HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

A Practical Javascript-Only Video-Over-IP Communication Platform

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2013

Tobias Blaser, Jannis Grimm
Betreuer: Prof. Dr. Luc Bläser

Aufgabenstellung Studienarbeit für Tobias Blaser und Beat Gutzwiller:

A Practical JavaScript-Only Video-Over-IP Communication Platform

1. Projektrahmen

Diese Studienarbeit findet als internes Projekt des IFS statt.

Betreuer HSR:

- Prof. Dr. Luc Bläser, Institut für Software, lblaeser@hsr.ch

2. Ausgangslage

Mit WebRTC (http://www.webrtc.org/) können Browser direkt mit HTML5 und JavaScript zeitkritische Kommunikationen unterstützten. Ziel dieses Projektes ist es, diese Technologie genauer zu analysieren und darauf aufbauend ein praxistaugliches „Voice & Video Over IP" Plattform zu entwickeln. Diese Plattform soll folgende Features unterstützen:

- Sprach und Video-Kommunikationen zwischen zwei und mehreren Teilnehmer basierend auf WebRTC, HTML5 und JavaScript
- Möglichst einfache Bedienung, hohe Effizienz und State-of-the Art Sicherheit
- Kontaktverwaltung und Verbindungsaufnahme per Server
- UI im Hinblick für mobile Anwendungen
- Optional weitere Features, wie Datenaustausch während Kommunikation, Optimierungen der Kommunikationsqualität etc.

Zum Projekt dazu gehört die Performanceanalyse und Kleinstudie zum Performance-Tuning der entwickelten Lösung.

3. Ziele und Aufgabenstellung

Die Aufgabe dieser Arbeit ist es, dass eine praxistaugliches rein Java-Script-basierte Video-over-IP Plattform WebRTC zu entwerfen und zu entwickeln. Client-seitig wird HTML5 und JavaScript mit WebRTC eingesetzt. Serverseitig kann in Absprache mit dem Betreuer eine geeignete Technologie gewählt werden.

Folgende spezifische Ziele werden vorgegeben:

- Sammlung der Anforderungen für die neue Plattform.
- Kleine Technologiestudie zu WebRTC
- Entwurf und Implementierung der Video-over-IP Plattform mit zugehörigen automatisierten Tests.
- Performance-Auswertung der Lösung mit Tuning/Optimierungen

4. Zur Durchführung

Mit dem HSR-Betreuer finden wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

5. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html?&L=0). Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in 3 Exemplaren abzugeben.

6. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html?&L=0

16.09.13	Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch die Betreuer.
16.12.13	Die Studierenden senden folgende Dokumente der Arbeit per Email zur Prüfung an ihre Betreuer: - Kurzfassung - A0-Poster Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.
20.12.13	Die Studierenden senden die vom Betreuer abgenommene und freigegebene Kurzfassung als **Word-Dokument** an das Studiengangsekretariat (cfurrer(at)hsr.ch).
20.12.13	Abgabe des Berichtes an den Betreuer bis 17.00 Uhr.

7. Beurteilung

Eine erfolgreiche Studienarbeit erhält 8 ECTS-Punkten (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Für die Modulbeschreibung der Studienarbeit siehe https://unterricht.hsr.ch/staticWeb/allModules/19456_M_SAI.html

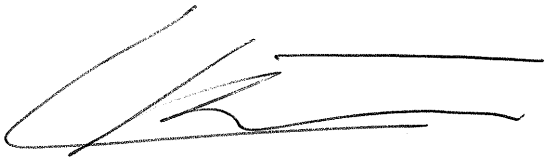
Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt *)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit festgelegt.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, den 13. September 2013

Der verantwortliche Dozent



Prof. Dr. Luc Bläser
Institut für Software
Hochschule für Technik Rapperswil

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Vereinbarung über Urheber- und Nutzungsrechte

Vereinbarung

1. Gegenstand der Vereinbarung

Mit dieser Vereinbarung werden die Rechte über die Verwendung und die Weiterentwicklung der Ergebnisse der Studienarbeit „A Practical JavaScript-Only Video-Over-IP Communication Platform" von Tobias Blaser und Jannis Grimm unter der Betreuung von Prof. Dr. Luc Bläser geregelt.

2. Urheberrecht

Die Urheberrechte stehen der Studentin / dem Student zu.

3. Verwendung

Die Ergebnisse der Arbeit werden unter eine Open Source (z.B. GPL3) gestellt. Die Resultate dürfen deshalb unter dieser Lizenz sowohl von der Studentin / dem Student, von der HSR sowie beliebig weiteren Personen nach Abschluss der Arbeit verwendet und weiter entwickelt werden.

Rapperswil, den 7.10.13 ..... T. Blaser .....
Tobias Blaser

Rapperswil, den 7.10.13 .....
Janis Grimm

Rapperswil, den 7.10.13 .....
Prof. Dr. Luc Bläser

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Erklärung über die eigenständige Arbeit

Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurden,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- das wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:	Name, Unterschrift:
Rapperswil, 17.12.13	J. Blaser
Rapperswil, 17.12.2013	

Änderungsnachweis

Version	Änderung	Autor	Datum
1.0	Dokumentenentwurf	Tobias Blaser	04.11.13
1.1	Hinzufügen zusätzlicher Dokumente	Tobias Blaser	07.11.13
1.2	Verfassen von Vision und Qualitätsmanagement	Tobias Blaser & Jannis Grimm	19.11.13
1.3	Lektorat	Jannis Grimm	22.11.13
1.4	Tobias Blaser	Überarbeitung verschiedener Kapitel	22.11.13
1.5	Doku Review	Jannis Grimm	22.11.13
1.6	Abstract & Management Summary	Tobias Blaser, Jannis Grimm	10.12.13
1.7	Performanceanalyse	Tobias Blaser, Jannis Grimm	12.12.13
1.8	Firewall Testing, Erkenntnisse, Überarbeitungen	Jannis Grimm	14.12.13
1.9	Schnittstellen und Protokolle	Tobias Blaser	15.12.13
1.10	Architektur Überarbeitung	Tobias Blaser	17.12.13
1.11	Rechtschreibkorrektur, Austausch Bilder, Überarbeitung Qualitymanagement	Jannis Grimm, Tobias Blaser	18.12.13
1.12	Lektorat von Marco Syfrig	Jannis Grimm	19.12.13
1.13	Schlussüberarbeitung	Jannis Grimm, Tobias Blaser	19.12.13

Inhaltsverzeichnis

1. Abstract

Ziel der Studienarbeit „A Practical JavaScript-Only Video-Over-IP Communication Plattform“ war die Entwicklung einer JavaScript-Applikation zur Audio- und Videokommunikation, die ausschliesslich im Browser läuft. Benutzer müssen keine Software auf ihrem Endgerät installieren, um die Applikation nutzen zu können. Teil der Arbeit waren die Untersuchungen des aktuellen Standes der Technologien und der notwenigen Schnittstellen, die Umsetzung von Referenzimplementationen der eigenen Schnittstellen, sowie eine Analyse der Performance.

Die Benutzeroberfläche wurde sowohl für Desktop wie Mobilgeräte konzipiert und übersichtlich gestaltet. Unterstützt werden Benutzerkonten sowie Import von Kontaktdaten. Über Schnittstellen kann die Applikation um eigene Kontaktdatenformate und Verbindungsmechanismen erweitert werden.

Die Unterstützung durch die Browser ist noch sehr unterschiedlich. Nur neuere Chrome-, Firefox- und Opera-Versionen unterstützen die Schnittstellen, wobei die Unterstützung experimentell ist und entsprechend variiert. Mediacodierung und Verschlüsselung verbrauchen auch noch spürbar viel Rechenleistung. Die automatische Datenratenskalierung wird bereits durch alle getesteten Browser unterstützt.

Die Technologien hinter den JavaScript-Schnittstellen setzen Protokolle wie STUN oder ICE ein, die von stark restriktiven Firewalls blockiert werden. In Netzwerkumgebungen wie Heimnetzwerken oder kleineren, schwach reglementierten Firmennetzwerken, sowie im Mobilfunknetz stellt dies jedoch keine Probleme dar.

Wir haben es erfolgreich geschafft, eine JavaScript-basierte Videokommunikationsplattform mit dem aktuellen Stand der Browsertechnologie aufzubauen. Alle browserseitig notwendigen Schnittstellen existieren und funktionieren bereits. Trotzdem ist deren Umsetzung noch experimentell, was sich vor allem durch den Ressourcenverbrauch bemerkbar macht.

Mit der erreichten Lösung sind wir sehr zufrieden. Wir haben die gesteckten Ziele sogar übertroffen, indem wir die Videokommunikation um einen Chat erweitert haben.

2. Management Summary

2.1. Ausgangslage

Videokommunikation wird bereits seit Jahrzehnten als Telefonie der Zukunft angeführt, verbreitet sich jedoch erst seit einigen Jahren. Besonders bekannt sind hier die Produkte „Skype“ von Microsoft, „Google Hangout“ sowie „FaceTime“ von Apple. Diese mittlerweile sehr verbreiteten Produkte benötigen alle die Installation einer Software auf jedem Endgerät. Die Produkte verwenden nicht offene Protokolle, sodass nur innerhalb desselben Anbieters kommuniziert werden kann. Bietet der Anbieter keine Applikation für eine bestimmte Plattform, kann diese nicht zur Kommunikation genutzt werden.

Daneben gibt es SIP als offenen Standard, der sich in vielen Unternehmen etabliert hat und für IP-Telefonie genutzt wird. Auch für SIP wird auf jedem Endgerät eine entsprechende Software benötigt. Durch die Offenheit des Standards ist es jedoch möglich, SIP für eigene Plattformen selbst umzusetzen. Telefonie zwischen verschiedenen SIP-Anbietern ist möglich.

Videokommunikation ohne Installation einer Software war bisher nicht möglich. Neue Browser unterstützen nun jedoch entsprechende Technologien.

2.2. Vorgehen

In der Anfangsphase des Projektes lag der Schwerpunkt darauf, die Grenzen der Technologie abzustecken, da die Schnittstellen sich noch in Entwicklung befinden und von den Browsern erst seit einiger Zeit unterstützt werden.

In der anschliessenden Phase ging es darum, mit den Prototypen der ersten Phase eine rudimentäre Applikation umzusetzen, die bereits Videokommunikation und Import von Kontaktdaten erlaubt. Gleichzeitig wurde begonnen einen eigenen SIP-Server zu installieren, um auch mit SIP eine Referenzimplementation umsetzen zu können.

Parallel fanden in der nächsten Phase Performance-Analysen und Ausbauarbeiten an der Applikation statt. Dabei wurden weitere Importformate ergänzt und die Videokommunikation verfeinert. Die Referenzimplementation für SIP wurde abgebrochen, da der verwendete SIP-Server noch keine stabile Implementation der benötigten Schnittstelle besass und eine Einrichtung unmöglich war.

Nach einem grösseren Refactoring wurde in der Abschlussphase die finale Benutzeroberfläche umgesetzt, die auf Desktop- wie Mobilnutzer ausgelegt ist. Zudem wurde die bereits während der zweiten Phase initialisierte und regelmässig aktualisierte Dokumentation finalisiert.

2.3. Ergebnisse

Entwickelt wurde eine Applikation, die vollständig im Browser läuft und somit ohne Installation zusätzlicher Software auskommt. Benutzer können Kontaktdaten importieren und über Video kommunizieren. Die Performance-Analyse hat gezeigt, dass die Technologien nach dem aktuellen Stand noch viel Ressourcen benötigen, grundsätzlich jedoch einsetzbar sind. Die Videokommunikation wird durch technische Hürden wie NATs und gewöhnlichen Firewalls nicht beeinträchtigt. Auch im Mobilfunknetz gibt es keine Einschränkungen. In stark reglementierten Netzwerken werden die Pakete für den Verbindungsaufbau blockiert, sodass ein Verbindungsaufbau nicht möglich ist. Die Umsetzung einer Referenzimplementation für den Verbindungsaufbau über SIP wurde verworfen, weil die SIP-Server die notwendigen Schnittstellen erst seit Kurzem und entsprechend instabil unterstützen.

2.4. Ausblick

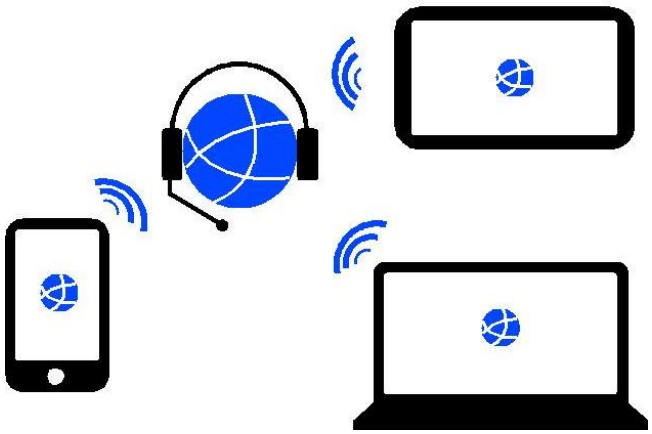
Die Entwicklung der Browser schreitet schnell voran. Es wird erwartet, dass die Schnittstellen bald stabiler werden und die Browser wesentlich weniger Ressourcen für die Videokommunikation benötigen. Ebenfalls wird die Unterstützung weiterer Browser erwartet. Mit Opera ist in den letzen Projektphase noch ein bekannter Browser hinzugekommen, der die Technologie unterstützt.

Ausbaumöglichkeiten für die Applikation wären ein direkter Datenaustausch zwischen den Teilnehmern, Konferenzschaltung, Screensharing und Verbindungsaufbau über SIP.

3. Vision

Im Rahmen der Semesterarbeit „A Practical JavaScript-Only Video-Over-IP Communication Plattform“ wird eine Video-Over-IP-Applikation entwickelt, die vollständig im Browser läuft.

Die Applikation läuft in modernen Browser ohne Plugins oder die Installation lokaler Software.



Durch den offenen Standard „WebRTC“ ist die Applikation auf jedem modernen Gerät benutzbar, welches den Standard umsetzt und genug Leistung für Audio- und Videokommunikation bereitstellt.

Durch die einfache Benutzung ist die Applikation auch für technisch nicht versierte Benutzer zugänglich.

Das integrierte Kontaktmanagement kann mit verschiedenen Importformaten umgehen und einfach um weitere Quellen erweitert werden. Dazu gibt es eine Adressbuchschnittstelle.

Die Applikation kann um beliebige Signalingchannel erweitert werden. Somit ist es unter anderem möglich, die Applikation um Signaling über SIP oder XMPP zu erweitern. Dazu gibt es eine Channelschnittstelle.

3.1. Ziele

Im Projekt sollen die folgenden konkreten Ziele erreicht werden:

- Audio- und Videokommunikation zwischen zwei Teilnehmern.
- Signaling[1] über beliebige Channels, Referenzimplementation eines Channels mit XHR.

[1] Benachrichtigen eines anderen Peers über eine Verbindungsanfrage und Austausch von Verbindungsparametern

- Adressbuchimport von beliebigen Quellen, Referenzimplementationen für JSON- und vCard-Import sowie eines Online-Adressbuches über eine JSON-Schnittstelle.
- Speichern von Adressbüchern und Benutzerinformationen im Browser (Local Storage[8]).
- Verschlüsselte Kommunikation zwischen den Clients. Die Kommunikation der Signaling-Channel-Referenzimplementation muss nicht zwingend verschlüsselt werden, da diese automatisch verschlüsselt wird, sobald die Verbindung mit TLS geschützt ist.
- Unterbricht die Verbindung, so soll der Stream automatisch weiterlaufen, sobald die Verbindung wieder verfügbar ist.

3.2. Abgrenzung

- Kommunikation mit mehreren Teilnehmern ist nicht Teil der Arbeit, es soll jedoch sichergestellt werden, dass dies später umgesetzt werden könnte.
- Für Adressbuchimport und Signaling-Channel sollen nur API und je eine Referenzimplementation umgesetzt werden und keine Implementationen für diverse Formate.

3.3. Optionale Erweiterungen

Optional könnte die Applikation um einige praktische Funktionen erweitert werden:

- Chat
- Austausch von Dateien
- Konferenzschaltung
- Screensharing, soweit die Browser solche Funktionalität überhaupt schon unterstützen

3.4. Mehrwert gegenüber bestehenden VoIP-Lösungen

3.4.1. Mehrwert gegenüber proprietären Diensten wie Skype, Google Hangout, FaceTime

Komplette P2P-Verschlüsselung Sowohl Videokommunikation wie Chatnachrichten laufen verschlüsselt komplett P2P. Der Signaling-Channel-Anbieter hat keinen Zugriff auf die Schlüssel. Entsprechend kann die Kommunikation nicht abgehört werden und auch serverseitig keine Backdoor eingebaut werden.

Plattformunabhängig Skype, Google Hangout und FaceTime sind nur auf Geräten verfügbar, für die die Entwicklerunternehmen eine App anbieten. Unsere App ist auf jeder Plattform verfügbar, die einen genug modernen Browser bietet.

Erweiterbarkeit Skype, Google Hangout und FaceTime lassen sich nicht um Adressbuchtypen oder Signaling-Mechanismen erweitern.

Anbieterunabhängigkeit Skype und FaceTime lassen Kommunikation mit fremden Diensten nicht zu. Durch die Erweiterbarkeit unserer Applikation ist dies gewährleistet.

3.4.2. Mehrwert gegenüber bestehenden WebRTC-Diensten wie SIPML5

Serverdienste SIPML5[2] bietet auch Support für nicht-WebRTC-fähige Geräte und nicht-WebSocket-fähige SIP-Server. Ensprechend ist für SIPML5 eine Serverkomponente notwendig, die einen SIP-Proxy beinhaltet und die MediaStreams konvertiert. Unsere Applikation bietet zwar weniger Gerätesupport, benötigt dafür jedoch keine Serverkomponente. Mit der Verbreitung von WebRTC-fähigen Browsern und WebSocket fähigen SIP Servern würde eine entsprechende Server-orientierte Architektur sowieso obsolet.

Updatezyklen WebRTC wird nur von den neusten Browsern unterstützt. Entsprechend sind die Nutzer angehalten, einen modernen Browser zu verwenden oder ihren alten zu aktualisieren. Durch die wegfallende Unterstützung für ältere Browser und nur teilweise implementierte Schnittstellen fällt unsere Applikation wesentlich schlanker aus.

Sicherheit Die Videokommunikation unserer Applikation läuft in jedem Fall P2P. Der Benutzer kann sicher sein, dass sein Datenstrom nicht über einen Server läuft, auf dem er entschlüsselt und mitgeschnitten wird.

3.4.3. Mehrwert gegenüber klassischen SIP Applikationen

Klassische SIP-Applikationen ermöglichen zwar eine plattformunabhängige und providerunabhängige Nutzung, bieten jedoch immer noch den Nachteil, dass eine Softwareinstallation notwendig ist. Unsere Applikation kann auch auf Clients genutzt werden, deren Benutzer keine Rechte zur Softwareinstallation besitzen.

3.5. Motivation

Motivation zur Entwicklung der Applikation sind vor allem die fehlenden offenen und unabhängigen Alternativen zu proprietären Diensten wie Skype, Google Hangout oder FaceTime und die Notwendigkeit für SIP-Services eine Applikation zu installieren.

