeführten System namens AppleTalk, welches im weiteren Verlauf auf eine Ethernet- und später zusätzlich eine WLAN-Basis gestellt wurde. 1999 wurde eine Arbeitsgruppe zur Harmonisierung des Standards auf verschiedenen Plattformen, die „Zeroconf Working Group“, gegründet. Diese hat eine Reihe von Standards ausgearbeitet, die heute die Grundlagen für die verschiedenen Zeroconf-Implementierungen auf den verschiedenen verfügbaren Systemen bilden.

Anforderungen

Eine Zeroconf-Implementierung muss folgende Punkte erfüllen:

- Adresszuweisungen ohne DHCP-Server
- Namensübersetzungen ohne dedizierten DNS-Server
- Finden spezieller Netzwerkdienste ohne Directory-Server
- Zuweisung von Multicast-Adressen

Im Einzelnen sind für unseren Anwendungsfall lediglich die ersten drei Punkte interessant. Insbesondere muss auch ein Konfliktmanagement bei Adress- bzw. Namensüberschneidungen gewährleistet sein.

Adress- und Namenszuweisungen sowie Dienstedeklaration

Eine Zeroconf-Implementierung weist innerhalb des Netzes Adressen aus dem IP-Adressraum 169.254.1.0 bis 169.254.254.255 zu. Da eine zentrale Kontrollinstanz fehlt, muss die Adresszuweisung auf dem Client stattfinden, der einem Netz beitritt. Hierzu wird i.d.R. auf Basis der MAC-Adresse der Netzwerkschnittstelle eine zufällige IP-Adresse erstellt, die dann im Netz auf Verfügbarkeit überprüft wird. Im Falle einer Überschneidung wird so lange eine neue Adresse getestet, bis eine freie gefunden wird.

Daraufhin wird ein Netzwerkname für den Client bestimmt. Die Auswahl eines Namens verläuft vom Prinzip her ähnlich, wie die Adresszuweisung. Als Ausgangsbasis dient der bereits gegebene Gerätename, dem im Regelfall nur das Suffix „.local“ angehängt wird. Ist aus irgendwelchen Gründen keiner vorhanden, ist die Ausgangsbasis implementierungsabhängig.

Jeder mit dem Netz verbundene Client kann Dienste deklarieren, die von anderen Netzteilnehmern in Anspruch genommen werden können. Hierzu gibt es eine Spezifikation, die Einträge aus dem Standard-DNS verwendet.

Namensauflösung

Die Namensauflösung läuft clientseitig analog zum Verfahren in serverbasierten Netzwerken. Das Fehlen eines DNS-Servers wird dadurch substituiert, dass alle Anfragen über die Multicast-Adresse 224.0.0.251:5353 gesendet werden. Alle Clients, die mit dem Netz verbunden sind, erhalten diese Anfragen und beantworten sie nur dann, wenn sie selbst das Ergebnis der Anfrage darstellen. Die Antworten werden ebenfalls per Multicast im Netzwerk verteilt, damit die übrigen Netzteilnehmer diese in eigene Caches übernehmen können. Programme, die von sich aus nicht zeroconf-fähig sind, senden ihre Anfragen

an den Standard-DNS-Port 53. Diese Anfragen werden allerdings mit einem Unicast-Paket beantwortet und haben eine deutlich reduzierte Time-To-Live, da das Programm wahrscheinlich keine plötzlichen Änderungen in der Netzwerktopologie erwartet.

Diensteauflösung

Das Mittel der bereits vorhandenen Service-Einträge im Standard-DNS wird dahingehend erweitert, dass für die konkrete Auswahl des gewünschten Dienstes Zeiger-Einträge verwendet werden. Dieser Umweg wird verwendet, um den Gewichtungen und Prioritäten in den Service-Einträgen des Standard-DNS zu entgehen. Dadurch ist hier auch eine benutzerseitige Auswahl der konkreten Dienste-Instanz möglich. Die Zeigereinträge weisen schließlich auf einen einzelnen Service-Eintrag, der dann ohne Einschränkungen verwendet werden kann. Die Dienst-Einträge enthalten neben der anzusprechenden Adresse auch noch weitere Informationen über eventuelle Konfigurationen des Dienstes, bspw. verfügbare Dateien.

Fazit

Die Verwendung von Zeroconf im Rahmen der Projektgruppe scheint sehr aussichtsreich, da zum Einen alle Anforderungen des Projektes, darunter eine hohe Flexibilität in Bezug auf Änderungen der Topologie, das Fehlen einer zentralen Kontrollinstanz und die Möglichkeit einer Abfrage verfügbarer Dienste, hier unser Projekt, erfüllt werden. Zum anderen besteht seit langem eine stabile Implementierung auf Linux-Basis, die sich ohne nennenswerten Aufwand auf unseren Geräten nutzen und in unser Projekt einbinden lässt.