[2] http://sipml5.org/

4. Projektorganisation

4.1. Projektteam

Abbildung 4.1.: Tobias Blaser

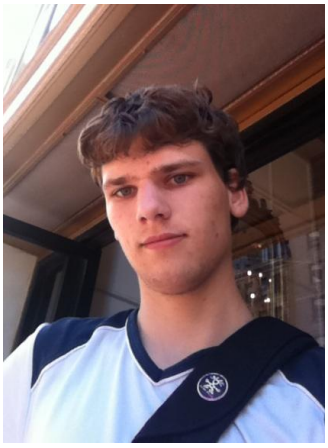


Abbildung 4.2.: Jannis Grimm

4.2. Projektbegleitung

Abbildung 4.3.: Prof. Dr. Luc Bläser

Prof. Dr. Luc Bläser betreut die Semesterarbeit und begleitet das Team durch regelmässige Meetings.

5. Anforderungen

5.1. Allgemeine Beschreibung

5.1.1. Produktfunktion

Anwender sollen Peer-to-Peer Audio- und Videotelefonie durchführen können, ohne dazu eine zusätzliche Software installieren zu müssen. Sie sollen die Telefonate direkt in ihrem Browser durchführen können und ihre Kontaktdaten über verschiedene Schnittstellen laden können.

5.1.2. Benutzer-Charakteristik

Zielgruppe der „WebRTC VoIP Applikation“ sind gewöhnliche Anwender ohne technische Kenntnisse.

5.1.3. Einschränkungen

Voraussetzung für die Nutzung der „WebRTC VoIP Applikation“ ist ein moderner Browser mit WebRTC-Unterstützung (aktuell Firefox, Chrome und Opera, Stand 18.12.13).

5.2. Use Cases

5.2.1. Use-Case-Übersicht

- Anrufen
- Telefonbuch importieren

5.2.2. Use-Case-Beschreibungen

Anrufen

Voraussetzungen: Der Benutzer hat eine Kamera und ein Mikrofon angeschlossen, ist angemeldet und hat einen Account beim gleichen Channel wie die Person, die er anrufen möchte.

Der Benutzer ruft einen anderen Benutzer an. Ist der andere Benutzer erreichbar und verwendet einen kompatiblen Client (WebRTC), so wird eine gemeinsame Audio- oder Videosession gestartet. Am Ende des Telefonates beendet der Benutzer die Session und die Verbindung wird beendet.

Während dem Telefonat wird die Auflösung der Verbindungsqualität angepasst.

Telefonbuch importieren

Voraussetzungen: Der Benutzer ist angemeldet und befindet sich in der Kontaktbuchansicht.

Der Benutzer wählt eine Quelle aus und importiert die entsprechenden Daten. Die Telefonbucheinträge werden dem Benutzer nach Quelle getrennt angezeigt.

5.3. Weitere Anforderungen

5.3.1. Qualitätsmerkmale

Die Applikation soll ohne Einschulung bedienbar sein und die Video- und Audio-Auflösung soll abhängig von der Verbindungsqualität skaliert werden.

Functionality

Interoperabilität Die Applikation soll auf existierende Standards wie SDP[1], STUN[2], HTTP und JSON[3] setzen und wo eigene Standards oder Schnittstellen nötig sind, ist eine Protokoll- oder Schnittstellendokumentation anzufertigen.

Sicherheit Die Audio- und Videokommunikation soll verschlüsselt werden. Die Verbindung zum Server soll Verschlüsselung unterstützen, sodass z. B. TLS[4] genutzt werden könnte.

Usability

Verständlichkeit Die Benutzeroberfläche soll so aufgebaut sein, dass sich ein Benutzer ohne externe Hilfe innerhalb von Minuten darin zurechtfindet. Buttons und Labels sollen sprechend bezeichnet sein.

Robustheit Die Oberfläche soll nicht hängen bleiben bei Operationen, die länger als 5s dauern, und auf Fehler angemessen reagieren.

Portability

Anpassbarkeit Die Applikation soll so gestaltet werden, dass sich die Unterstützung für weitere Channel- und Adressbuch-Technologien hinzufügen lässt. Dazu sollen Schnittstellen definiert und dokumentiert werden.

[1] Javascript Object Notation [1]
[2] Session Traversal Utilities for NAT [10]
[3] JavaScript Object Notation[1]
[4] Transport Layer Security, früher SSL

Installierbarkeit Die Applikation soll auf jedem modernen, WebRTC-fähigen Browser laufen, der sich an die Standards hält.

Austauschbarkeit Die Oberfläche soll durch eigene Styles einfach umgestaltet und angepasst werden können. Es soll eine Template-Engine verwendet werden, sodass Mehrsprachigkeit später hinzugefügt werden kann.

5.3.2. Schnittstellen

Signaling-Schnittstelle

Die Applikation soll als Signaling-Technologie einen offenen Standard (z. B. SIP[5] oder XAMPP[6] oder JSON über XHR[7]) unterstützen.

Telefonbuchschnittstelle

Die Telefonbuchschnittstelle soll offene und verbreitete Standardformate unterstützen (z. B. vCard[8], CSV[9], JSON . . .).

[5] Session Initiation Protocol[9]
[6] Extensible Messaging and Presence Protocol[18]
[7] XML HTTP Request, asynchrones Nachladen von HTTP-Seiten [3]
[8] Elektronische Visitenkarte [2]
[9] Comma Separated Values[19]

6. Architektur

6.1. Architektur-Übersicht

Das folgende Diagramm zeigt eine Übersicht über die notwendigen Komponenten für eine Videoplattform mit JavaScript. Die Applikation ist als „Fat-Client“-Architektur aufgebaut. Es gibt keine anwendungsspezifischen Serverkomponenten für die Applikation selbst. Erst für den Verbindungsaufbau werden Vermittlungsserver benötigt.

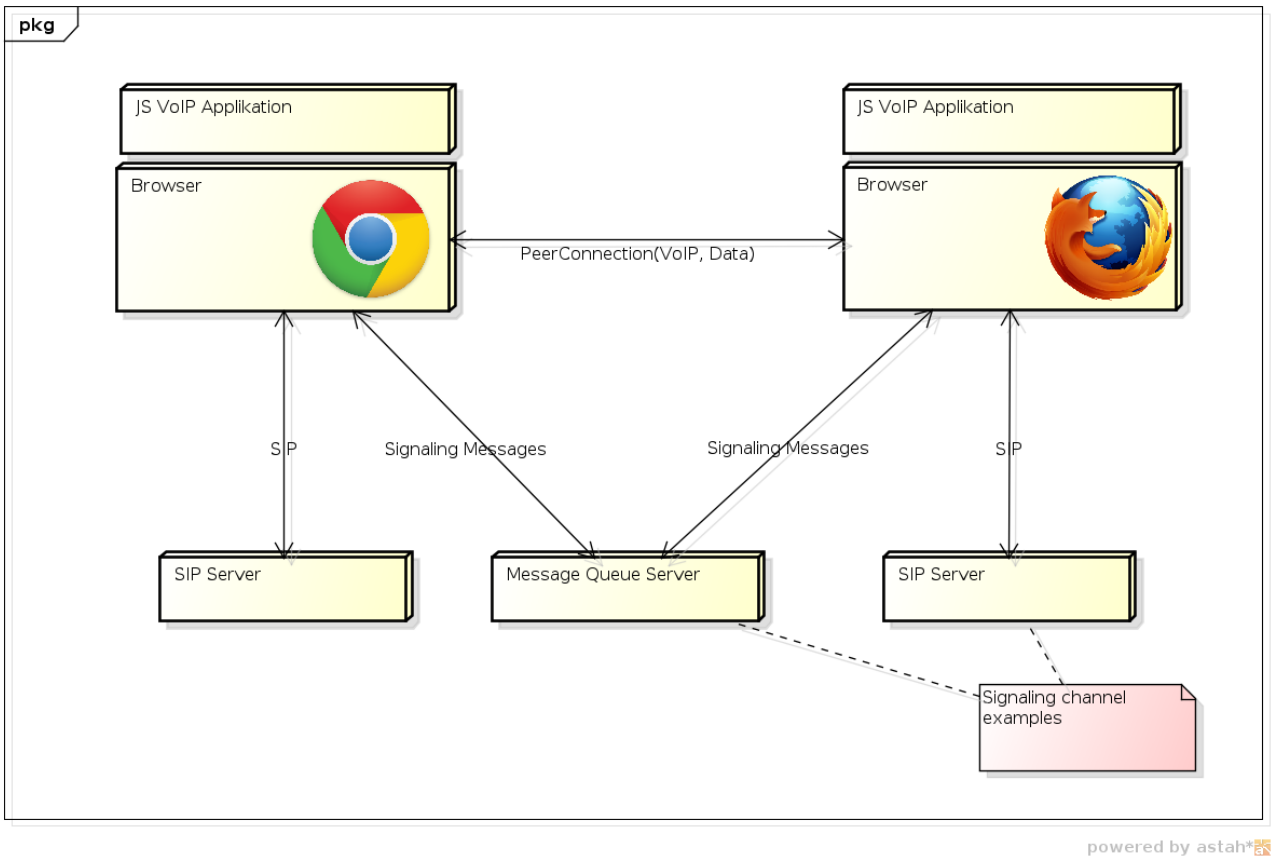


Abbildung 6.1.: Big Picture JS VoIP App

Jeder Client besitzt Signaling Channels, die mit einem Signaling Server interagieren. Hier wurde als Beispiel eine Message Queue und ein SIP Server verwendet. Die Kommunikation läuft entsprechend über XHR oder SIP.

Ein Verbindungsaufbau beginnt bei einem Client. Der Browser generiert eine Offer für den anderen Client und sendet diesen in einer Signaling Message über einen Signaling Server, wie eine einfache Message Queue oder einen SIP Server, an den anderen Client.

Der Browser des anderen Clients generiert eine Answer und schickt diese auf dem gleichen Weg zurück. Anschliessend bauen die Clients die direkte P2P Connection auf. Sobald diese aufgebaut wurde, können Video und Audio gestreamt sowie Daten übertragen werden.

Für den Abbau sendet ein Client ein Bye. Anschliessend bauen beide Clients die P2P Connection ab.

6.1.1. Client/Server vs. reine Client-Applikation

Vorteile Client/Server-Umsetzung

- Schlanker Client
- Es wird nur eine Schnittstelle benötigt zwischen Client und Server, der gesamte Verkehr kann über HTTP abgewickelt werden.
- Nur der Server muss die Schnittstellen zu anderen Diensten unterstützen. Keine zusätzlichen Schnittstellen-Anforderungen an den Client.
- Es ist einfacher durch Firewalls durchzukommen, da die Kommunikation genau kontrolliert werden kann.

Nachteile

- Redundanzen, da Funktionalität zwischen Client und Server durch ein eigenes Protokoll abgebildet und damit auf beiden Seiten Teile derselben Funktionalität umgesetzt werden müssen.
- Ein Anwender ist an den Server oder die Serverimplementation gebunden. Die Applikation läuft nicht ohne den Server.
- Der Server ist ein „Single Point of Failure“, falls keine redundanten Server eingesetzt werden. Zudem stellt der Server eine zentrale Angriffsstelle dar, über die Angreifer einfacher das ganze System blockieren könnten.
- Möchte ein Entwickler die Applikation um weitere Schnittstellen erweitern, so muss er Server und möglichweise den Client trotzdem auch anpassen, was aufwendiger ist. Zudem muss er den Server selbst betreiben und kann keinen bestehenden mehr nehmen, was für ihn zusätzlichen Aufwand bedeutet.

Vorteile reine Client-Lösung

- Es wird kein Server benötigt.
- Anwender können einen beliebigen Kommunikationsserver (beispielsweise SIP[1] oder XMPP[2]) für das Signaling verwenden und sogar selbst eine eigene Channelimplementation hinzufügen.

[1] Session Initiation Protocol [9]
[2] Extensible Messaging and Presence Protocol

- Um die Applikation für weitere Schnittstellen zu erweitern, sind keine Kenntnisse der Servertechnologie notwendig.
- Die Applikation ist insgesamt weniger aufwendig aufgebaut, da sie weniger Redundanzen beinhaltet.
- Es ist keine eigene SIP-/XMPP-Server-Implementation erforderlich (bereits vorhandene Serverinfrastrukturen werden genutzt).

Nachteile

- Die möglichen Schnittstellen sind auf die Browserfunktionalität begrenzt (HTTP, WebSockets, SDP, STUN, UserMedia). Dadurch muss ein SIP- oder XMPP-Provider WebSockets unterstützen, damit er für das Signaling genutzt werden kann.
- Der Client wird umfangreicher.
- Probleme mit Firewalls und Routern möglich, die WebSocket-Pakete nicht weiterleiten. WebSockets können grundsätzlich über einen beliebigen Port laufen, meist wird jedoch Port 80 verwendet.
- Performanceprobleme möglich, da sämtliche Logik, inklusive dem Parsen der verschiedenen Protokolle, vom Client abgearbeitet werden muss.

Fazit

Aufgrund geringerer Abhängigkeiten und der Wahrscheinlichkeit, dass die SIP-Server-Provider irgendwann WebSockets unterstützen, wird die Lösung „reine Client Applikation“ gewählt. Für die Kommunikation mit Signalingservern, wie SIP oder XMPP, werden WebSockets eingesetzt. Für das Signaling über XHR[3] wird mit der Option „Allow Cross Origin“ gearbeitet.

6.2. Architektur-Schichten

Der logische Aufbau der Software besteht aus vier Schichten:

[3] XML HTTP Request

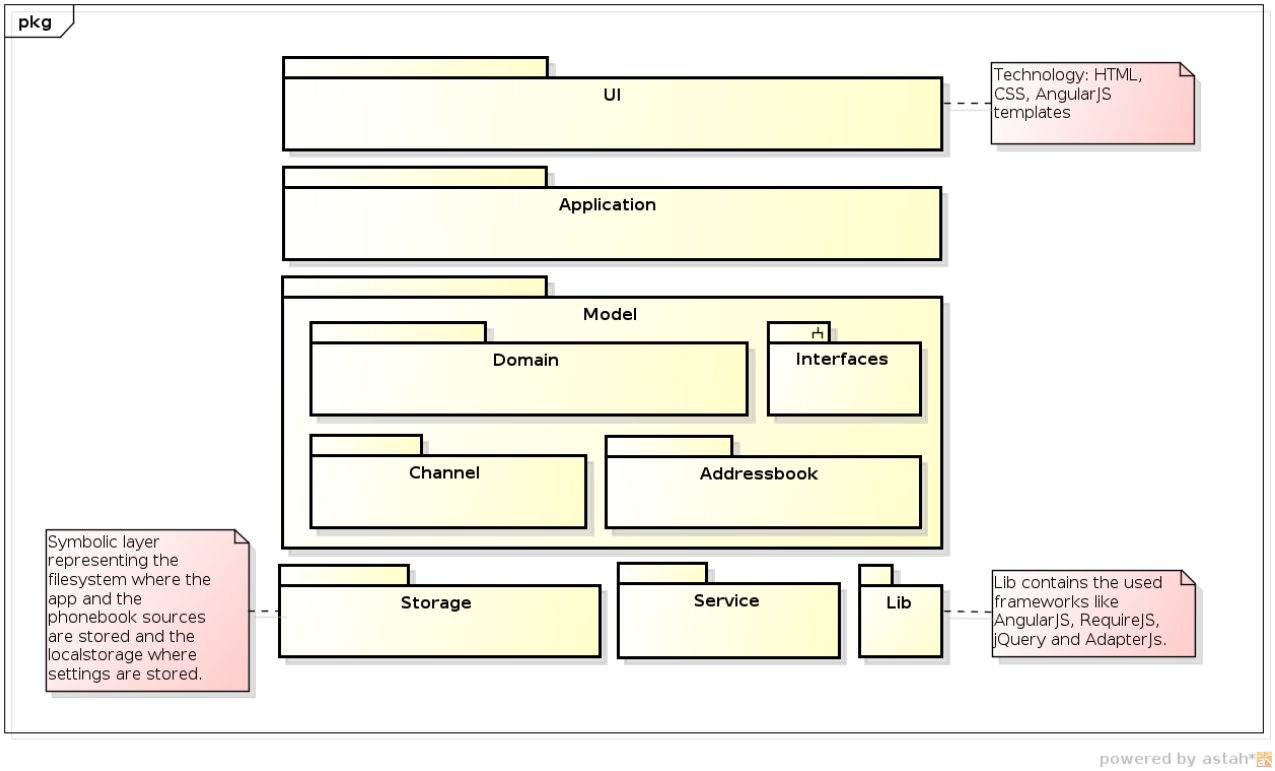
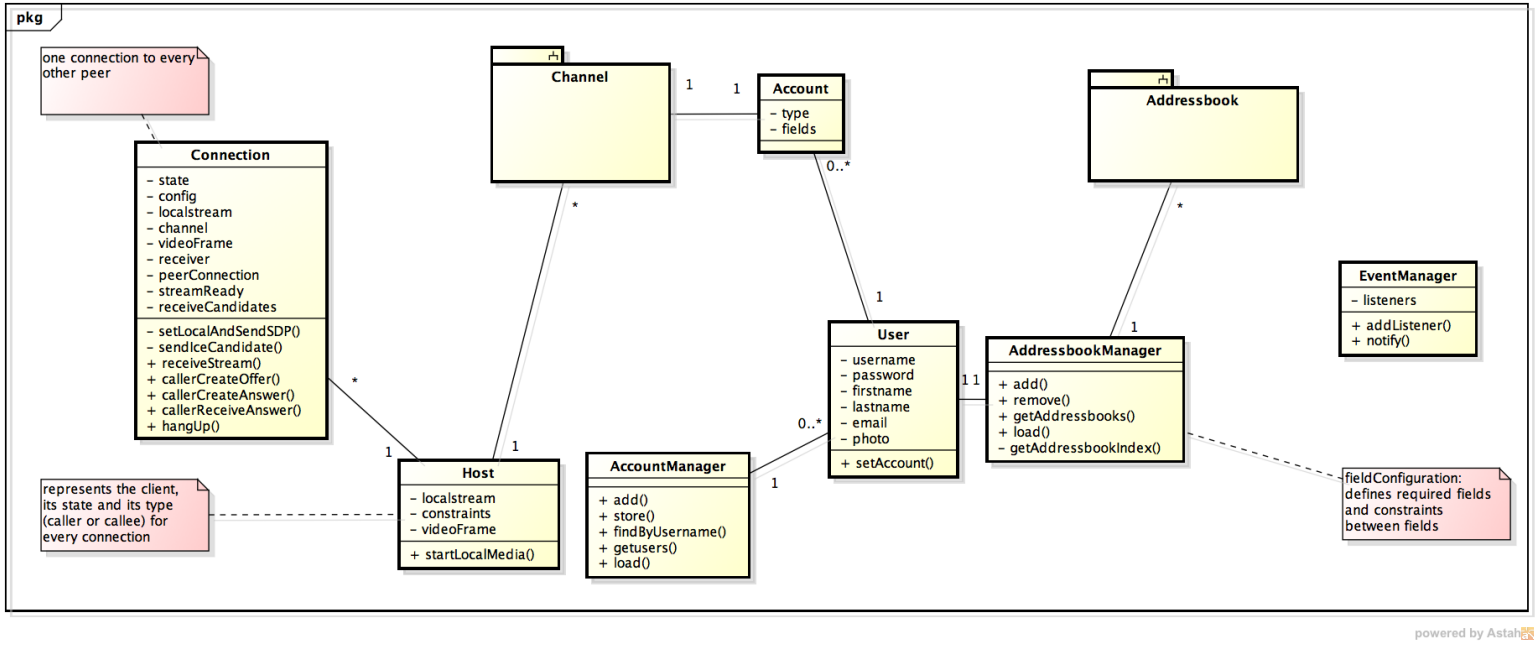


Abbildung 6.2.: Architekturdiagramm JS VoIP App

Im untersten Layer sind Services und Libraries enthalten, auf die die oberen Layer zugreifen. Der Application Layer beinhaltet View Models für das User Interface und interagiert mit dem Model.