B N810

Dieses Kapitel behandelt die unserer Projektgruppe zugrundeliegende Hardwareplattform. Da das gesamte Projekt UbiMuC am Ende auf dieser lauffähig sein soll, sollte bekannt sein, welche Hardware denn nun genau zur Verfügung steht. Im ersten Abschnitt wird zuerst kurz der Sinn und Zweck der Architektur dargestellt. Im folgenden zweiten Abschnitt wird auf die Hardware selbst näher eingegangen, wobei an ausgewählten Stellen auch der Anschaulichkeit halber Vergleiche zu anderen bekannten Produkten, wie zum Beispiel Desktop-Computern, gezogen werden. Zum Abschluss wird noch betrachtet, wie sich die Batterielaufzeit des N810 in realen Anwendungsfällen verhält, um so eine Idee zu vermitteln, welche Laufzeiten in der Praxis zu erwarten sind.

B.1 Allgemeines zur Architektur (Nils, Stefan, Jens)

Das N810 wird von Nokia als „Internet-Tablet“ vertrieben. Es soll die allgegenwärtige Internetnutzung ermöglichen. Bedingt durch den mobilen Ansatz müssen entsprechend den Erfordernissen auch die Hardwareeigenschaften angepasst sein. Das bedeutet, dass eine hohe Akkulaufzeit und möglichst vielfältige Verbindungsmöglichkeiten vorliegen müssen. Des weiteren muss das Gerät hinreichend klein sein, so dass es möglich ist, es immer bei sich zu tragen, vergleichbar mit einem Handy oder Smartphone. Nokia grenzt das Gerät dabei bewusst durch Weglassen des GSM/UMTS Senders/Empfängers von den Mobiltelefonen ab. Man kann mit dem N810 nicht auf klassische Art und Weise Telefonieren, sondern nur über Voice over IP.

B.2 Die Hardware und deren Features (Nils, Stefan, Jens)

B.2.1 Der OMAP2420

Das N810, gefertigt von Nokia, baut auf einen OMAP2420 Chip der Firma Texas Instruments auf. Dieser Chip stellt sämtliche Grundfunktionen, wie die ARM11 basierende CPU, den 2D/3D Beschleuniger, den digitalen Signalprozessor (hierzu mehr im nächsten Abschnitt), sowie den „Imaging Video Accelerator“ zur Verfügung. Die ARM11 CPU ist mit 400MHz getaktet und hat Zugriff auf 128MB „mobile DDR memory"[19].

B.2.2 Der DSP des N810

Als digitalen Signalprozessor verwendet Nokia beim N810 einen TMS320C55x von Texas Instruments. Dieser DSP besitzt eine Taktrate von 220 MHz und verbraucht laut Angaben von TI lediglich 0,05 mW pro MIPS bei 0,9 V Betriebsspannung. Auf der technischen Seite bietet er zwei ALUs (Arithmetic Logical Units) von 40 und 16 Bit, die beide parallel arbeiten können. Zur Entlastung des Transfers zum Hauptspeicher wurden zusätzlich vier

40-Bit Akkumulationseinheiten verbaut[20]. Um den DSP nutzen zu können, sind bereits Kernel-Treiber verfügbar, welche die Funktionen per „Gateway“ weiteren Anwendungen verfügbar machen. Auf diese Weise können einige Audio- und Video-Berechnungen an der CPU vorbeilaufen und vom DSP durchgeführt werden. Auf der Maemo-Website [21] ist ein Schaubild zu finden, welches die Architektur und den Aufbau gut visualisiert und im Folgenden beschrieben ist. Aufgrund Unsicherheiten bezüglich der Rechtslage wurde auf eine direkte Darstellung verzichtet.

Die beiden zentralen Bibliotheken ALSA library und libesd gehören dabei zu den Standard-Paketen für Linuxsysteme und sind weit verbreitet. Beide setzen auf einem eigens für das N810 geschaffenen DSP gateway driver auf, der seine Aufgaben an den dedizierten DSP übermittelt. Bei der Grafikdarstellung verhält es sich so ähnlich, nur dass hier der Framebuffer das Glied zwischen dem Xserver und der eigentlichen Hardware ist. Für Eigenentwicklungen bezüglich der DSP-Nutzung stellt TI einige Development-Kits bereit, unter anderem auch das Code Composer Studio (CCS). Die Software kann bei TI heruntergeladen werden, sobald man sich dort einen Account erstellt hat. Mithilfe dieser Umgebung ist es nun möglich, eigenen C-Code auf dem TMS320C55x auszuführen. Der geschriebene Code kann allerdings nicht direkt gestartet werden, sondern muss über mehrere Schritte an die Architektur der ARM-11/ TMS320C55x-Kombination angepasst werden. Bei Texas Instruments [22] ist auch eine Programmier-Anleitung zu finden, die weitere Informationen und Optimierungshinweise enthält.

B.2.3 Kommunikationsmöglichkeiten

Die Verbindung mit der Außenwelt kann man mithilfe des integrierten WLANs herstellen. Es wurde ein WLAN-Chip verbaut, der dem IEEE 802.11 b/g Standard folgt, welcher Datenraten von bis zu 54MBit/s ermöglichen soll. Außerdem ist Bluetooth 2.0 mit EDR (Enhanced Data Rate) verbaut. Daher kann man zum Beispiel ein Mobiltelefon als Modem nutzen oder auch eine externe Tastatur als Eingabemedium verwenden.

Eine in Desktop-Systemen gebräuchliche RJ45-Schnittstelle ist nicht vorhanden, diese würde aber auch nicht zur Ausrichtung des Gerätes passen, die auf die mobile und nicht kabelgebundene Nutzung ausgelegt ist. Das verwendete WLAN sowie Bluetooth entspricht auch den in Desktop-Systemen sowie Notebooks verwendeten kabellosen Netzwerkschnittstellen und braucht hier den Vergleich nicht zu scheuen.

B.2.4 Eingabegeräte

Das N810 verfügt über eine beleuchtete Tastatur, welche per Schiebemechanismus im geschlossenen Zustand unterhalb des Bildschirms verborgen ist. Sie verfügt nur über 46 Tasten, was im Vergleich zu den zumindest 105 Tasten der gebräuchlichen PC-Tastaturen eine sehr geringe Anzahl darstellt. Entsprechend müssen einige Kompromisse im Tastaturlayout eingegangen werden. Abgesehen von den 26 Tasten für die Buchstaben „a“ bis

„z“ müssen im deutschen Layout auch noch die Tasten „ä“, „ö“ und „ü“ untergebracht werden. Damit sind schon 29 der 46 Tasten belegt. Dazu kommen nun noch Shift (doppelt vorhanden), Enter (doppelt vorhanden), Backspace, die Leertaste, Steuerung, eine Funktionstaste, sowie die 4 Cursortasten. Des weiteren gibt es eine Taste zum Öffnen des Menüs, eine Taste zum Einblenden einer Auswahl von ansonsten nicht eingebbaren Sonderzeichen per Bildschirmtastatur, sowie die Zeichen „.“, „,“ und „-“ direkt verfügbar. Sämtliche anderen Zeichen müssen über die Funktionstaste in Kombination mit den entsprechend mehrfach belegten Buchstabentasten eingegeben werden. So sind zum Beispiel die Zahlen von 0 bis 9 auf die oberste Zeile der Buchstabentasten gelegt. Insgesamt ist das Tastaturlayout an das normale Layout der bei Desktop-Computern verwendeten Tastaturen angelehnt, was sich in der Buchstabenanordnung widerspiegelt. Insgesamt sind die Tasten der Tastatur recht klein und auf eine Bedienung mit dem Daumen ausgelegt. Entsprechend ist die Tastatur kaum geeignet um dort stundenlang mit anderen Nutzern in diversen Netzwerken zu chatten. Allerdings ist es vollkommen ausreichend um kurz wenige Zeilen zu tippen.

Abgesehen von der normalen Tastatur kann man aber auch den Touchscreen zur Texteingabe nutzen. So wird, wenn man die entsprechende Option aktiviert hat, automatisch eine Bildschirmtastatur eingeblendet, auf der man die entsprechenden Zeichen für die Eingabe direkt auswählen kann. Außerdem kann man diese auch auf Handschrifterkennung umschalten, in der man dann Druckbuchstaben schreiben kann, welche vom System erkannt und übernommen werden. Die Erkennung der einzelnen Zeichen ist dabei recht gut.

Das Gerät verfügt an der oberen Kante des Gehäuses noch über weitere Tasten, mit welchen man das angezeigte Programm auf den Vollbildmodus umschalten kann, oder auch andere plattformabhängige Optionen, wie hinein- und hinauszoomen oder lauter und leiser steuern kann. Eine der Tasten dient dabei als Aus-/Ein-Schalter und eine andere dafür, den Touchscreen zu blockieren, so dass es nicht möglich ist, aus Versehen Eingaben über selbigen durchzuführen, wenn man das Gerät zum Beispiel in einer Tasche mit sich trägt. An der Vorderseite sind neben dem Bildschirm noch zwei weitere Tasten angebracht. Eine dient dazu, die aktuelle Anwendung zu wechseln, die andere als Escape-Taste.