6.3. Domainmodel

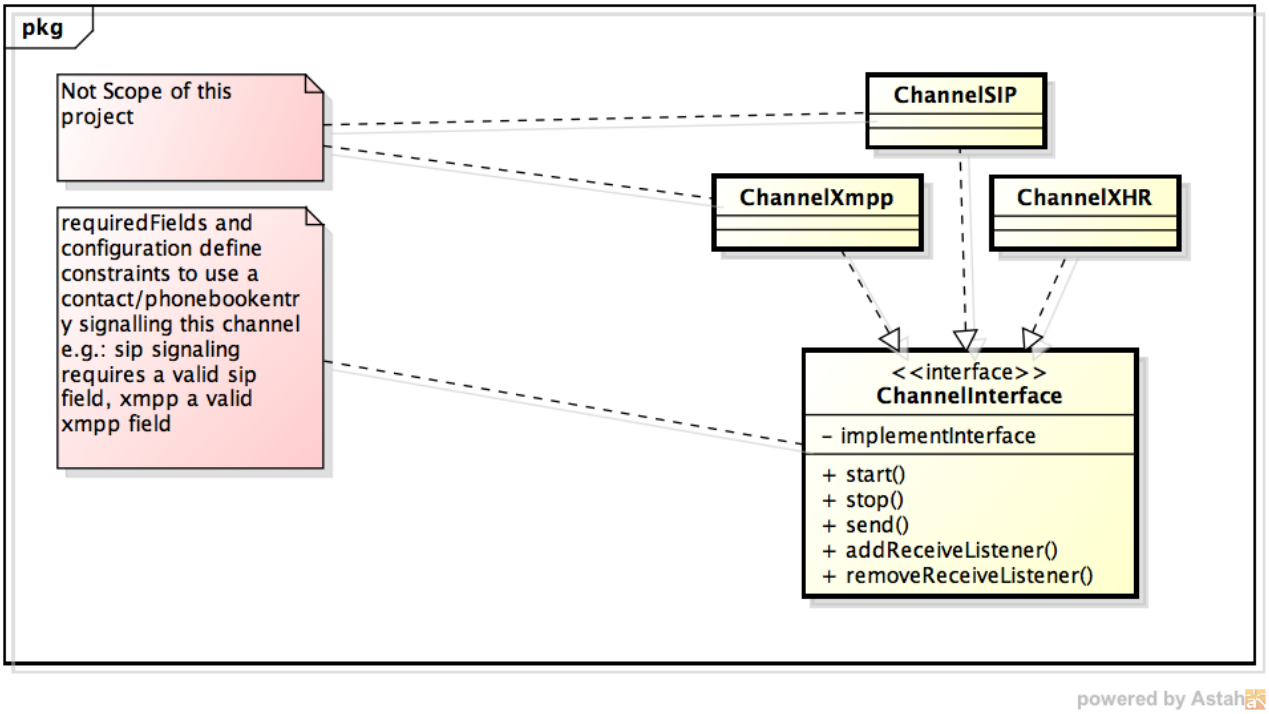
Im folgenden Diagramm sind die wichtigsten konzeptionellen Klassen und ihre Beziehungen untereinander aufgeführt.



Herzstück der Applikation sind die Channels und die Connection sowie das Contactbookmanagement.

6.4. Channel

Die Channels laufen im Hintergrund und verbinden den User mit einem oder mehreren Signaling-Servern. Sie werden beim Login des Benutzers gestartet und müssen entsprechend ein Channel-Interface implementieren. So laufen z. B. ein SIP Channel und ein XHR Channel, abhängig davon, was der Benutzer für Accounts besitzt.



Die interne Implementation des Channels ist dem entsprechenden Entwickler überlassen. Als Referenzimplementation wurde ein Channel über XHR und ein Channel über WebSockets umgesetzt.

Beim XHR-Channel wird mit dem Starten des Channels ein Polling auf den Queue-Server gestartet. Sobald dieser eine Antwort zurückliefert, werden die Listener benachrichtigt.

Beim WebSocket-Channel wird eine WebSocket-Verbindung geöffnet, die anschliessend offen bleibt. Sobald der Queue-Server einen neue Nachricht hat, erhält der Channel diese vom Server und kann die Listener benachrichtigen.

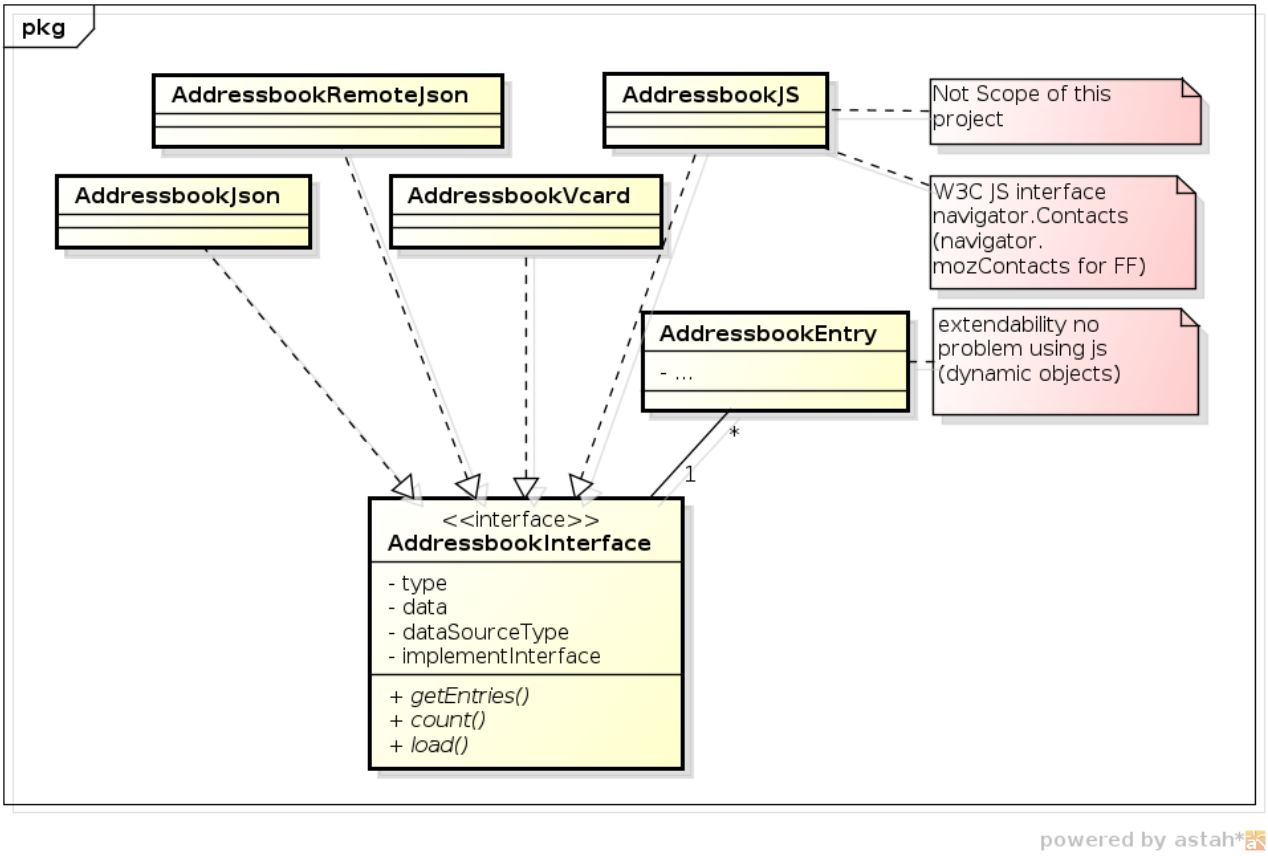
Sowohl der XHR-Channel wie der WebSocket-Channel lassen sich verschlüsselt nutzen, indem beim XHR-Channel über HTTPS[4] zugegriffen wird und beim WebSocket-Channel über WSS[5].

6.5. Adressbook

Die Kontaktverwaltung übernimmt der ContactbookManager. Er importiert Kontaktbücher anhand der Konfiguration von den Quellen „Datei“, „Ordner“ oder „Online“. Für die format-

[4] TLS geschützte HTTP Verbindung
[5] Secure WebSocket, TLS geschützte WebSocket-Verbindung

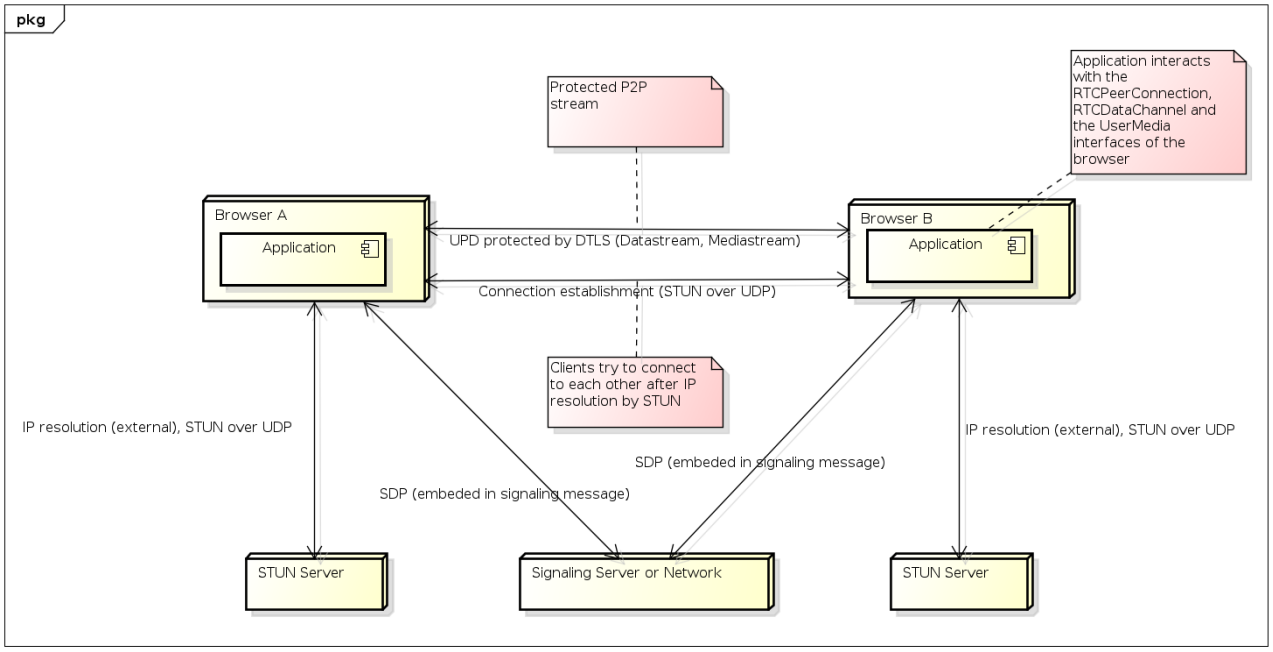
spezifischen Importprozesse sind die jeweiligen Kontaktbücher zuständig. Dazu müssen alle Kontaktbücher das Addressbook-Interface umsetzen.



Kontaktbücher werden im LocalStorage gespeichert und nach einem Neustart wieder geladen. Da im LocalStorage nur serialisierte Daten gespeichert werden können, müssen neue Kontaktbuchobjekte erstellt und die Daten injected werden.

6.6. Connection

Die Connection übernimmt das komplette Handling der P2P-Verbindung inklusive Verbindungsauf- und -abbau. Sie greift über den Host auf die lokale Kamera zu und sendet und empfängt über einen Channel „Offers“ und „Answers/Replies“.



6.6.1. Verbindungsaufbau

Die Kommunikation zwischen zwei Clients, einem STUN-Service und einem Signaling-Service wird im folgenden Diagramm dargestellt.

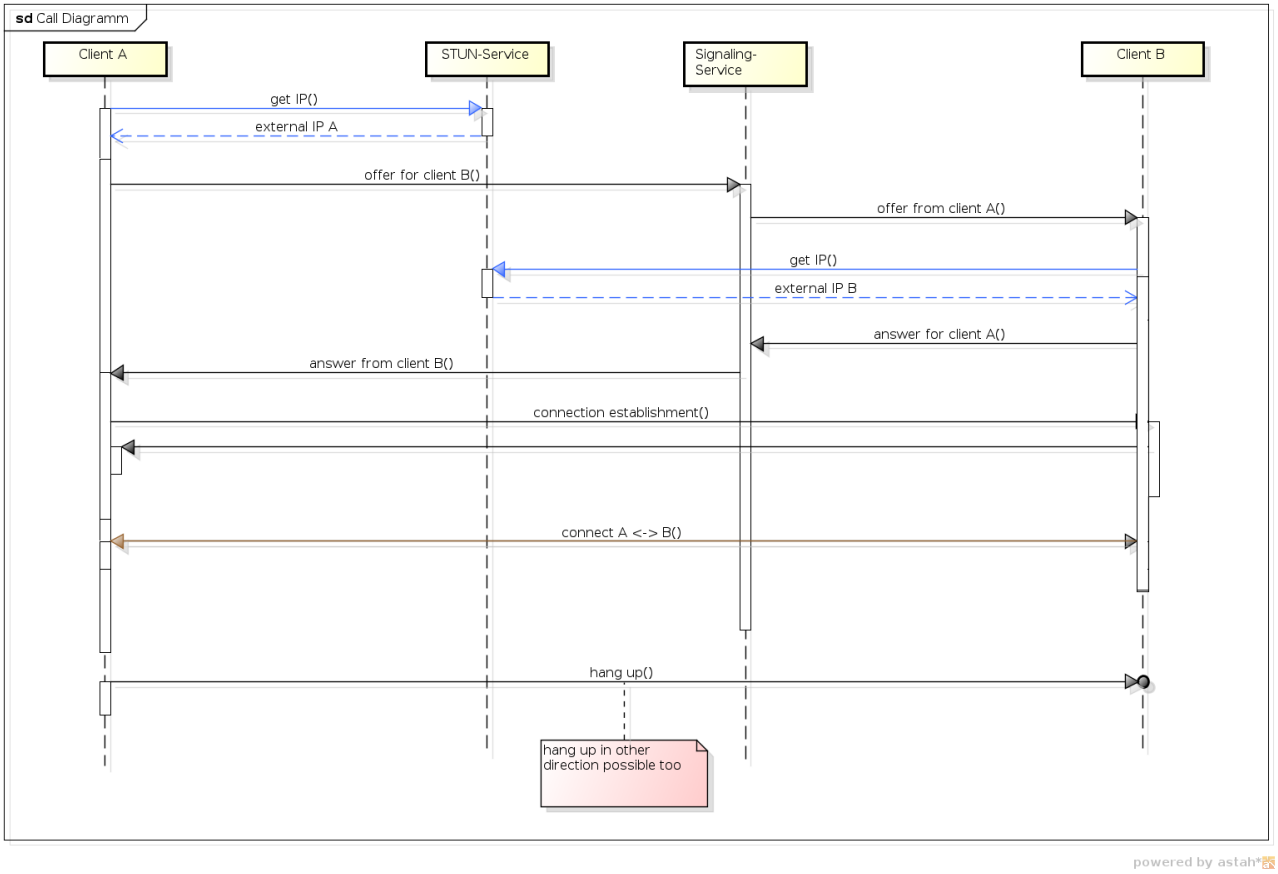


Abbildung 6.3.: Clientkommunikation

Die Clients beziehen vom STUN-Service ihre extern sichtbare IP. Nach dem Austauschen der SDP[6] über den Signaling Channel versuchen die Peers sich gegenseitig direkt zu erreichen mit STUN Messages. Dazu verwenden sie die Ports, die sie von den Candidates[7] des SDP erhalten haben. War dies erfolgreich, wird die PeerConnection aufgebaut und die Datenströme sind verbunden. Die Peers kommunizieren nun direkt und benötigen den Signaling Service nicht mehr.

[6] Session Description Protocol[9]
[7] Endpunktkandidaten (Ports) für DataChannel, Audio- und Videostream

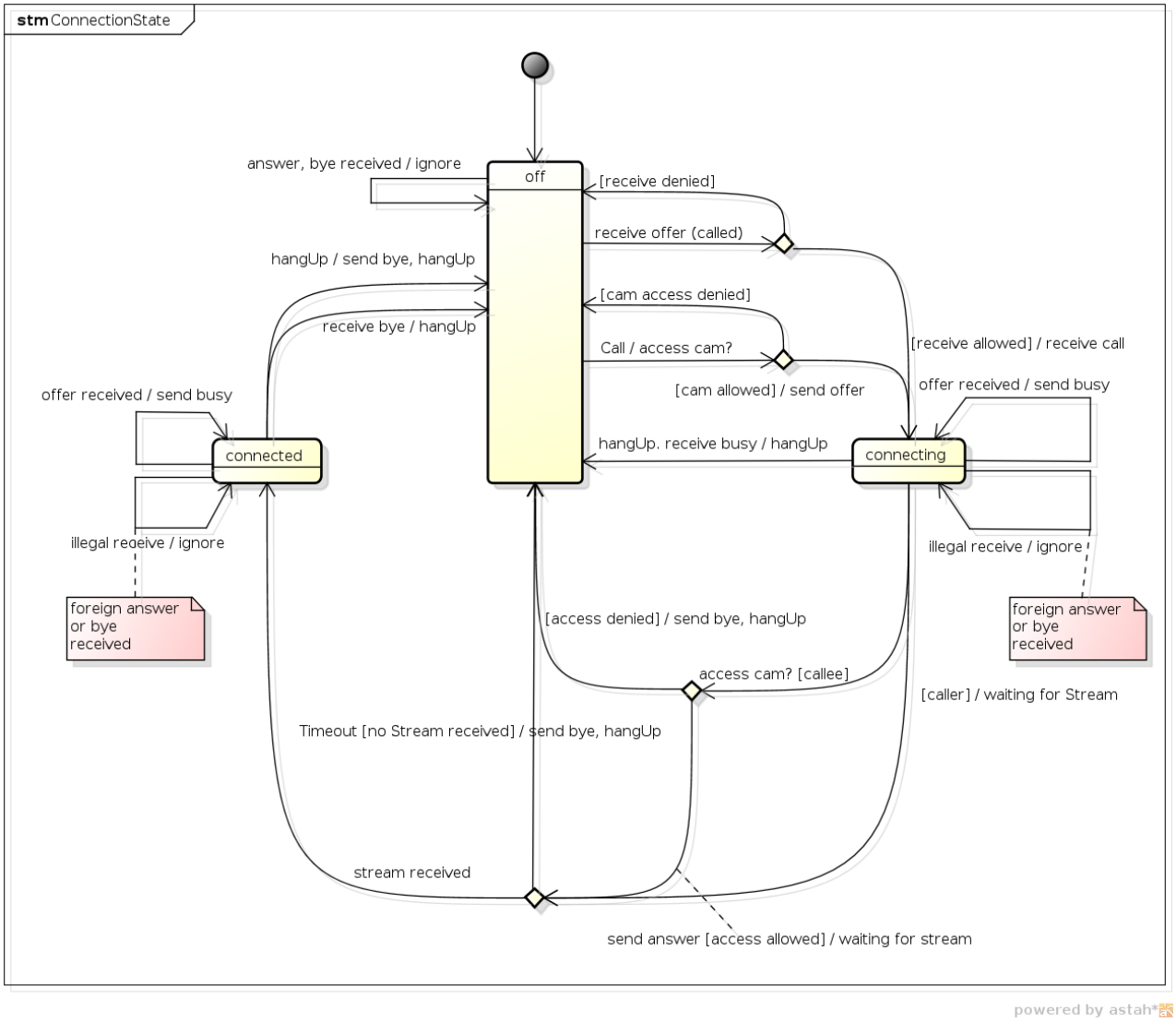


Abbildung 6.4.: Applikationszustände

Beim Verbindungsaufbau nimmt jeder Client entweder die Rolle „Caller“ oder „Callee“ ein. Erlaubt der Benutzer Zugriff auf die Kamera, so wird die Verbindung aufgebaut und bleibt bestehen, bis einer der Clients die Verbindung mit einem „Bye“ beendet.