Abgesehen von den zuvor genannten Eingabegeräten verfügt das N810 noch über eine VGA Webcam, welche Bilder und Videos mit bis zu 640x480 Pixeln aufnehmen kann und auch für Videotelefonie nutzbar ist. Um das Gerät für die Telefonie nutzen zu können, ist ein Mikrofon integriert.

B.2.5 Stromversorgung

Die Stromversorgung wird durch den verwendeten Lithium-Polymer-Akku sichergestellt. Dieser bietet eine Nennspannung von 3,7V und hat eine Kapazität von 1500mAh. Das gleiche Modell wird auch im Nokia E61i und dem E90 Communicator eingesetzt [23]. Laut

Datenblatt soll der Akku bei kontinuierlicher Nutzung, sprich bei eingeschaltetem Display und WLAN, Energie für bis zu 4 Stunden Betrieb liefern. Mehr zum Themenbereich Akkulaufzeit in Kapitel B.3. Der Akku wird über ein im Handheld integriertes Ladegerät wieder aufgefüllt, für das man ein Netzteil anschließen muss, welches 5V Gleichspannung bei einer Stromstärke von 890mA liefert. Damit dauert es ungefähr drei Stunden, den integrierten Akku wieder komplett zu laden.

Die Akkulaufzeit ist durchaus akzeptabel, wenn man sie mit günstigen Notebooks vergleicht. Grund hierfür ist, dass der verwendete OMAP-Chipsatz recht sparsam zu Werke geht. Allerdings ist die Akkulaufzeit im Vergleich zu Handheld Spielekonsolen wie dem Nintendo DS Lite, der laut Angaben von Nintendo bis zu 19 Stunden mit einer Akkuladung durchhält [24], doch recht gering. Die Akkulaufzeit des N810 ist für das gebotene durchaus akzeptabel, auch wenn man sich als Nutzer sicherlich mehr wünschen würde.

B.2.6 Datenspeicher

Es sind 256MB Festspeicher verbaut, auf dem das Betriebssystem sowie weitere grundsätzliche Daten und Programme gespeichert sind. Abgesehen davon sind im Gerät weitere 2GB für Nutzerdaten vorhanden. Von diesen 2GB sind von vorne herein 400MB für die Straßenkarten enthaltener Software vergeben. Ansonsten sind die restlichen 1,4 GB zur freien Verwendung durch den Anwender gedacht, um dort zum Beispiel Videos, Audiodateien oder auch gänzlich anderes Material abzulegen. Dabei ist zu beachten, dass der Speicher fest im Gerät verlötet ist. Der Speicher lässt sich über einen integrierten mini-SD-Karten Einschub noch erweitern. Mit diesem kann man laut der technischen Spezifikation [1] von Nokia bis zu 8GB zusätzlichen Speicher ansteuern, wobei die Karten nach dem SDHC Standard gefertigt sein müssen, wenn mehr als 2GB auf der Karte Platz finden sollen. Neben miniSD Karten kann man außerdem mithilfe eines nicht mitgelieferten Adapters microSD Karten verwenden, um so mehr Speicher zur Verfügung zu haben. Sämtliche SD Karten bauen auf NAND-Speicher auf und weisen eine entsprechend limitierte Anzahl von 100.000 bis 1.000.000 Schreibzyklen auf [25]. Allerdings ist dies aufgrund des geringen Anschaffungspreises bei den Speicherkarten kein größeres Problem, da man eine defekte Karte mit nur recht geringen Unkosten durch eine neue Karte ersetzen kann, was bei dem intern verwendeten Speicher bei einem Defekt nicht möglich ist.

Bei Desktop-Systemen hat man für gewöhnlich mehrere hundert Gigabyte Speicherkapazität. Für solch ein kleines Handheldgerät ist der vorhandene Speicher jedoch vollkommen ausreichend, da man selbigen ja über Speicherkarten leicht erweitern kann. Auch Videos, wohl jenes Medium, welches den höchsten Platzbedarf auf der Hardware aufweist, kann man aufgrund des kleinen Bildschirmes in einer recht geringen Auflösung encodieren. Der reale Platzbedarf kann so derart gering gehalten werden, dass man auf einer günstigen 2GB Speicherkarte mehrere Stunden Filmmaterial ablegen kann.

B.3 Betrachtung der Batterielaufzeit des N810 in verschiedenen Szenarien (Nils, Stefan, Jens)

Um eine erste Abschätzung zu erhalten, welche Ressourcen für den Betrieb der jeweiligen Teilaspekte der Endanwendung (Streaming, Videodarstellung, Netzwerklast) nötig sind, wurden mehrere unterschiedliche Laufzeitanalysen durchgeführt.