Ungültige Messages werden ignoriert. „Bye“ führt in jedem Zustand zu einem Abbruch der Verbindung bzw. des Aufbaus, vorausgesetzt das „Bye“ kommt vom anderen Kommunikationspartner.

6.6.2. Verbindungsabbau

Für den Verbindungsabbau teilen sich die Peers dies mit und bauen anschliessend die Verbindung ab. Der Verbindungsabbau kann wahlweise über den Signaling-Server oder direkt über den Datachannel der PeerConnection gesendet werden. Wurde die PeerConnection als unzuverlässig konfiguriert, so sind Bestätigungsnachrichten sinnvoll, damit kein Client

die Verbindung offen behält, obwohl der andere Peer bereits die Verbindung abgebaut hat.

6.6.3. STUN-Service

STUN[8]-Services sind zwingend notwendig, um die nach aussen sichtbare IP-Adresse des Clients herauszufinden. Neben der Möglichkeit, selbst einen STUN-Service aufzusetzen, gibt es frei verfügbare STUN-Services, beispielsweise von Mozilla oder Google.

Für ein abgeschottetes Netz ist ein eigener STUN-Server zwingend notwendig. Für unseren Fall reichen die frei verfügbaren. Durch anpassen der Konfiguration ist es möglich, den Server festzulegen.

Im Netz gibt es fertig konfigurierte virtuelle Maschinen mit einsatzbereitem STUN-Service[9].

6.6.4. SIP-Proxy vs. SIP-Server mit WebSockets

Unterstützt ein SIP-Server keine WebSockets, so kann ein SIP/WebSocket-Proxy dazwischen geschaltet werden. Die Konfiguration eines SIP-Proxies ist ähnlich aufwändig wie die Installation eines Kamailio[10]-SIP-Servers, der in der neusten Version WebSockets unterstützt.

6.6.5. Sicherheit

WebRTC-Streams können nur verschlüsselt übertragen werden. Es gibt keine Möglichkeit, die Verschlüsselung abzuschalten. Dies führt zwar zu einem erhöhten Rechenbedarf auf dem Client, garantiert jedoch eine verschlüsselte Verbindung unabhängig von den Vorlieben des Entwicklers. Die Verschlüsselung erfolgt über DTLS-SRTP[11] keyings[12].

Die Verschlüsselung des Signaling-Channels ist abhängig von der eingesetzten Technologie. Werden z. B. Secure WebSockets oder XHR über HTTPS eingesetzt, so ist die Kommunikation verschlüsselt und nicht abgreifbar.

Obwohl die Daten sicher übertragen werden, kann ein Provider oder eine anderweitige Zwischenstelle Metadaten darüber sammeln, wer mit wem telefoniert.

6.7. Third Party Code

Für den Unterbau und das Testing werden die folgenden Frameworks genutzt:

- adapter.js
- AngularJS
- jQuery

[8] Session Traversal Utilities for NAT [10]
[9] Mozilla, stun-vm [4]
[10] Kamailio SIP Server Project, [5]
[11] Datagram Transport Layer Security [6]
[12] Adam Roach, WebRTC: Security and Confidentiality [7]

- LESS
- RequireJS
- QUnit

Für das Benutzerinterface werden die Iconfonts „Batch“ und „Ligature“ verwendet.
Für die Priorisierung des Codecs Opus wurde ein Skript zur SDP-Verarbeitung von Google verwendet (SDPService).

7. Performance-Analyse

7.1. Leistungsverbrauch

Zur Analyse der Performance wurden Versuche mit Desktoprechnern und Mobilgeräten gemacht. Erwartet wurde, dass die Performance auf den Desktopgeräten wesentlich besser ist als bei den Mobilgeräten. Angeschaut wurden die CPU-Belastung, RAM-Verbrauch und Netzwerktraffic sowie Audio- und Videoqualität.

Es hat sich gezeigt, dass die Performance sehr stark von der Leistung des Prozessors abhängt. Die Vermutung, dass Desktopgeräte besser performen als Mobilgeräte war jedoch falsch. Auf den Desktoprechnern verbraucht ein P2P-Stream ca. 10–20 % CPU-Leistung, auf dem Netbook und dem Tablet wesentlich mehr. Das verwendete Smartphone lag näher bei den Desktopgeräten, wurde allerdings spürbar warm.

7.2. Speicherverbauch und Netzwerktraffic

Der Speicherverbauch ist bei allen Geräten vernachlässigbar und im Systemmonitor praktisch nicht zu sehen.

Der Netzwerktraffic hängt von zwei Faktoren ab, von der Leistungsfähigkeit des Prozessors sowie der effektiv verfügbaren Bandbreite zwischen den Peers. Videodecodierung und Verschlüsselung verbrauchen in den aktuellen Versionen von Firefox und Chrome noch viel Leistung. Netbook und Tablet hatten entsprechend Mühe, einen Datenstrom mit hoher Datenrate zu liefern. Das Empfangen bereitete keinem der Geräte Probleme.

Die Browser fahren die Videoauflösung langsam hoch, sobald sie feststellen, das genügend Leistung und Bandbreite vorhanden ist. Dies lässt sich gut zeigen, indem ein verbundenes Smartphone gedreht wird. Dabei wird das Video ebenfalls gedreht und neu dargestellt, wobei die Auflösung und der Bandbreitenverbrauch ab einem Minimum von ca. 50KiB/s kontinuierlich ansteigen und sich bei ca. 300 KiB/s einpendeln.

7.3. Zusammenfassung

Die folgende Tabelle fasst die Bedingungen kurz zusammen, unter denen eine gute und akzeptable Performance möglich ist:

	Gute Performance	**Akzeptable Performance**	**Schlechte Performance**
Audio/Video	• Video flüssig • Audio verständlich	• Video Einzelbilder • Audio verständlich	• Nur Audio möglich
Bandbreite zwischen den Peers	> 50KiB/s	> 25KiB/s	< 25KiB/s
Prozessor	> 1.5Ghz, 2 Core	> 1Ghz, 2 Core	weniger Leistung
Kamera Auflösung	> 640 x 480 Pixel (Maximale Auflösung in den aktuellen Browser-Implementationen)		

Bei schwächeren Geräten und einer kleineren möglichen Datenrate beeinträchtigt das Video die Audioübertragung zu stark. Entsprechend lohnt es sich, unter diesem Umständen das Video gar nicht zu aktivieren.

Während dem Entwickeln ist aufgefallen, dass die Performance von Chrome etwas besser ist als die von Firefox. Ist die Performance auf einem Gerät mit Firefox zu schlecht, so kann man es mit Chrome erneut versuchen.

Ausführliche Performance-Analyse siehe Anhang E.

7.4. Fazit

Videocodierung und Verschlüsselung benötigen noch sichtlich viel Rechenleistung. Desktop- und Mobilgeräte im mittleren- und oberen Leistungssegment sind allerdings in der Lage, die notwendige Leistung zu liefern.

Die bevorzugten Codecs, die auf der VP8-Engine basieren, werden bisher von keinem Gerät hardwareseitig unterstützt. Nach der Veröffentlichung einer OpenSource-Implementation des Codecs h.264 durch Cisco ist zu erwarten, dass dieser Einzug in die Browser hält und durch die bereits existierenden Hardwarechips die erforderliche Rechenleistung enorm senken wird.

Ebenfalls ist zu erwarten, dass zukünftige Implementation der Videocodecs ohne Hardwareunterstützung einen Teil über die GPU rendern und damit erhebliche Performanceverbesserungen bringen werden.

8. Firewall Testing

Beim Firewall Testing ging es darum, herauszufinden, in welchen Netzwerkumgebungen WebRTC-Kommunikation eingesetzt werden kann und in welchen nicht.

8.1. Signaling

Für das Signaling können verschiedene Protokolle eingesetzt werden, bevorzugterweise HTTP oder WebSockets. Davon hängt auch ab, ob die Pakete von Firewalls ausgefiltert werden oder nicht.

WebSocket Pakete sind insbesondere älteren Firewalls noch nicht bekannt und werden von diesen verworfen. Dies betrifft jedoch vor allem Unternehmensnetzwerke. In den Netzwerken der Schweizer Provider sowie im Netzwerk der HSR konnten wir dies nie beobachten. Mit VPN könnte man das Problem umgehen, sofern VPN im Netzwerk zugelassen ist.

Wird HTTP für das Signaling eingesetzt, kommt es zu keinen Paketverwürfen, da HTTP von allen Firewalls erlaubt wird. In Unternehmen ist es vorstellbar, dass unverschlüsselte HTTP-Pakete analysiert und aufgrund der enthaltenen SDP blockiert werden. Durch den Einsatz von HTTPS lässt sich dies jedoch lösen.

Werden Signaling Pakete blockiert, so erhalten die Peers nie Verbindungsanfragen von anderen Peers.

8.2. P2P Connection

Für den Aufbau der Peer-to-Peer Connection ermitteln beide Clients über einen STUN-Service ihre externe IP, tauschen diese über den Signalingchannel mittels SDP[1] aus und verbinden sich direkt (P2P), um Parameter für die Verschlüsselung auszutauschen. Für diesen Austausch wird ebenfalls das UDP basierte STUN-Protokoll verwendet.

In stark reglementierten Umgebungen werden unter Umständen diese STUN-Pakete geblockt. Dadurch kann keine Verbindung zwischen den Clients aufgebaut werden und die Applikation wird nach dem eingestellten Timeout den Verbindungsaufbau abbrechen. Um durch eine derart restriktive Firewall hindurchzukommen, kann ein VPN Client eingesetzt werden, vorausgesetzt VPN ist zugelassen.

In Netzwerkumgebungen, wie sie Heimanwender oder kleinere Firmen verwenden, sollten STUN-Pakete nicht blockiert werden. Auch im Mobilfunk konnten wir dies bisher nicht beobachten.

Detailierte Firewalltests: Siehe Anhang F.

[1] Session Description Protocol

9. Erkenntnisse

Die während der Entwicklung der Architekturprototypen und dem Ausbau der Connection gesammelten Erkenntnisse sind hier zusammengetragen:

9.1. Media Streaming

Verbindungsabbruch Bricht die Verbindung ab, so bleibt die Video- und Audiowiedergabe stehen, bis die Verbindung wieder da ist und die UDP Pakete wieder ankommen.

Downsampling Der Mediastream wird automatisch mit Minimalqualität gestartet und abhängig von Prozessorleistung und Bandbreite hochgefahren, bis zu einer Datenrate von etwa 300MiB/s.

Resolution Properties Obwohl die Spezifikation für getUserMedia eine Möglichkeit zur Festlegung der gewünschten Auflösung definiert, ist dies in den Browsern bisher nicht implementiert.

Audio/Video attach/detach Hinzufügen und Entfernen von Streams sollte eigentlich möglich sein. Die Browser implementieren dies bisher jedoch nicht. Gemäss dem Mozilla-Blog soll dieses Feature jedoch demnächst umgesetzt werden.

Autorotation Video Wird als Endgerät ein Smartphone oder Tablet mit integriertem Beschleunigungssensor genutzt, so wird das Video automatisch gedreht, sobald das Smartphone gedreht wird. Dadurch ist der Teilnehmer immer in korrekter Ansicht zu sehen, egal ob das Device Quer- oder Hochformat gehalten wird. Auch das Videoelement beim Empfänger erhält die Information über die Änderung des Videoformates und skaliert das Videoframe entsprechend, sofern die Grösse nicht durch Styles übersteuert wurde.

9.2. Peerconnection

Eine Peer-Connection kann nur durch den Austausch von SDP aufgebaut werden. Innerhalb der SDP werden Parameter für die Peer-Connection sowie IP und allenfalls Endpunktkandidaten übertragen.

Endpunktkandidaten Je nach Implementation übertragen die Browser die Endpunktkandidaten (Candidates) einzeln und nicht in einer bestimmten Reihenfolge. Ist die Verbindung noch nicht initialisiert worden, so müssen diese zwischengespeichert werden, bis die Verbindung bereit ist.

Verschlüsselung Der übertragene Stream ist standardmässig verschlüsselt. Die Verschlüsselung ist Teil des Standards und kann nicht abgeschaltet werden.

Verbindungsaufbau Ice Candidates müssen einer RTCPeerConnection attatched werden, bevor eine Offer oder eine Answer generiert wird. Ansonsten hat der andere Endpunkt zu wenig Informationen um eine Verbindung aufzubauen. Die Ice Candidates sind Proposals für Verbindungsendpunkte (Ports). Die Browser senden einander mehrere möglich Kombinationen für jede Art von Verbindung (DataChannel, Video, Audio). Anschliessend werden alle Kombinationen probiert, bis eine funktioniert.

9.3. Filezugriff

Dateizugriff ist in JavaScript nur über einen User-Event möglich (Upload Field oder Drag/Drop). Daher ist es nicht möglich, Adressbücher auf die Festplatte abzulegen und wieder zu öffnen, ohne den User dazu aufzufordern.

9.4. Offline-Einschränkungen

STUN-Services Ohne eigener STUN-Service ist eine Internetverbindung zwingend. Für Telefonie in einem abgeschotteten Netz wird ein eigener STUN-Service benötigt. Es ist nicht möglich auf die STUN-Services zu verzichten und die Adresse manuel zu setzen, weil das STUN-Protokoll auch den P2P-Verbindungsaufbau übernimmt.

9.5. SIP

WebSockets Obwohl der OpenSource-SIP-Server Kamailio WebSocket als Feature aufführt, ist die Implementation noch nicht vollständig, sodass SIP-Kommunikation über WebSockets noch unmöglich ist.

9.6. Browserunterschiede

ICE Candidates Firefox packt die ICE Candidates[1] direkt in die Session Description ein. Chrome schickt sie einzeln. Da Chrome nicht auf die Reihenfolge achtet, kann es passieren, dass ICE Candidates vor einer Offer ankommen. Aus diesem Grund müssen ankommende ICE Candidates in einer Queue gebuffert werden, bis eine Offer ankommt.

DataChannel Firefox unterstützt DataChannel[2] bereits vollständig. Chrome unterstützt ihn nur im Modus „reliable: false“. Setzt man ihn auf „reliable: true“, kommen die Events nicht zuverlässig an.

[1] Vorschläge für Ports
[2] P2P-Datenkanal parallel zum Video- und Audiostream

Media Access Firefox erlaubt mehrfachen Kamera-Access und teilt sich die Kamera auch problemlos mit anderen Prozessen. So ist es im Firefox möglich, von einem Tab in einen anderen anzurufen. Chrome will die Kamera komplett für sich beanspruchen und wirft eine Fehlermeldung, falls sie bereits von einem anderen Prozess genutzt wird.

Schnittstellenbezeichnung Die Schnittstellen unterstützen beide Browser erst mit Präfix. Die offizielle Bezeichnung wird noch nicht unterstützt. Adapter.js wurde dafür entwickelt und mappt die offiziellen Schnittstellenbezeichnungen auf die mit Prefix, sodass in der Applikation trotzdem mit den offiziellen gearbeitet werden kann und auch Kompatibilität besteht, sobald die Browser auf die definitiven Schnittstellen umstellen.

10. Schlussfolgerungen

10.1. Zielerreichung

Es ist gelungen, mit WebRTC eine funktionsfähige Video-Over-IP-Applikation zu entwickeln, die ohne Serverkomponenten auskommt.

Die Applikation benutzt die neuen WebRTC-Schnittstellen „MediaStream“, „RTCPeerConnection“ und „RTCDataChannel“. Daneben werden „LocalStorage“ für die Speicherung der Kontaktbücher und der Benutzeraccounts sowie „FileReader“ für den Import der Kontaktbücher verwendet.

Die Benutzeroberfläche wurde responsive gestaltet um auf dem Desktop wie auf Mobilgeräten eine angenehme Benutzung zu ermöglichen.

10.2. Offene Punkte

Internationalisierung, SIP-Support, CSV Contactbook Adapter wurden bereits im dritten Viertel des Projektes aus den Featurelist entfernt, da sie tieferer Priorität waren und ein umfangreiches Refactoring der Core-Funktionalität wichtiger war.

Auf eine Authentifizierung beim Channel und entsprechende Mechanismen im Queue-Server wurde verzichtet, da es sich nur um eine Referenzimplementation handelt und jeder Channel dies selbst umsetzen kann.

10.3. Bugs und technische Probleme

Es ist nicht gelungen, SIP als Signaling-Protokoll einzusetzen, da die SIP-Server WebSockets erst seit Kurzem unterstützen und die Schnittstelle im verwendeten SIP-Server Kamailio noch nicht stabil ist. WebSockets sind zwingend notwendig.

10.4. Zusätzlich umgesetzte Features

Durch die Umsetzung des Verbindungsabbaus über den P2P Datachannel wurde die Voraussetzung geschaffen, um mit wenig Zusatzaufwand Chat-Funktionalität in die Applikation einbauen zu können. Chatnachrichten werden als sogenannte User-Messages über den Datachannel gesendet und sind somit ebenfalls verschlüsselt.

10.5. Erweiterbarkeit

Es ist gelungen, die Applikation so erweiterbar zu entwickeln, dass in einem weiteren Projekt Funktionen wie Dateiaustausch, Konferenzschaltung oder Screensharing hinzugefügt werden können.

10.6. Schlussbericht Jannis

Durch meinen nachträglichen Wechsel in das Projektteam in der dritten Woche war es zu Beginn des Projekts etwas anstrengender, da ich in kurzer Zeit sämtliche Vorarbeiten nachvollziehen musste, um produktiv mitarbeiten zu können.

Es war sehr spannend, neue Web-Technologien zu erforschen, gleichzeitig aber auch frustrierend, als ich feststellen musste, dass ich SIP über WebSockets nicht umsetzen konnte. Somit musste ich die Arbeit von zwei Wochen aufgeben.

Es war jedoch sehr toll, dass am Ende alles wie geplant funktioniert hat und die Applikation selbst als Prototyp sehr gut nutzbar war und fehlerfrei funktionierte.

Von meinem Teamkollegen und von unserem Betreuer habe ich mich sehr gut unterstützt gefühlt und würde jederzeit wieder gerne eine Arbeit umsetzen.

10.7. Persönlicher Schlussbericht Tobias Blaser

Persönlich habe ich sehr viel gelernt, vor allem im Bereich JavaScript Architektur und Frameworks. Auch im Bereich CSS habe ich mir einiges an neuem Wissen aneignen können, vor allem was Flexbox und LESS betrifft.

Das Erforschen einer sehr modernen Technologie zog die Vorteile mit sich, das als Zielgruppe nur moderne Browser zum Einatz kommen konnten. Dies hat sich sehr positiv auf die Entwicklung ausgewirkt. Wir konnten modernste CSS Funktionalität und JavaScript Schnittstellen verwenden und mussten uns keine Sorgen um nicht unterstützende Browser machen, was sehr angenehm war.

10.7.1. Teamarbeit

Unangenehm war die Unsicherheit zu Begin der Studienarbeit wegen der entzogenen Projektzulassung meines ursprünglichen Partners.

Trotz dieser anfänglichen Unsicherheiten hat die Zusammenarbeit nach dem Teamwechsel sehr gut funktioniert. Wir haben beide sehr engagiert gearbeitet und viel Zeit investiert. Besonders in den letzten Wochen.

Die Teamkoordination war angfangs allerdings nicht immer einfach, da wir an unterschiedlichen Tagen an der SA arbeiteten. Nachdem ich meine Büro-Arbeitstage geschoben hatte, wurde das Zusammenarbeiten wesentlich einfacher.

Auch die Zusammenarbeit mit Herrn Bläser war sehr angenehm. Wir haben sehr viel Unterstützung und wertvolle Inputs erhalten und Herr Bläser hat ein Code Review der Applikation durchgeführt. Hierfür möchte ich mich an dieser Stelle herzlich bedanken.

10.7.2. Fazit

Für die knappe Zeit hatten wir uns sehr viel vorgenommen. Trotzdem haben wir es geschafft, sogar noch optionale Features einzubauen.

Das Projekt hat eine Menge Spass gemacht, auch wenn es zwischendurch Moment gab, in denen mich JavaScript durch seine Eigenheiten einiges an Nerven gekostet hat.

Nicht ganz so viel Spass hat das Finishing der Dokumentation gemacht. Vor allem weil durch Abstract, Management Summary, Hauptteil und Anhang viele Themen vier Mal durchgekaut werden mussten um sie in unterschiedlicher Kompaktheit zu produzieren.

Literaturverzeichnis

[1] IETF: „The application/json Media Type for JavaScript Object Notation (JSON)“, RFC4627 (Stand Juli 2006), http://tools.ietf.org/html/rfc4627, [Abruf am 21.11.13]

[2] IETF: „vCard Format Spezifikation“, RFC6350 (Stand August 2011), http://tools.ietf.org/html/rfc6350, [Abruf am 21.11.13]

[3] Mozilla: „HTTP access control (CORS)“ https://developer.mozilla.org/en/docs/HTTP/Access/Access_control_CORS, [Abruf am 07.10.13]

[4] Mozilla: STUN-VM (Stand: 15.01.13), https://github.com/mozilla/stun-vm, [Abruf am 28.10.13]

[5] Kamailio SIP Server Project: Kamailio Features (Stand: 18.03.2013), http://www.kamailio.org/w/features/, [Abruf am 28.10.13]

[6] IETF: „Datagram Transport Layer Security“, RFC5764 (Stand: 2010), tools.ietf.org/html/rfc5764, [Abruf am 28.10.13]

[7] Adam Roach: „WebRTC: Security and Confidentiality“ (Stand: 07.06.13), http://sporadicdispatches.blogspot.ch/2013/06/webrtc-security-and-confidentiality.html, [Abruf am 28.10.13]

[8] Mozilla: „DOM Storage guide“, https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage, [Abruf am 21.10.13]

[9] IETF: „SDP: Session Description Protocol“ (Stand: Juli 2006), http://tools.ietf.org/html/rfc4566, [Abruf am 28.10.13]

[10] IETF: „Session Traversal Utilities for NAT (STUN)“ (Stand Oktober 2008), http://tools.ietf.org/html/rfc5389, [Abruf am 05.11.13]

[11] QUnit: „QUnit API Documentation“, http://api.qunitjs.com/, [Abruf am 30.09.13]

[12] LESS: „The dynamic stylesheet language“, http://lesscss.org/#docs [Abruf am 25.11.13]

[13] Chris Coiyer: „http://css-tricks.com/snippets/css/a-guide-to-flexbox/“, http://css-tricks.com/snippets/css/a-guide-to-flexbox/, [Abruf am 14.10.13]

[14] RequireJS: „RequireJS API“, http://requirejs.org/docs/api.html, [Abruf am 28.11.13]

[15] AngualrJS: „AngularJS API Docs“, http://docs.angularjs.org/api/, [Abruf am 28.11.13]

[16] HTML5rocks: „http://www.html5rocks.com/en/tutorials/webrtc/basics/“, http://www.html5rocks.com/en/tutorials/webrtc/basics/, [Abruf am 23.09.13]

[17] Muaz Khan: „How to use RTCDataChannel“, https://www.webrtc-experiment.com/docs/how-to-use-rtcdatachannel.html [Abruf am 07.11.13]

[18] IETF: „Extensible Messaging and Presence Protocol (XMPP): Core“, http://tools.ietf.org/html/rfc3920, [Abruf am 16.12.13]

[19] IETF: „Common Format and MIME Type for Comma-Separated Values (CSV) Files“, http://www.ietf.org/rfc/rfc4180, [Abruf am 16.12.13]

A. Risikomanagement

A.1. Risiken

Risiko-ID	1
Titel	Implementierungshindernis
Beschreibung	Aufgrund einer nicht bedachten Schwierigkeit verzögert sich die Entwicklung des Programms.
max. Schaden	10h
Eintrittswahrscheinlichkeit	0.4
Gewichteter Schaden	4h
Vorbeugung	Sorgfältige Abklärungen im Vorfeld der Implementierung.
Massnahmen	Zusätzliche Entwicklungszeit, um das Problem zu lösen.

Risiko-ID	2
Titel	Verbindungsverlust
Beschreibung	Kurze Unterbrüche in der Verbindung könnten die ganze Session beenden.
max. Schaden	8h
Eintrittswahrscheinlichkeit	0.9
Gewichteter Schaden	7.2h
Vorbeugung	Einarbeitung in SIP Connection Management.
Massnahmen	Session bei kleinen Unterbrüchen aufrecht erhalten und einen schnellen Reconnect einrichten.

Risiko-ID	3
Titel	Browserperformance
Beschreibung	Video und Audio müssen flüssig wiedergegeben werden.
max. Schaden	3h
Eintrittswahrscheinlichkeit	0.1
Gewichteter Schaden	0.3h
Vorbeugung	Performance-Tests mit Prototypen.
Massnahmen	Skalierung der Videoqualität, notfalls Priorisierung auf Audio.

Risiko-ID	4
Titel	Langsame Verbindung
Beschreibung	Bei langsamen Verbindungen kann es zu Problemen in der Wiedergabe führen.
max. Schaden	8h
Eintrittswahrscheinlichkeit	0.8
Gewichteter Schaden	6.4h
Vorbeugung	Performance-Tests mit Prototyp und künstlichem Traffic.
Massnahmen	Skalierung der Videoqualität, notfalls Priorisierung auf Audio.

Risiko-ID	5
Titel	Konferenzschaltung
Beschreibung	Hohe Teilnehmerzahl führt zu hoher Anzahl Verbindungen.
max. Schaden	1h
Eintrittswahrscheinlichkeit	1.0
Gewichteter Schaden	1h
Vorbeugung	Performance-Tests mit Prototypen.
Massnahmen	Maximale Anzahl Teilnehmer festlegen.

Risiko-ID	6
Titel	Kompetenzmangel SIP
Beschreibung	Erfahrung mit SIP mangelt. Unbekannte Probleme möglich.
max. Schaden	14h
Eintrittswahrscheinlichkeit	0.7
Gewichteter Schaden	9.8h
Vorbeugung	Frühe Einarbeitung in SIP.
Massnahmen	Zusätzliche Entwicklungszeit.

Risiko-ID	7
Titel	NAT & Firewall Traversal
Beschreibung	NATs und Firewalls lassen sich nicht ohne Weiteres durchdringen.
max. Schaden	20h
Eintrittswahrscheinlichkeit	0.25
Gewichteter Schaden	5h
Vorbeugung	Früh Verbindungstests mit NATs und Firewalls durchführen.
Massnahmen	Zusätzliche Entwicklungszeit investieren zur Unterstützung von Proxyservern und Tunneling-Mechanismen.

Risiko-ID	8
Titel	Verschlüsselte Verbindung
Beschreibung	Das Verschlüsseln der P2P-Verbindung und des Server-Connects gestalten sich schwieriger als gedacht.
max. Schaden	16h
Eintrittswahrscheinlichkeit	0.5
Gewichteter Schaden	8h
Vorbeugung	Früh Möglichkeiten zur Verschlüsselung und deren Implementierungsaufwand analysieren.
Massnahmen	Zusätzliche Entwicklungszeit investieren zur Umsetzung des Verschlüsselungsmechanismus.

Risiko-ID	9
Titel	Social Risks
Beschreibung	Die Projektmitglieder sind aufgrund anderer Projekte zu stark absorbiert und können zu wenig Zeit für das Projekt aufwenden.
max. Schaden	25h
Eintrittswahrscheinlichkeit	0.25
Gewichteter Schaden	6h
Vorbeugung	Externe Projekte in der Iterationsplanung berücksichtigen.
Massnahmen	Projektmitglieder opfern ihre Freizeit und trinken mehr koffeinhaltige, flügelverleihende Energy Drinks.

A.2. Umgang mit Risiken

Um die Riskien möglichst kalkulierbar zu halten, ist es für dieses Projekt entscheidend, möglichst früh Evaluationen durchzuführen. Besonders die frühe Einarbeitung in SIP sowie die Bereitstellung eines ersten Prototypen mit SIP-Verbindung muss gewährleistet werden. Sobald Prototypen vorhanden sind, müssen erste Performance-Test durchgeführt werden. Weiterhin sollten die Risiken laufend neu beurteilt werden.

B. Projektplan

Die einzelnen Arbeitspakete werden mithilfe des Projektverwaltungstools Redmine organisiert. Für die Betreuer wurde ein eigener Zugang zum Redmine-Projekt eingerichtet.

B.1. Milestones

Elaboration I2 - MS core prototypes

- Prototypen der Kernkomponenten, insbesondere der Komponenten mit hohen Risiken
  - WebRTC P2P Communication Prototype
  - SIP Server Connection Prototype
  - Addressbook Interface Implementation Prototype
  - Channel Interface Implementation Prototype (XHR Simple Queue Server)
- Risikoanalyse
- Architekturanalyse

Construction I2 - MS core functionality

- Implementation der Kernkomponenten (Prototyp-Komponenten-Elaboration)
  - P2P Communication
  - Channel Reference-Implementation
  - Addressbook Reference-Implementation
- Zwischenstand Projektdokumentation

Construction I4 - MS final app

- Implementation der Advanced Komponenten
  - Media Scaling
  - Scallable User Interface
- Projektdokumentation

Roadmap

Inception

09/21/2013

Material fetching, tool installations

100%

3 closed (100%) 0 open (0%)

Related issues

~~Tool #1~~: Redmine install & Config
~~Tool #29~~: Mobile Devices organisieren
~~Meeting #2~~: Kickoff Meeting

Elaboration I1

09/28/2013

Tutorial information, concept creation

100%

3 closed (100%) 0 open (0%)

Related issues

~~Planning #7~~: Arbeitspakete definieren
~~Planning #18~~: Projektplan Risiken
~~Planning #19~~: Projektplan Tools+Begründung

Elaboration I2 - MS core prototypes

10/12/2013

Core prototypes, risk evaluation

100%

10 closed (100%) 0 open (0%)

Related issues

~~Feature #5~~: WebRTC concept app
~~Feature #35~~: SIP Adapter Concept Implementation
~~Feature #38~~: WebSocket Prototyp
~~Support #31~~: Review Risiken
~~Documentation #39~~: Anforderungsanalyse
~~Information Collection #4~~: Einarbeitung WebRTC
~~Information Collection #34~~: SIP Protokoll Einarbeitung
~~Interface #6~~: Telefonbuch Schnittstelle Definieren
~~Planning #17~~: Projektplan Arbeitpakete/Meilensteine
~~Planning #30~~: Arbeitspakete Feinplanung

Construction I1

10/19/2013

App injection, implementation of core functionality from prototypes

100%

5 closed (100%) 0 open (0%)

Related issues

~~Feature #43~~: Umsetzung Kommunikation über SIP Proxy / SIP Server
~~Feature #46~~: Multikommunikation Konzept Prototype
~~Feature #53~~: WebRTC Demo objektorientiert umsetzen
~~Documentation #48~~: Rückblick Elaboration
~~Planning #41~~: Iterationsplanung

Construction I2 - MS core functionality

11/02/2013

Core functionality, testing, ui draft

100%

12 closed (100%) 0 open (0%)

Related issues

~~Feature #9~~: Addressbook Manager / Storage
~~Feature #10~~: Addressbook vcard adapter
~~Feature #21~~: User Interface Draft
~~Feature #45~~: Downsampling Prototype / Quality Scaling
~~Feature #55~~: Verschlüsselung
~~Feature #64~~: Core und Modell Test Coverage
~~Support #59~~: Code Review P2P Connection, Channel
~~Support #66~~: Refactoring
~~Tool #60~~: Installation SIP Server
~~Documentation #49~~: Rückblick Construction I1
~~Documentation #57~~: Architektur Begründungen (Big Picture)
~~Planning #54~~: Iterationsplanung

Construction I3

11/16/2013

Additional functionality, ui, core enhancement

100%

13 closed (100%) 0 open (0%)

Related issues

~~Bug #67~~: Get App running in Chrome
~~Bug #68~~: INVITE WebSocket termination or deregistration
~~Feature #22~~: User Interface Final Design
~~Feature #24~~: SIP Protocol Interface Interpreter / Responder
~~Feature #36~~: SIP Channel Implementation
~~Feature #62~~: Json Remote Addressbook Adapter
~~Feature #65~~: User management / storage
~~Feature #72~~: Send 'Bye' through RTCDataChannel
~~Support #71~~: Firewall Testing
~~Documentation #50~~: Rückblick Construction I2
~~Documentation #56~~: Systematische Performanceanalyse
~~Documentation #70~~: Schlussdokumentation Draft initialisieren
~~Planning #69~~: Iterationsplanung

Construction I4 - MS final app

11/30/2013

End of development

100%

16 closed (100%) 0 open (0%)

Related issues

~~Bug #74~~: Timeout wenn Verbindung nicht korrekt zu Stande kommt.
~~Feature #11~~: Addressbook csv adapter
~~Feature #23~~: Channel / Connection Manager
~~Feature #33~~: User Interface Final Design Implementation
~~Feature #63~~: internationalization
~~Feature #73~~: refactor strict declaration
~~Feature #77~~: Move SDP Utility Function to Service Class
~~Feature #78~~: Firewall P2P Test Mobile->Home
~~Feature #80~~: Refactor Namespacing
~~Feature #81~~: Refactoring (based on Review Feedback)
~~Documentation #44~~: WebRTC Tipps, Tricks und Fallstricke dokumentieren
~~Documentation #47~~: Qualitätsmanagement Doku (reviews, testing, ...)
~~Documentation #51~~: Rückblick Construction I3
~~Documentation #75~~: Gesamtdokumentation
~~Documentation #76~~: Review Documentation
~~Test #79~~: Test and Debug WebRTC implementation in Various Browsers