Das N810 selbst bietet einige Optionen, um den Energieverbrauch einzuschränken. Unter anderem kann die Display-Helligkeit verändert werden, ebenso wie Standby-Zeiten und automatische Dimmung nutzbar sind. Um ein möglichst aussagekräftiges Worst-Case-Szenario (Dauernutzung) zu erstellen, wurde zur Messung des Batterieladestands das Tool „battery-status“ [26] installiert. Werksseitig sind auf dem N810 leider keine Möglichkeiten gegeben, den Ladestand auszulesen. Zur Simulation der Dauernutzung des Geräts wurde die Display-Helligkeit auf die höchste Stufe gesetzt und sämtliche Standby- bzw. Display-Abschalt-Zeiten auf den Maximalwert von 1440 Minuten hochgestuft. Dies geschah über das Tool „moreDimmingoptions“ [27], welches deutlich höhere Abschalt- und Abdunkelungszeiten erlaubt als die Standard-Werte. Auf diese Weise bleibt das N810 eingeschaltet und simuliert durch den Verbrauch des Displays eine Art „Dauernutzung“. Mit Hilfe des erwähnten battery-status-Programms wurden die jeweiligen Ladestände der Batterie in 10-minütigen Intervallen in eine Log-Datei gespeichert. Anhand der Log-Länge ist am Ende des Testlaufs ersichtlich, wie lange das Gerät eingeschaltet bzw. wie hoch der jeweilige Verbrauch war.

Als Referenzwert wurde das N810 bei aktivierter WLAN-Verbindung, eingeschaltetem Sound und neutralem Displayhintergrund betrachtet, wobei eine Gesamtlautzeit von rund 8,3 Stunden erreicht wurde, bei ungefähr 2% Batterieverbrauch pro 10 Minuten Laufzeit. Neben battery-status liefen nur die vom Betriebssystem automatisch gestarteten Prozesse. Die erreichte Laufzeit hängt natürlich wesentlich von der Güte des Akkus ab, so dass gerade bei älteren Geräten mit einem höheren Benutzungsgrad aufgrund von Verschleißerscheinungen niedrigere Laufzeiten zu erwarten sind.

Nachdem nun eine Obergrenze für die Laufzeit feststand, musste als nächstes überprüft werden, inwiefern die Nutzung der drahtlosen Netzwerkschnittstelle zusammen mit einem P2P-Framework die Gesamtlaufzeit beeinflusst. Hierfür wurde GNUnet verwendet, jeweils in unterschiedlichen Konfigurationen. Da eine Parametrisierung im Hinblick auf den Grad der CPU- und Netzwerk-Nutzung möglich ist, wurden beide Werte in verschiedenen Testläufen variiert. Es stellte sich dabei heraus, dass die freigegebene Durchsatzrate für das Netzwerk die Laufzeit weitaus mehr beeinflusst als die angegebene Schranke für die CPU-Nutzung:

Ein gesonderter Testlauf mit ausgeschaltetem Display zeigte dann erneut die offensichtliche Abhängigkeit zwischen Stromverbrauch und Display-Betrieb. Insgesamt konnte durch die Tests auf Basis des GNUnet-Frameworks jedoch gezeigt werden, dass hinsichtlich CPU-Nutzung und Beanspruchung der Netzwerkschnittstelle keine zu großen Engpässe bei der Laufzeit des N810 zu erwarten sind. Das Display ist einer der größten Verbraucher

max. CPU-Last	max. Netzwerk-Last	Laufzeit	
100%	50.000 Byte	6 Stunden	
50%	300.000 Byte	5 Stunden	
15%	250.000 Byte	5,5 Stunden	
50%	1024.000 Byte	3,3 Stunden	
100%	50.000 Byte	14 Stunden	(Display aus)

Tabelle 3: Eckdaten der Messreihen und Ergebnisse der Laufzeitanalyse

und spielt damit eine zentrale Rolle bei der Betrachtung der Akkulaufzeit. Die gewährte CPU-Last von GNUnet ist beim Verbrauch eher nachrangig, während die erlaubte Belastung der drahtlosen Schnittstelle hier deutlich stärker zum Stromverbrauch beiträgt.

Neben diesen Framework-Tests wurde ebenso der Verbrauch beim Betrieb eines Streaming-Servers bzw. Nutzung eines Streaming-Angebots ausgelotet. Mit Hilfe des VLC-Players wurde das N810 zum einen als Ziel (Client) eines Multimedia-Streams genutzt, ebenso wie es als Quelle fungierte. Auf diese Weise wurden beide Seiten eines Ad-Hoc-Angebots von Multimediainhalten beleuchtet.

Bei der Nutzung als Client ohne weitere Programme im Hintergrund zeigte sich eine Laufzeit von etwas mehr als 2,7 Stunden. Um hier noch einen weiteren Verbraucher hinzuzufügen, wurde GNUnet im Hintergrund gestartet und parallel zum Stream laufen gelassen. Es konnten dabei immer noch Werte um 2,5 Stunden erreicht werden.