Transition

12/14/2013

Documentation, software packaging, required documents

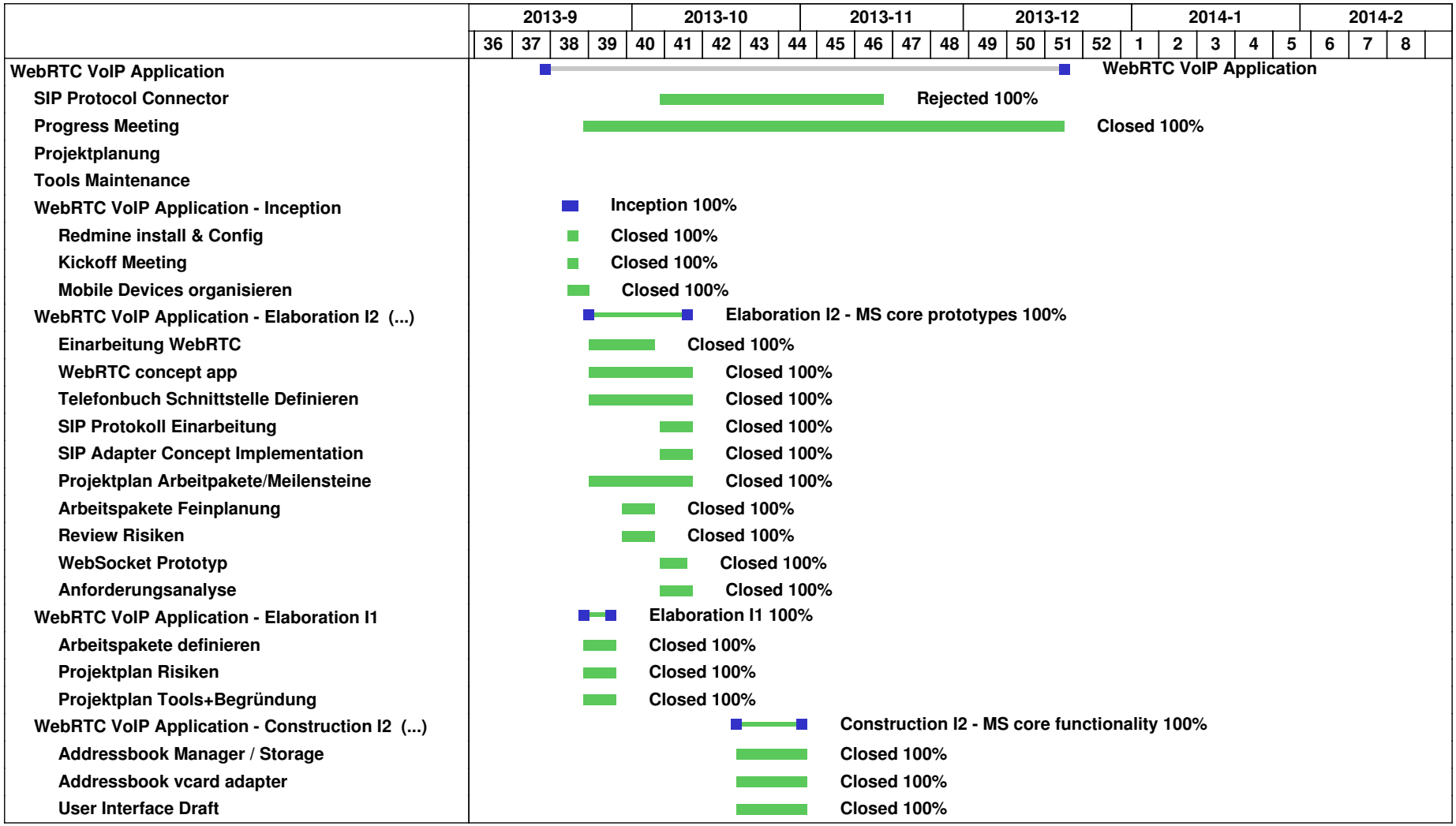
100%

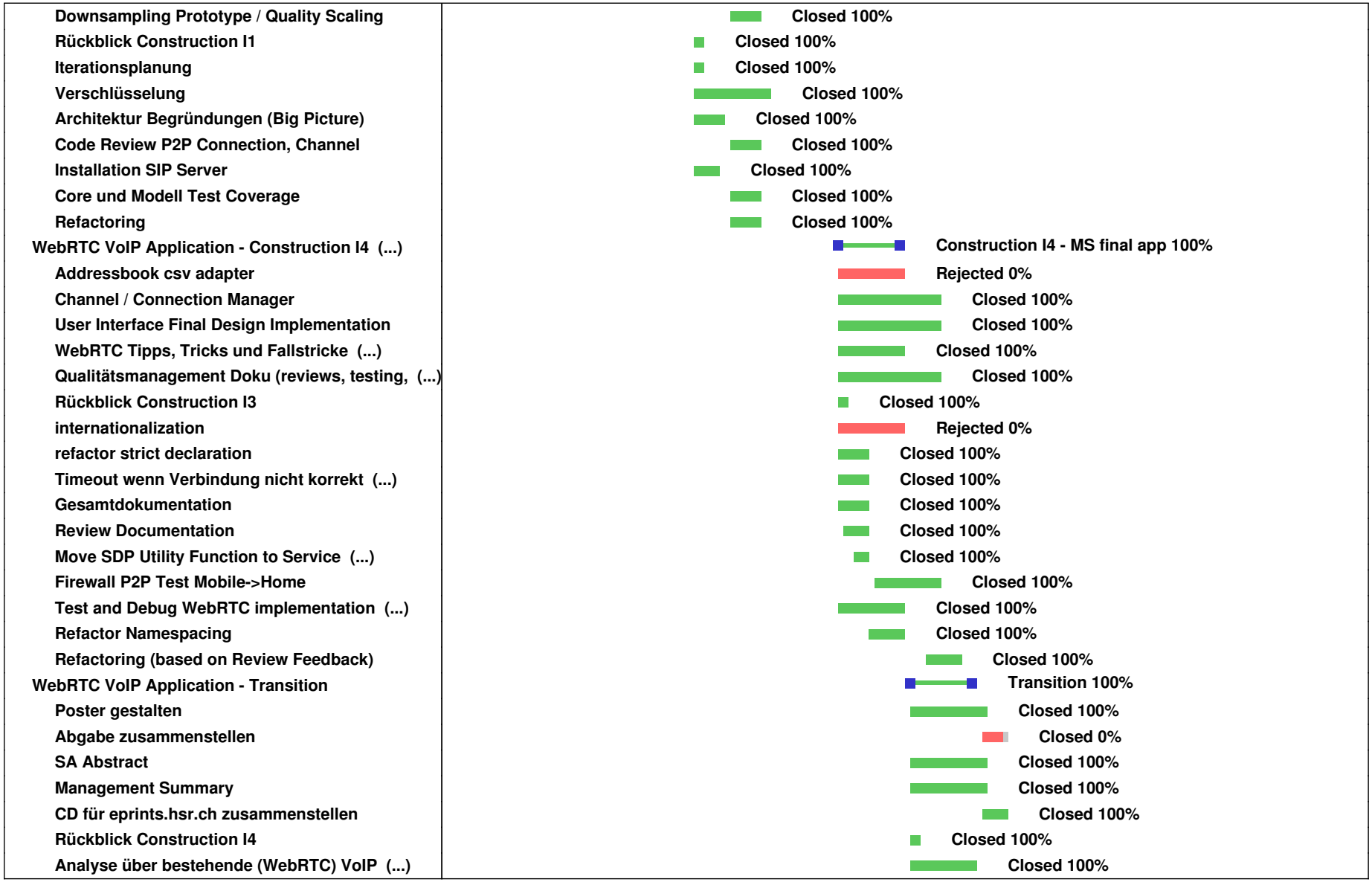
8 closed (100%) 0 open (0%)

Related issues

~~Documentation #13~~: Poster gestalten
~~Documentation #14~~: Abgabe zusammenstellen
~~Documentation #25~~: SA Abstract
~~Documentation #26~~: Management Summary
~~Documentation #27~~: CD für eprints.hsr.ch zusammenstellen
~~Documentation #52~~: Rückblick Construction I4
~~Documentation #58~~: Analyse über bestehende (WebRTC) VoIP Lösungen
~~Documentation #82~~: Finalize Doku

WebRTC VoIP Application




Downsampling Prototype / Quality Scaling
Closed 100%
Rückblick Construction I1
Closed 100%
Iterationsplanung
Closed 100%
Verschlüsselung
Closed 100%
Architektur Begründungen (Big Picture)
Closed 100%
Code Review P2P Connection, Channel
Closed 100%
Installation SIP Server
Closed 100%
Core und Modell Test Coverage
Closed 100%
Refactoring
Closed 100%
WebRTC VoIP Application - Construction I4 (...)
Construction I4 - MS final app 100%
Addressbook csv adapter
Rejected 0%
Channel / Connection Manager
Closed 100%
User Interface Final Design Implementation
Closed 100%
WebRTC Tipps, Tricks und Fallstricke (...)
Closed 100%
Qualitätsmanagement Doku (reviews, testing, (...)
Closed 100%
Rückblick Construction I3
Closed 100%
internationalization
Rejected 0%
refactor strict declaration
Closed 100%
Timeout wenn Verbindung nicht korrekt (...)
Closed 100%
Gesamtdokumentation
Closed 100%
Review Documentation
Closed 100%
Move SDP Utility Function to Service (...)
Closed 100%
Firewall P2P Test Mobile->Home
Closed 100%
Test and Debug WebRTC implementation (...)
Closed 100%
Refactor Namespacing
Closed 100%
Refactoring (based on Review Feedback)
Closed 100%
WebRTC VoIP Application - Transition
Transition 100%
Poster gestalten
Closed 100%
Abgabe zusammenstellen
Closed 0%
SA Abstract
Closed 100%
Management Summary
Closed 100%
CD für eprints.hsr.ch zusammenstellen
Closed 100%
Rückblick Construction I4
Closed 100%
Analyse über bestehende (WebRTC) VoIP (...)
Closed 100%

Finalize Doku	 Closed 100%
WebRTC VoIP Application - Construction I3	 Construction I3 100%
SIP Channel Implementation	 Rejected 0%
SIP Protocol Interface Interpreter / (...)	 Rejected 100%
INVITE WebSocket termination or (...)	 Rejected 30%
User Interface Final Design	 Closed 100%
Rückblick Construction I2	Closed 100%
Systematische Performanceanalyse	 Closed 100%
Json Remote Addressbook Adapter	 Closed 100%
User management / storage	 Closed 100%
Get App running in Chrome	 Closed 100%
Iterationsplanung	 Closed 100%
Schlussdokumentation Draft initialisieren	 Closed 100%
Firewall Testing	 Closed 100%
Send 'Bye' through RTCDataChannel	 Closed 100%
WebRTC VoIP Application - Construction I1	 Construction I1 100%
Umsetzung Kommunikation über SIP Proxy (...)	 Closed 100%
Iterationsplanung	Closed 100%
Multikommunikation Konzept Prototype	 Closed 100%
Rückblick Elaboration	Closed 100%
WebRTC Demo objektorientiert umsetzen	 Closed 100%

B.2. Aufgewendete Zeit

Activity	Member	2013-9	2013-10	2013-11	2013-12	Total
Design		1.5	1	2.5		5
	Tobias Blaser	1.5	1	2.5		5
Development			48.75	62.5	76.5	187.75
	Tobias Blaser		37.75	46.5	23.5	107.75
	Jannis Grimm		11	16	53	80
Projectmanagement		3.75	15.25	6	2	27
	Tobias Blaser	3.75	7.25	3.5	2	16.5
	Beat Gutzwiller		6			6
	Jannis Grimm		2	2.5		4.5
Inform		13.5	25.75	11	6	56.25
	Tobias Blaser	1.5	3.75			5.25
	Beat Gutzwiller	4	6			10
	Jannis Grimm	8	16	11	6	41
Experiment		7.75	53	23.5		84.25
	Tobias Blaser	7.75	30	2.5		40.25
	Jannis Grimm		23	21		44
Meeting		5.5	13.75	8.5	7.5	35.25
	Tobias Blaser	4	5.25	3.75	4.5	17.5
	Beat Gutzwiller	1.5				1.5
	Jannis Grimm		8.5	4.75	3	16.25
Configure		1.5	12	6		19.5
	Tobias Blaser	1.5				1.5
	Jannis Grimm		12	6		18
Documentation		3.75	4.75	20	69.5	98
	Tobias Blaser	3.75	4.75	14	43.5	66
	Jannis Grimm			6	26	32
Maintanance			1			1
	Tobias Blaser		1			1
Quality Management			4	18.25	4	26.25
	Tobias Blaser			3	2	5
	Jannis Grimm		4	15.25	2	21.25
Total		37.25	179.25	158.25	165.5	540.25

C. Infrastruktur

C.1. Hardware

- Persönliche Entwicklungsgeräte für jedes Teammitglied, bevorzugt Laptop (eigene Geräte)
- Zugewiesene Arbeitsplätze im Zimmer 1.258
- Mobile Devices mit WebRTC-fähigem Browser
- Virtual Server für Projektmanagement
- SIP Service (z. B. sipcall.ch), SIP-Server mit WebSockets-Support oder Webserver für XHR basierte Lösung

C.2. Tools

C.2.1. Projektmanagement

Das Projektmanagementtool Redmine bietet sich aus verschiedenen Gründen an:

- Redmine ist bekannt vom SE2-Projekt.
- Redmine lässt sich auf den Virtual Servern der HSR leicht installieren.
- Redmine bietet den benötigten Funktionumfang und ist einfach zu bedienen.
- Redmine bietet Git-Repository-Integration, falls dies gewünscht ist.

C.2.2. Versionsverwaltung

Git

Git ist ein bewährtes Versionsverwaltungstool, bietet den Vorteil von lokalen Repositories, ist sehr schlank und bringt eine gute Merge-Automatik mit.

GitHub

Mit GitHub besitzen die Studenten durch andere Projekte bereits Erfahrung. Als Studenten haben sie Zugriff auf kostenlose „Private-Repositories“. Zudem bietet GitHub noch zusätzliche Funktionen wie Wiki, RST- und MD-Viewer sowie Repository-Zugriff und Dateibearbeitung über ein Webinterface.

Backup

Ein zusätzliches Backup ist nicht notwendig, da durch die Versionierung mit Git die komplette Versionshistorie bei jedem Teilnehmer vorhanden ist. Somit ist das gesamte Projekt dreifach abgelegt (bei den Entwicklern sowie bei GitHub).

C.2.3. Dokumentation

Für grosse Dokumentationen und Abgabedokumente: LaTeX

LaTeX ist perfekt geeignet für grosse, gemeinsam zu erarbeitende Dokumente, weil die Source-Dateien über Git versioniert und gemergt werden können und wenig Platz verbrauchen. Zudem besteht ein sehr kleines Risiko auf Dokumentenverlust bzw. Dokumentenfehler durch die Software, weil LaTeX die Source-Dateien gar nicht verändert, im Unterschied zu einer Office-Applikation.

Für Notizen & Meetingprotokolle: Restructured Text (rst), txt, Markdown (md)

Für Notizen und kleine Dokumente reichen RST, TXT oder MD vollständig aus. Sie sind schlank, bieten nur das notwendigste, können versioniert und gemergt werden, weil es nur Textfiles sind, und werden von „GitHub Document aPreview“ unterstützt.

Für Diagramme, Skizzen: LibreOffice Draw (OpenDocument)

Wo es nicht anders geht, wird OpenDocument eingesetzt. Dabei wird berücksichtigt, dass es über Git nicht inkrementell versioniert und nicht gemerged werden kann.

C.2.4. Modeling

Als Modeling-Tool wird Astah gewählt, weil es das beste den Studenten bekannte Tool ist. Es deckt den geforderten Funktionsumfang grosszügig ab und bietet Image- sowie PDF-Export.

C.2.5. UI Drafting

- Evolus Pencil für UI Drafts
- eventuell LibreOffice Draw für UI Design Finals

C.2.6. Frameworks

Adapter.js

Adapter.js abstrahiert die verschiedenen Browserschnittstellen von WebRTC und vereinfacht die Entwicklung. Zudem muss bei einer Schnittstellenanpassung browserseitig nur der Adapter aktualisiert werden und nichts an der App geändert werden.

Angular.js

Angular.js ist ein bekanntes MVW- und Templating Framework, das eine saubere Trennung von Logik und Darstellung ermöglicht. Angular.js bindet darüber hinaus ViewModel Properties und Functions ans Template, wodurch sich Observerkonstrukte sparen lassen.

Require.js

Require.js soll zur Strukturierung und Autolading der Klassen und komponenten eingesetzt werden.

LESS

Less soll als clientseitiger CSS Parser eingesetzt werden, da es den CSS Code stark verschlankt und Vorteile wie Variablen und Mixins bietet.

jQuery

jQuery als eine der besten JavaScript-Bibliotheken, bietet sehr einfache Elementselektoren und viele Funktionen, die das Entwickeln vereinfachen.

C.2.7. Testing

Testing Framework Anforderungen:

- Testing mit realem Browser, Browsersimulationen unterstützen vermutlich WebRTC noch nicht
- Einfach einzubinden
- Einfach zu erweitern
- Bekannte Benutzung mit Tests und Asserts
- Möglichkeit zur Anbindung eines Build Tools

JsUnit / QUnit

JsUnit wie QUnit arbeiten mit einem realen Browser, sind einfach handzuhaben und bieten typische Assert-Syntax.

C.2.8. Building

Ein Build-Server wie Ant ist nicht nötig für dieses Projekt. JsUnit bietet zwar eine Anbindungsmöglichkeit. Für unsern Anwendungsfall und die nicht sehr komplexe Tool-Umgebung lohnt sich der Aufwand eines Build-Servers jedoch nicht.

C.2.9. Entwicklungsumgebung

Jeder Entwickler verwendet seine eigene bevorzugte Entwicklungsumgebung.

C.2.10. RunTime Environment

Als RunTime-Environment wird ein WebRTC kompatibler Browser (Firefox, Chrome(ium)) benötigt.

D. User Interface

D.1. User Interface Draft 1

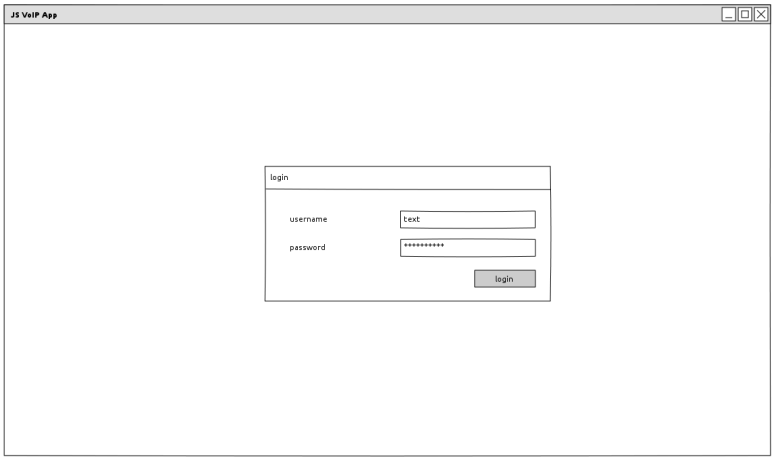


Abbildung D.1.: Über den Login Screen loggen sich Benutzer ein.

Abbildung D.2.: In der Hauptansicht hat der Benutzer Zugriff auf das Adressbuch.

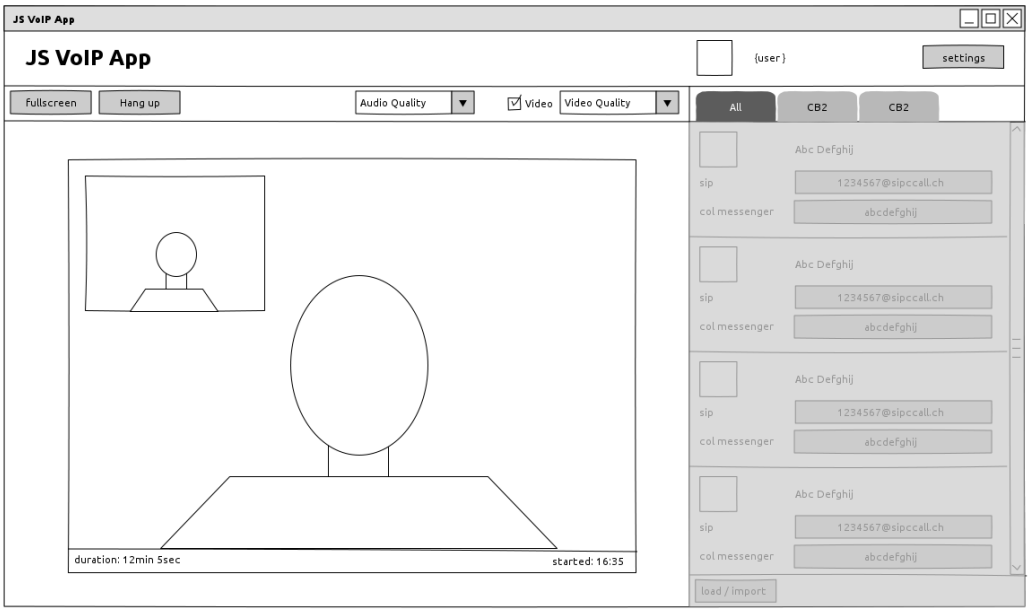


Abbildung D.3.: Ruft der Benutzer einen Kontakt an, so wird das Video in der Hauptansicht eingeblendet und die Kontakte werden inaktiv.

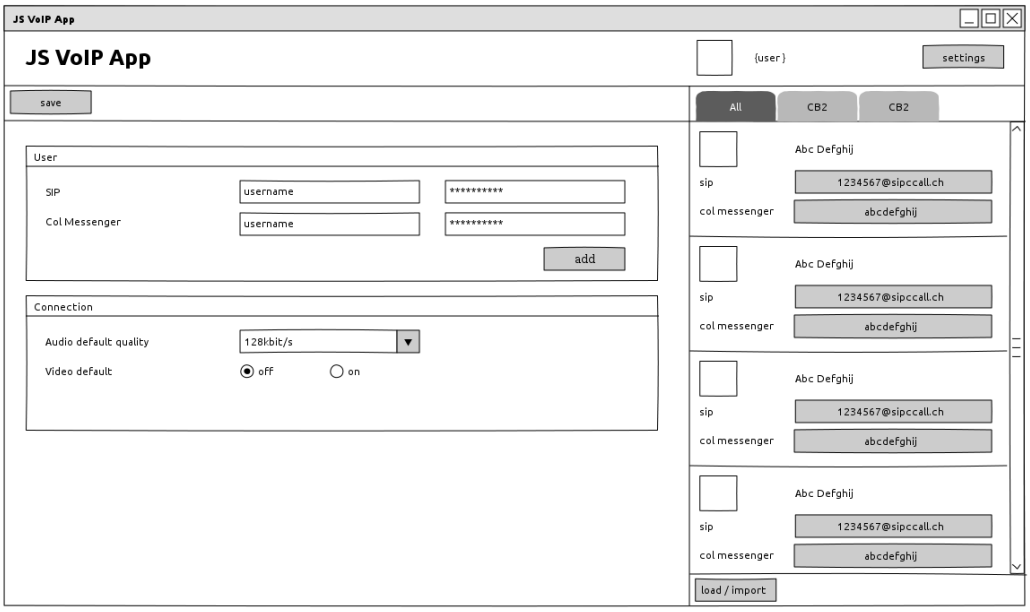


Abbildung D.4.: Über die Settings kann der Bentzer Einstellungen verändern.