Wird das N810 als Server für ein Streaming-Angebot benutzt, so war ebenfalls eine durchschnittlichen Laufzeit von etwa 2,5 Stunden erreichbar, wobei GNUnet erneut im Hintergrund aktiv war.

Fazit

Als Fazit stellte sich heraus, dass selbst bei Nutzung eines Streaming-Dienstes zusammen mit einem P2P-Framework im Hintergrund, ein Spielfilm einer durchschnittlichen Länge von 1,5 Stunden ohne Probleme über das Netzwerk per Stream empfangen werden kann. Angesichts dieser Messreihen kann davon ausgegangen werden, dass die anvisierte Endanwendung (als Kombination aus P2P-Framework und Streaming-Dienst) auf dem N810 ohne offensichtliche Batterieprobleme eingesetzt werden kann.

C Unsere Arbeitsumgebung

In diesem Kapitel wird darauf eingegangen, wie wir die Arbeit für die Projektgruppe innerhalb der Universität organisieren. Insbesondere geht es dabei um die technische Unterstützung und die verwendeten Medien.

Zuerst wird auf den uns bereitgestellten Rechnerpool eingegangen und die Anpassungen, die wir an der vorgefundenen Software vornehmen mussten. Im nächsten Abschnitt werden die Werkzeuge vorgestellt, mit denen wir unsere Arbeitsergebnisse dokumentieren und innerhalb der Gruppe zugänglich machen. Im letzten Abschnitt wird schließlich das von uns zum Erstellen unserer Software verwendete SDK angesprochen.

C.1 Die Arbeitsumgebung im CI-Lab (Markus S.)

Arbeitsumgebung

Für unsere PG wurden im CI-Lab Software und Accounts bereitgestellt, um mit dem von Nokia bereitgestellten SDK, das die Entwicklung von Anwendungen für das Nokia 810 sehr erleichtert, innerhalb der Universität arbeiten zu können. Im CI-Lab stehen Windows-Rechner zur Verfügung, auf denen ein VMWare-Player installiert ist und ein von Maemo angebotenes VMWare-Image [32] abgelegt wurde, das eine Linux-Umgebebung inkl. vorinstallierter Scratchbox, eine Eclipse-Programmierumgebung und die Standard-Software einer Xubuntu-Distribution bereitstellt. Die Eclipse-Umgebung ist bereits so vorkonfiguriert, dass die Scratchbox direkt auf der GUI aufgerufen werden kann und die Erstellung von Maemo-Programmen möglich ist. Jeder Rechner verfügt über ein eigenständiges, lokal gespeichertes Image, so dass unabhängig voneinander an den Rechnern gearbeitet werden kann. Individuelle Anpassungen für die Linux-Umgebung, wie die Änderungen von Tastatur-Layouts o.ä. waren leicht umzusetzen, da jedem Nutzer innerhalb des Images Root-Rechte zugänglich sind. Jedoch ergaben sich anfangs Probleme mit dem Internet-Zugang, USB-Support und dem Austausch von Daten zwischen Host-Rechner und dem Image.

Anpassungen für Internet-Zugang und USB-Support

Da mit den Default-Einstellungen ein Zugriff auf das Internet innerhalb des Images nicht möglich war, mussten als Umgebungsvariablen Proxys für HTTP, HTTPs und FTP in '/etc/environment' definiert werden. Überraschenderweise war innerhalb des Images auch die Ansteuerung der USB-Ports im Host-System anfänglich deaktiviert, obwohl die Datenübertragung aus dem Image auf das N810 per USB das Standardverfahren ist. Jedoch ließ sich durch die Änderung weniger Einträge in den Konfigurationsdateien des VMImage der USB-Support aktivieren. Dadurch ist der direkte, exklusive Zugriff auf USB-Speichermedien aus Windows oder dem VMWare-Image heraus umschaltbar. Direkten

Zugriff auf das N810, wie es z.B. der Flasher benötigt, war auf diesem Weg im CI-Lab allerdings weiterhin nicht möglich. Diese Einschränkung ist vermutlich durch Restriktionen durch das Host-System begründet und vermutlich nicht umgehbar. Der direkte Datenaustausch zwischen Host-System und Image ist von der VMWare in Form von Shared Folders angedacht. Da diese innerhalb des Images allerdings nicht konfiguriert sind und die Konfiguration auch ohne zusätzliche kostenpflichte VMWare-Software nicht möglich ist, wurde die Option, Daten zwischen diesen Systemen auszutauschen, als unbedeutend eingeschätzt und nicht weiter daran gearbeitet, diese zu ermöglichen.

C.2 Verwendete Werkzeuge zur Koordinierung von Arbeitsergebnissen (Markus S.)

UbiMuC-Wiki

Für die Dokumentation unserer Arbeitsergebnisse und die Sammlung allgemein nützlicher Informationen haben wir uns auf die Nutzung eines Wiki geeinigt, das über *https://ls12-www.cs.tu-dortmund.de/wiki/ubimuc/* erreichbar, aber nur für Beteiligte per Login zugreifbar ist. Auf dieser Plattform werden neben Arbeitsergebnissen auch Anleitungen zu vielen Themen gesammelt, die wöchentlichen Sitzungsprotokolle veröffentlicht, die wöchentlichen To-Do's gelistet, sowie deren Erledigung dokumentiert und mit entsprechenden Ergebnissen direkt verlinkt. Weiterhin haben wir zur schnellen Kommunikation noch eine Mailing-Liste für alle Teilnehmer parallel laufen.

UbiMuC-Repository

Um unsere N810s und unsere Programmierumgebungen konsistent halten zu können, wenn größere Programmteile fertiggestellt werden, legen wir wichtige Debian-Pakete in einem für Maemo-Anwendungen gedachtem Repository ab, welches unter *http://ls12-www.cs.tu-dortmund.de/ubimuc/repository/* verfügbar ist. Dies wurde in den Paketmanager all unserer Geräte eingefügt und enthält zudem auch noch nützliche zusätzliche Programme und Tools, die aus dem offiziellen Repository nicht bezogen werden konnten oder von uns angepasst werden mussten.

SVN-Repository

Zur Versionierung unserer Entwicklung am Quellcode sowie der Berichte benutzen wir Subversion. Das dazugehörige SVN-Repository liegt ebenfalls auf dem Server des Lehrstuhls. Damit ist auch die Konsistenz unserer Programme beim Entwickeln und ein einfaches, verteiltes Arbeiten am Gesamtprojekt sichergestellt.

C.3 Das Software Development Kit Scratchbox (Markus S.)

Zur Entwicklung der Software für das N810 benutzen wir das Maemo 4.0 SDK, dessen wichtigster Bestandteil die Scratchbox darstellt. Diese ist eine unter Linux aufrufbare Umgebung, die ein Linux-Betriebssystem mit dem Funktionsumfang des im N810 bereitgestellten Betriebssystems innerhalb der normalen Systemumgebung emuliert. Dazu gehört ebenfalls die Möglichkeit, das Verhalten der GUI des Betriebsystems im N810 zu simulieren.

Die Einrichtung des SDK ist auf jedem Linux-Rechner nach offiziellen Anleitungen leicht möglich [33]. Eine Alternative ist das VMWare-Image [32], das im CI-Lab verfügbar ist. Hier ist die Scratchbox bereits vorinstalliert und in die Programmierumgebung Eclipse mit zusätzlichen Menü-Einträgen integriert und direkt aufrufbar. Bisherige Entwicklungen in der Scratchbox haben jedoch gezeigt, dass deren Verhalten nicht immer mit dem Originalsystem übereinstimmt. So werden Fehler teilweise viel restriktiver behandelt als im realen System.

Literatur

[1] Funktionsbeschreibung des N810 auf der deutschen Nokia-Homepage:
http://www.nokia.de/A4630299

[2] Peter Mahlmann, Christian Schindelhauer
http://wwwcs.upb.de/cs/ag-madh/WWW/Teaching/2004SS/AlgoP2P/Skript/skript-04-4.pdf

[3] Björn Schießle
http://www.schiessle.org/data/can_report.pdf

[4] Werner Gaulke
http://www.gaulke.net/werner/arbeiten/Ausarbeitung_Seminar_P2P_Chord.pdf

[5] Sun Microsystems:
http://www.sun.com

[6] JXTA Homepage:
https://jxta.dev.java.net

[7] JXTA-C Projektseite:
https://jxta-c.dev.java.net

[8] Apple Darwin Streaming Server Homepage:
http://developer.apple.com/opensource/server/streaming/index.html

[9] LScube Homepage:
http://live.polito.it

[10] VideoLAN Homepage:
http://www.videolan.org

[11] MPlayer Homepage:
http://www.mplayerhq.hu

[12] DES:
http://www.itl.nist.gov/fipspubs/fip46-2.htm

[13] AES:
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[14] RSA:
http://mathcircle.berkeley.edu/BMC3/rsa/node4.html

[15] DSA:
http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

[16] RFC 2246: The TLS-Protocol Version 1.0:
www.ietf.org/rfc/rfc2246.txt

[17] Network Security with OpenSSL:
John Viega, Matt Messier und Pravir Chandra, O'Reilly, ISBN 0-596-00270-X