D.2. User Interface Draft 2

Das UI2 folgt dem Prinzip „Mobile First“.

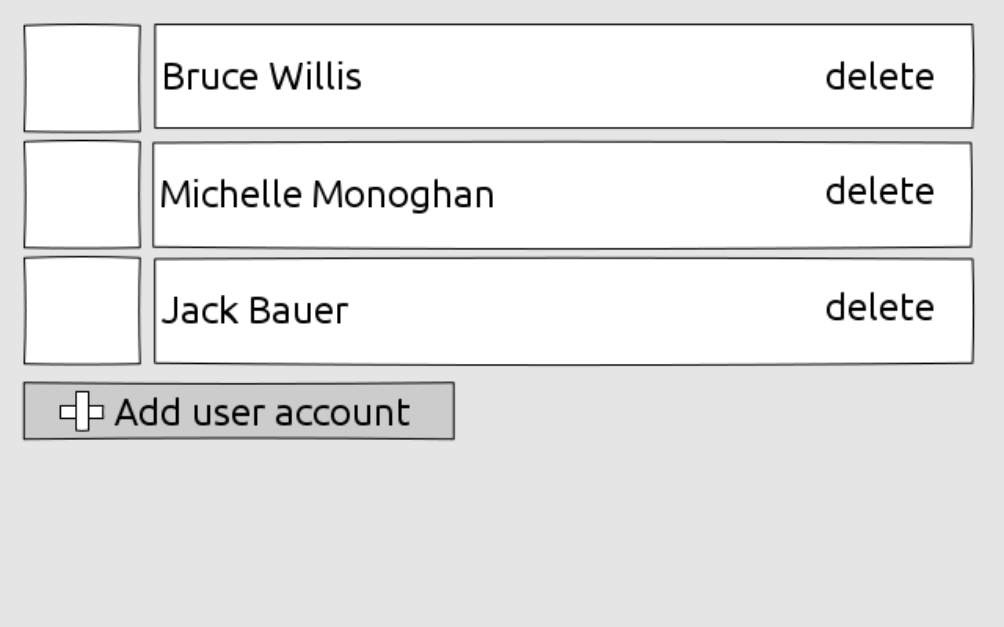


Abbildung D.5.: Benutzer Verwaltung und Login Screen. Durch klick auf einen Benutzer kann sich der Benutzer mit dessen Konto anmelden.

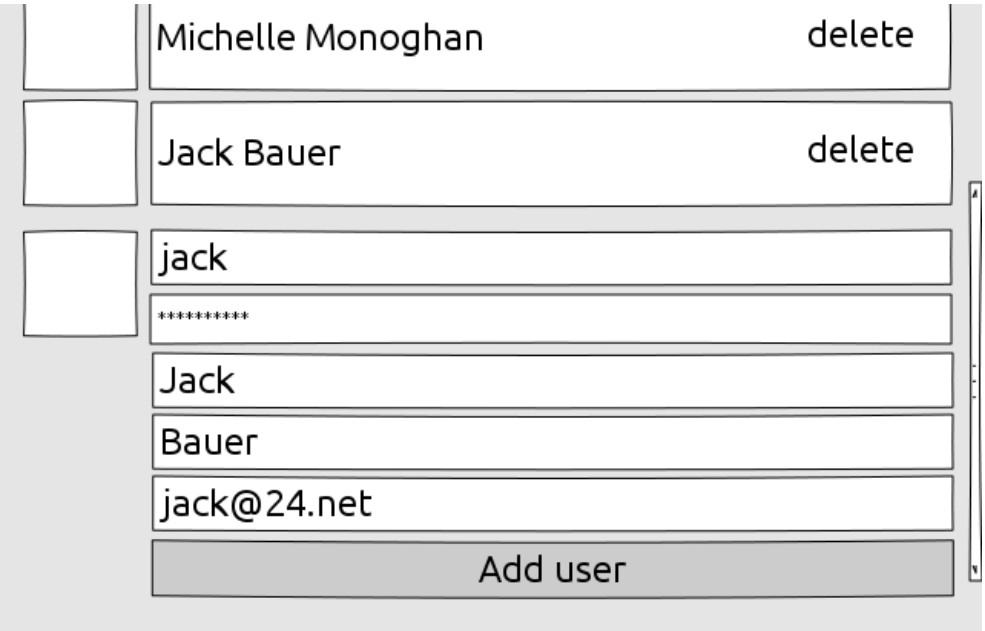


Abbildung D.6.: Die Benutzerverwaltung erlaubt es dem benutzer auch gleich einen neuen Bentzer zu erfassen.

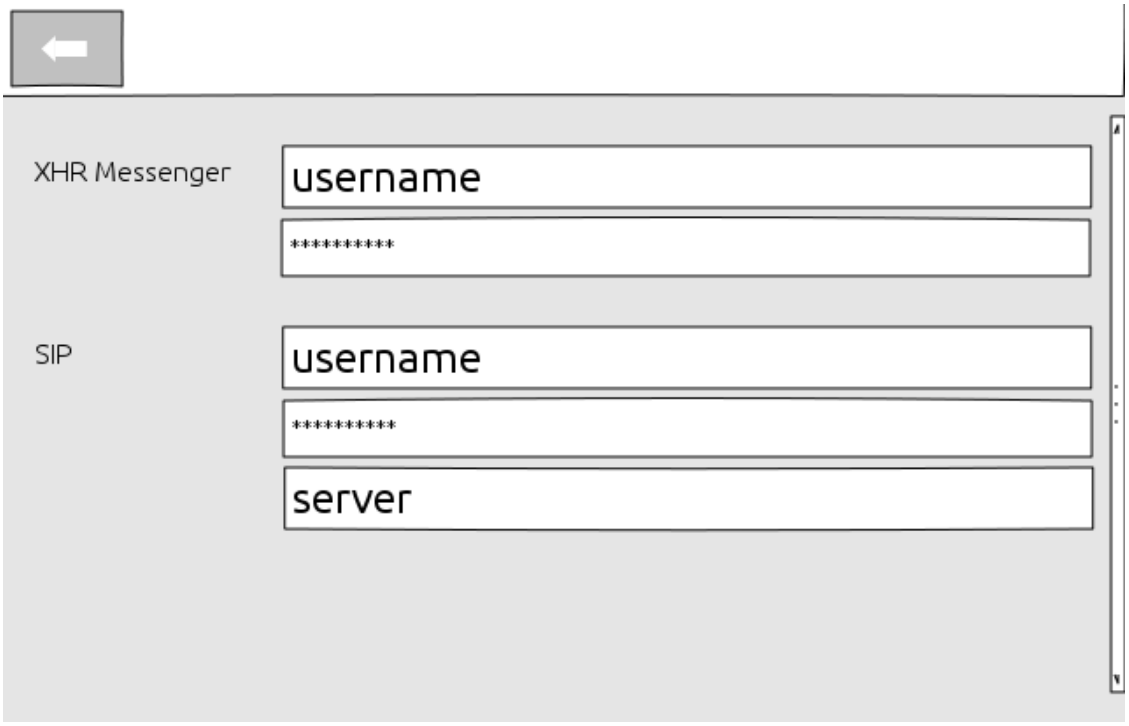


Abbildung D.7.: Die Bentzerverwaltung ermöglicht dem Benutzer das verwalten der verfügbaren Channel Accounts.

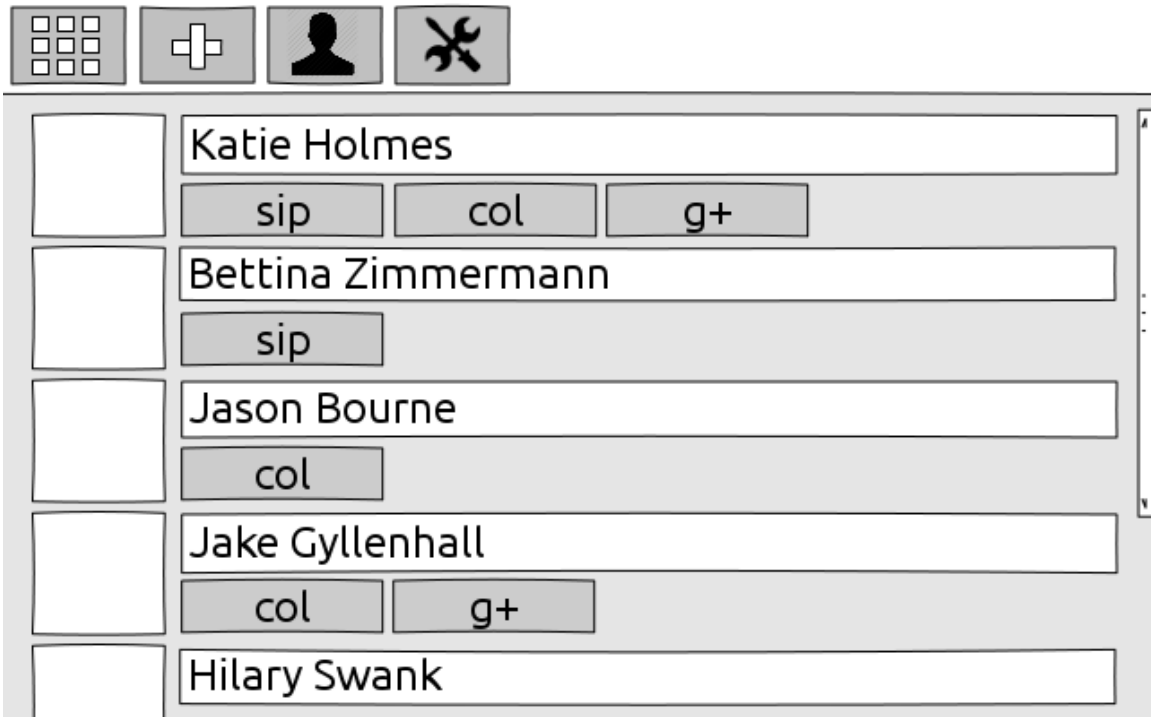


Abbildung D.8.: Adressbuch: Von hier aus ruft der Benutzer seine Kontakte an. Die verschiedenen Adressbücher sind über das Listensymbol erreichbar.

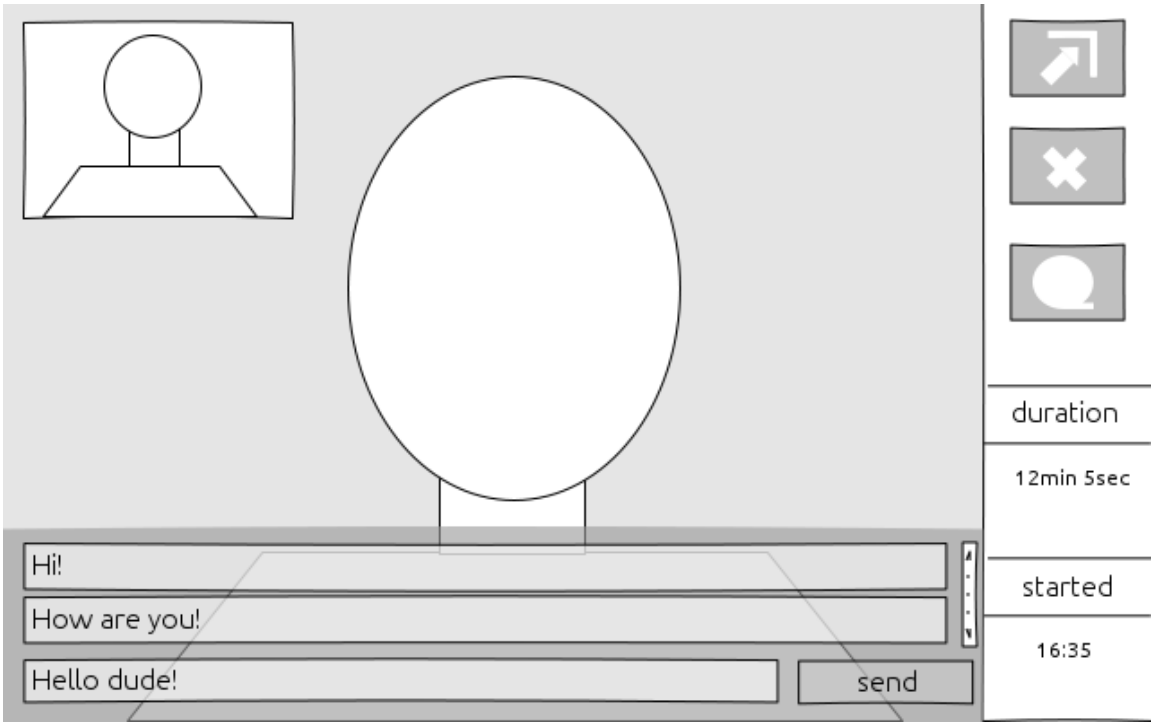


Abbildung D.9.: Phone View: Nebst dem Video sieht der Benutzer elementare Informationen über den Anruf.

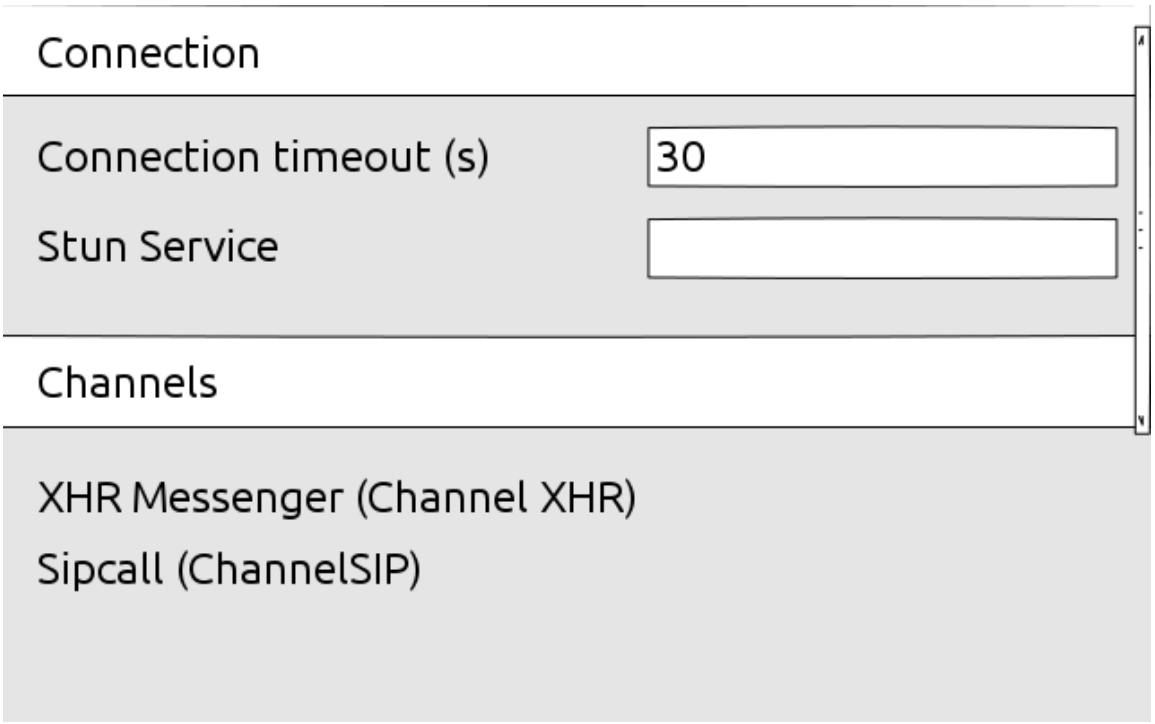


Abbildung D.10.: In den Settings kann der Benutzer Konfiguration und Channels einsehen.

D.3. Finales User Interface

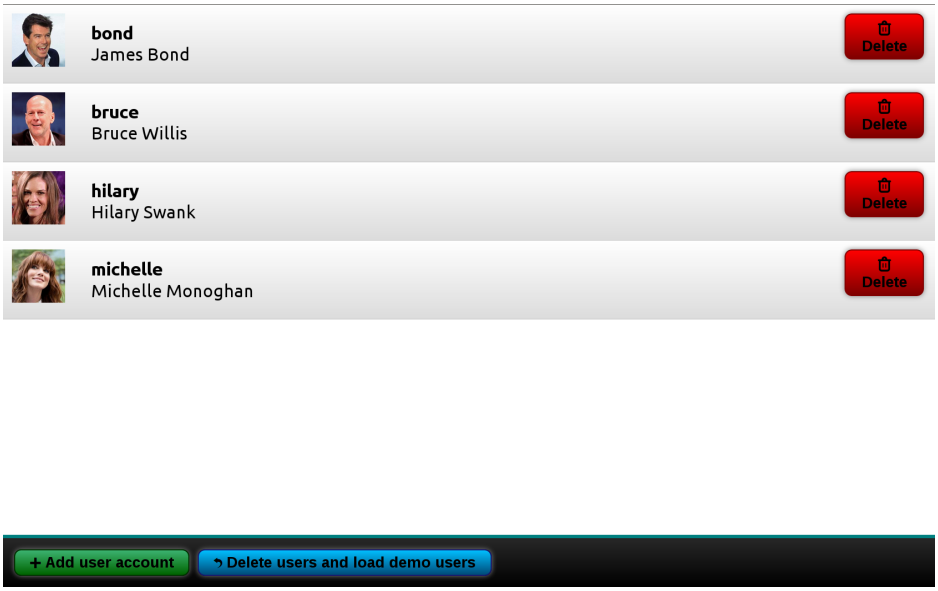


Abbildung D.11.: Benutzer Verwaltung und Login Screen. Durch klick auf einen Benutzer kann sich der Benutzer mit dessen Konto anmelden.
***Bildlizenz(en)**: cc by-sa 3.0 Gage Skidmore, cc by-sa 2.5 Rita Molnár, cc by-sa 3.0 Tabercil, cc by-sa 3.0 Manfred Werner*