[18] Produktseite des OMAP2420 bei Texas Instruments:
http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?contentId=4671
&navigationId=11990&templateId=6123

[19] Die OMAP2 Architektur:
http://focus.ti.com/pdfs/wtbu/TI_omap2420.pdf

[20] Der DSP TMS320C55x:
http://focus.ti.com/lit/ug/spru393/spru393.pdf

[21] Die Multimedia Architektur von „Maemo“:
http://maemo.org/development/documentation/how-tos/3-x/multimedia_architecture.html

[22] Programmers Guide für den DSP TMS320C55x:
http://focus.ti.com/lit/ug/spru376a/spru376a.pdf

[23] Kompatibilitätsliste für den Akku BP-4L:
http://www.nokia.de/A4787860

[24] Produktdaten des Nintendo DS:
http://www.nintendo.de/NOE/de_DE/systems/ueber_nintendo_ds_1024.html

[25] FAQ über SD Karten mit Angaben zur Lebenserwartung:
http://web.archive.org/web/20080117082102/http://www.sdcard.org/about/faq/#life

[26] Verschiedene Anwendungen für das N810, u.a. „battery-status“:
http://nitapps.com/ bzw. http://nitapps.com/repository.install

[27] Verschiedene Anwendungen für das N810, u.a. „moreDimmingoptions“:
http://maemo.org/downloads/OS2007/system/

[28] GNUnet Homepage:
http://gnunet.org/

[29] Internetseite des Entropy Projekts (Stand: 30. April 2008):
http://entropy.stop1984.com/

[30] Internetseite des Mute Projekts:
http://mute-net.sourceforge.net/

[31] Generelles Paper zu GNUnet:
http://gnunet.org/download/main.pdf

[32] VMWare-Image eines Xubuntu inklusive Maemo SDK:
http://maemovmware.garage.maemo.org/

[33] Einführung in die Nutzung des Maemo SDK:
http://maemo.org/development/documentation/tutorials/maemo_4-0_tutorial.html

[34] GTK+-Homepage:
http://www.gtk.org/

[35] Hildon-Framework auf Gnome.org:
http://live.gnome.org/Hildon

[36] Basis-Layouts und Bildschirmaufteilung bei Einsatz des Hildon-Toolkits: http://maemo.org/development/documentation/tutorials/maemo_4-0_tutorial.html#user-interface-parts

Hinweis:

Referenzen auf Internetseiten entsprechen - wenn nicht anders angegeben - dem Stand vom 24. September 2008.

Abbildungsverzeichnis

1 UbiMuC-Anwendungsfälle . . . 9
2 Anwendungsfall: Verbindungsaufbau . . . 10
3 Anwendungsfall: Serversuche . . . 11
4 Anwendungsfall: Person geht online . . . 11
5 Anwendungsfall: Person sucht in Datenbank . . . 12
6 Anwendungsfall: Inhaltssuche . . . 12
7 Anwendungsfall: Veröffentlichung von Inhalten . . . 13
8 Anwendungsfall: Weiterleitung von Daten . . . 14
9 Anwendungsfall: AV-Konferenz . . . 15
10 Anwendungsfall: AV-Streaming . . . 16
11 Schema der UbiMuC-Schichten . . . 17
12 Ablauf eines Verbindungsaufbaus mit Datenstreaming . . . 19
13 Aufbau eines serverbasierten P2P-Netzes . . . 22
14 Aufbau eines reinen P2P-Netzes . . . 22
15 Aufbau eines hybriden P2P-Netzes . . . 23
16 Netmerge: Ausgangssituation . . . 27
17 Netmerge: Knoten A sieht Knoten B . . . 27
18 Netmerge: Ad-hoc-Netz1 (Knoten A und B) und Ad-hoc-Netz2 (Knoten C) 28
19 Netmerge: Zwei Ad-hoc-Netze . . . 28
20 Netmerge: P2P Netz . . . 29
21 Metadaten eines PDF Dokuments . . . 38
22 Metadaten einer Audio Datei . . . 38
23 Personensuche in GNUnet: Client tritt Netz bei . . . 62
24 Personensuche in GNUnet: Client sucht nach Personen . . . 63
25 Erste Originalskizze der UbiMuC-GUI . . . 71
26 GTK-Version der UbiMuC-GUI . . . 72
27 Hildon-Version der UbiMuC-GUI . . . 72
28 Tab-Version des Suche-Fensters der UbiMuC-GUI . . . 73

Tabellenverzeichnis

1 Ergebnisse der Testreihe zur Inhaltssuche . . . 39
2 Vergleich JXTA - GNUnet . . . 44
3 Eckdaten der Messreihen und Ergebnisse der Laufzeitanalyse . . . 103