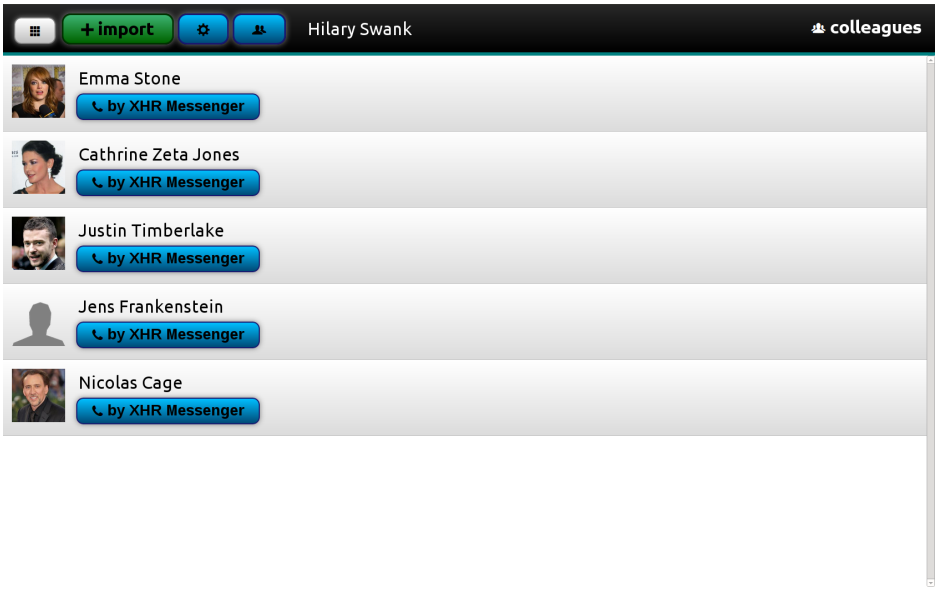


Abbildung D.12.: Adressbuch: Von hier aus ruft der Benutzer seine Kontakte an. Die verschiedenen Adressbücher sind über das Listensymbol erreichbar.
***Bildlizenz(en)**: cc by-sa 2.0 Gerald Geronimo, cc by 2.0 LeNair Xavier, modified by deerstop, cc by 3.0 Caroline Bonarde Ucci, cc by-sa 2.0 Nicolas Genin*

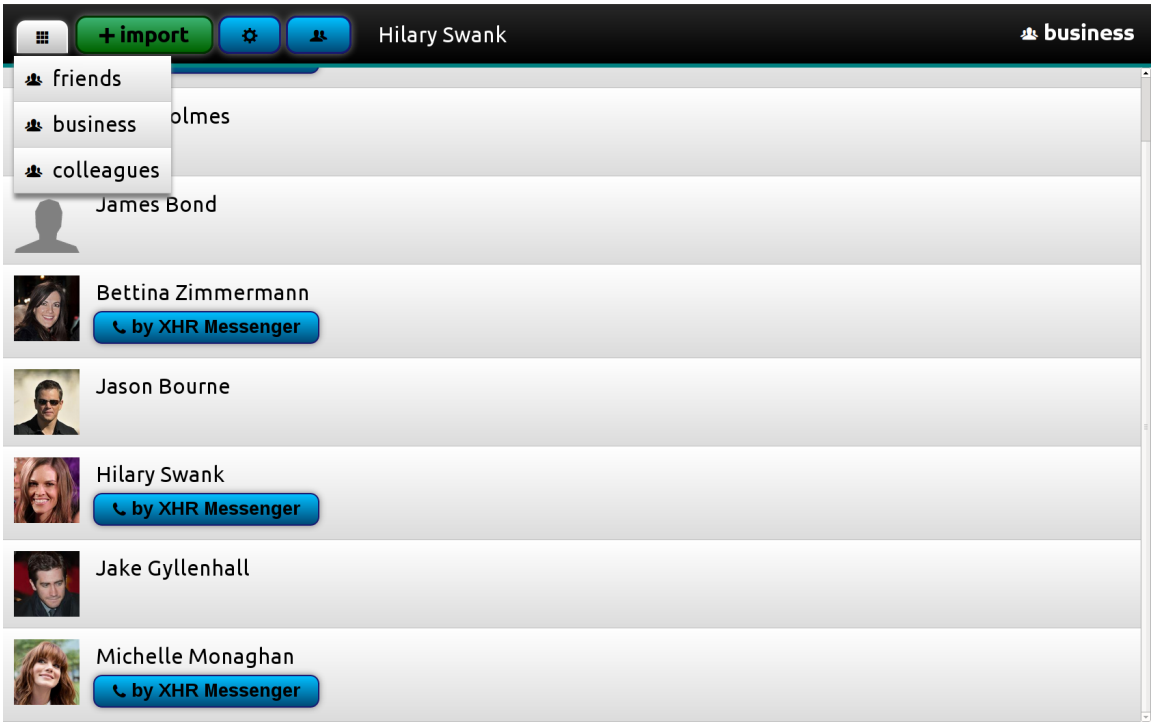


Abbildung D.13.: Auswahl des anzuzeigenden Adressbuches.
***Bildlizenz(en)**: cc by 3.0 Siebbi, cc by-sa 2.0 Nicolas Genin, modified by CherryX, cc by-sa 3.0 Manfred Werner, cc by 3.0 Siebbi, cc by-sa 3.0 Tabercil*

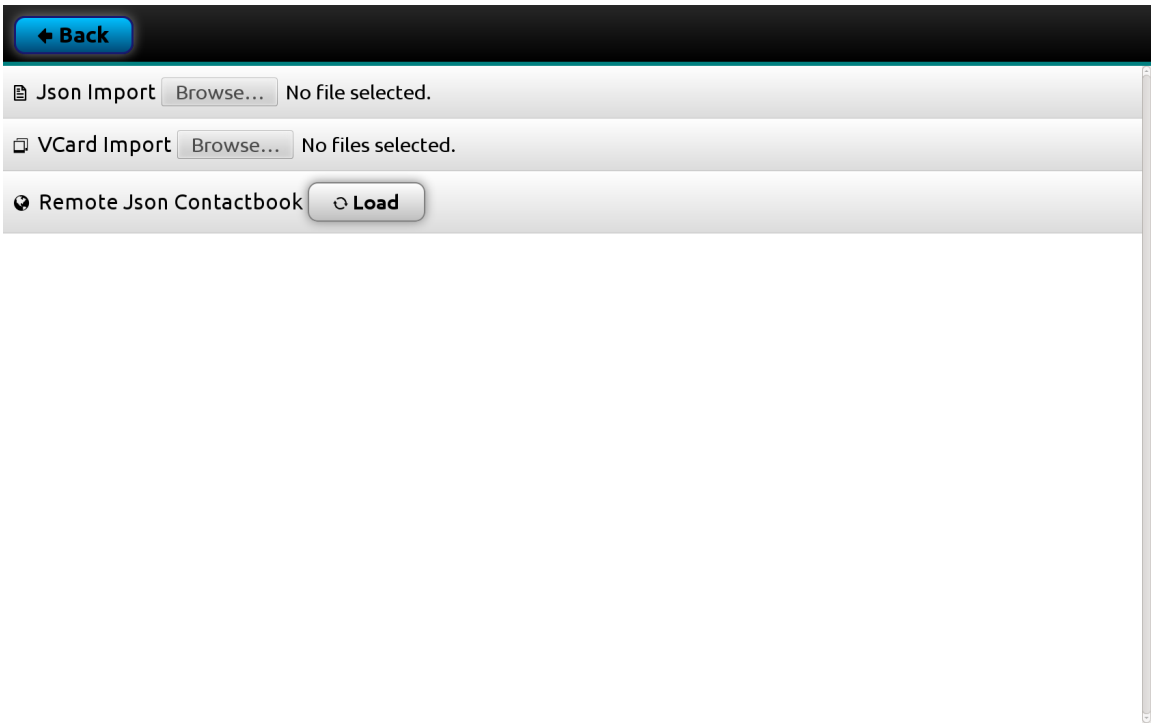


Abbildung D.14.: Adressbuch Import

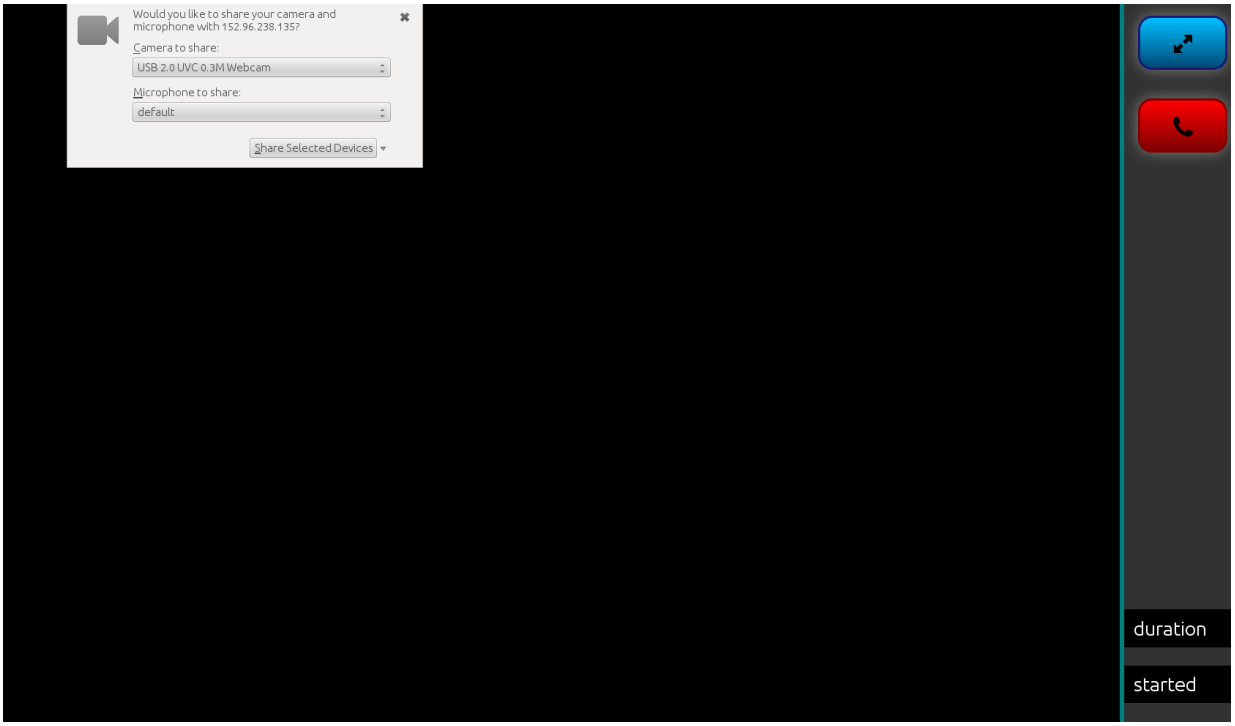


Abbildung D.15.: Anrufen: Kamerazugriff

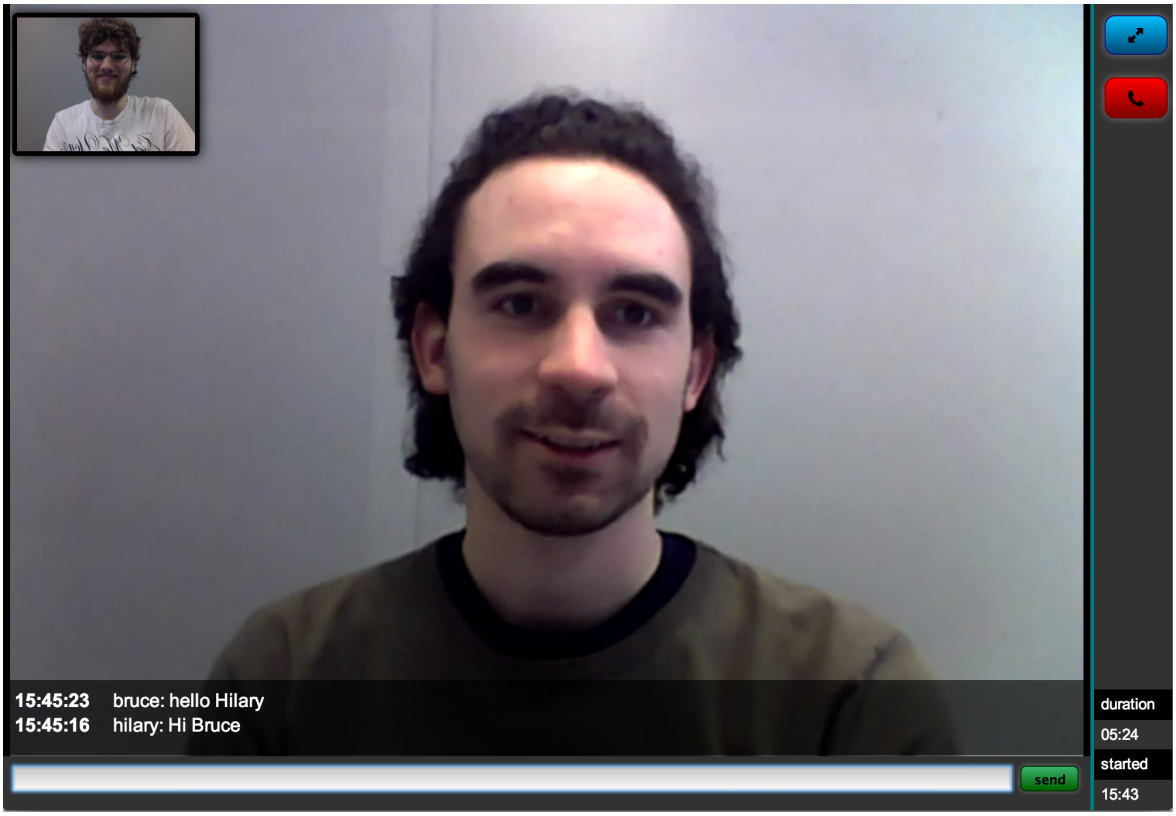


Abbildung D.16.: Anrufen

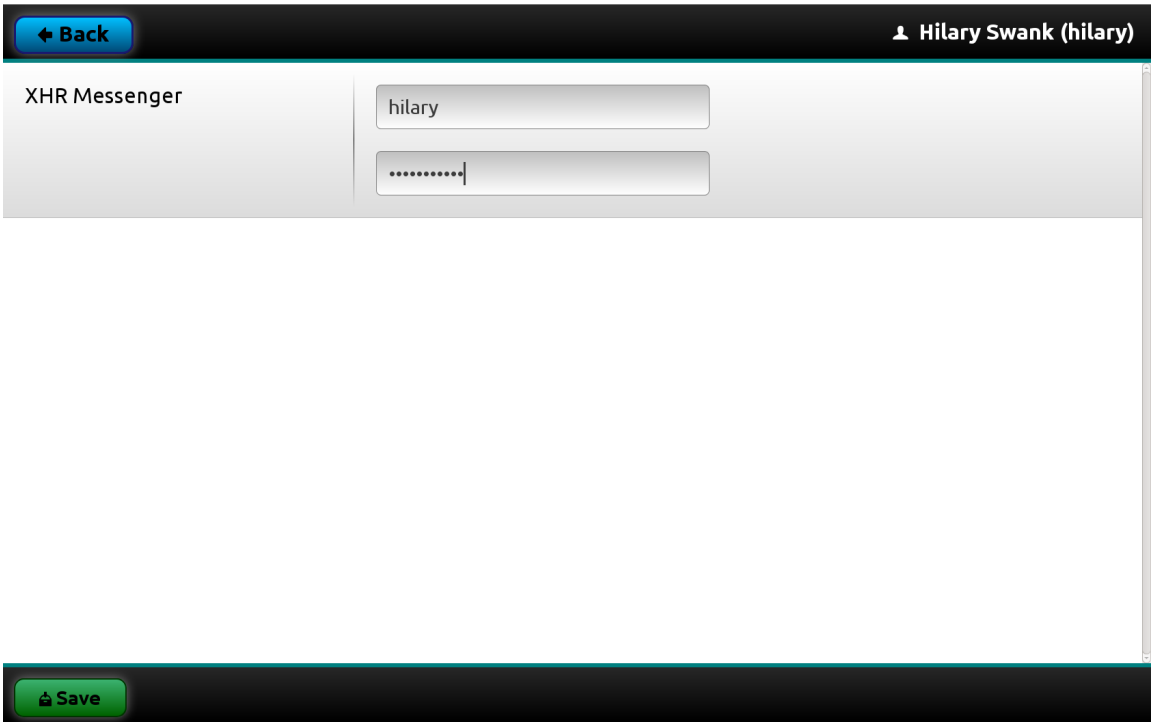
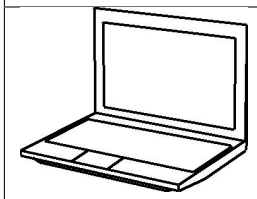


Abbildung D.17.: Account Setting: Verwaltung der Zugänge der aktiven Channels

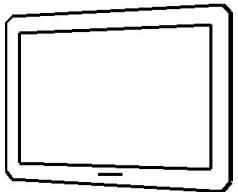
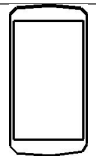
E. Performance-Analyse

E.1. Testgeräte

E.1.1. Laptops

Abbildung	Gerät	Hardware	Software
	Ultrabook, Asus UX 31	Intel Core i7 2x 1.8 GHz, 3.8GiB Memory	Ubuntu 12.04 64Bit, Browser: Firefox 25
	Netbook, Samsung NC 10	Intel Atom 1.6 GHz, 992MiB Memory	Ubuntu 12.10 32Bit, Browser: Firefox 23
	Mac Book Pro 2012	Intel Core i7 4x 2.3 GHz, 16GB Memory	Mac OS X 10.9 Mavericks, Firefox 24

E.1.2. Mobilgeräte

Abbildung	Gerät	Hardware	Software
	Tablet, Samsung Galaxy Tab 10.1	Nvidia Tegra 2x 1 GHz, 1GB Memory	Android 4.2.1 32 Bit, Firefox 25
	Smartphone, Google Nexus 4	NQualcomm Snapdragon S4 Pro 2x 1.5 GHz, 2GB Memory	Android 4.3 32 Bit, Firefox 25

E.2. Testszenarien und Ergebnisse

Die Tests wurden Bott-Up durchgeführt. Beginnend mit einem Verbindungstest zwischen zwei Browsertabs auf dem gleichen Rechner bis zu einem Test zwischen einem Client im Kabelnetz und einem Client im Mobilfunknetz.

E.2.1. Local Round

Geräte	Setup	Ergebnisse
Ultrabook - Ultrabook	• 1 Browser • Sender und Empfänger Instanz laufen jeweils in einem Browsertab • Datenverkehr macht roundtripp über Netzwerkkarte • Externe Services wie STUN Service und Signaling Channel	• Zunahme CPU Auslastung mit einem Channel: ca 20% • Zunahme Memory Verbrauch: ca 1-2% • Zunahme Netzwerk Traffic: keine da local loop • Video Qualität: flüssig, genügend Frames für angenehme Bewegungsdarstellung • Die Übertragung wird sauber beendet, keine weitere Leistungsaufnahme sobald der Stream beendet wird.

Abbildung E.1.: CPU Leistung für einen Stream

Abbildung E.2.: Vollständige Rückgabe der CPU an andere Prozesse nach dem Ende der Übertragung

E.2.2. Remote

Geräte	Setup	Ergebnisse
Ultrabook - Netbook	• 1 Ultrabook, 1 Netbook • Jeweils gleicher Browser • Datenverkehr läuft über HSR Wlan	**Netbook**: • Zunahme CPU Auslastung: ca 50% • Zunahme Memory Verbrauch: nicht spürbar • Zunahme Netzwerk Traffic: 10KiB/s out, 15KiB/s **Qualität**: • stockend, wenige Frames/s, unbrauchbar für Bewegungsdarstellung • Audio Qualität: unbrauchbar • Zunahme Netzwerk Traffic: 10KiB/s out, 15KiB/s

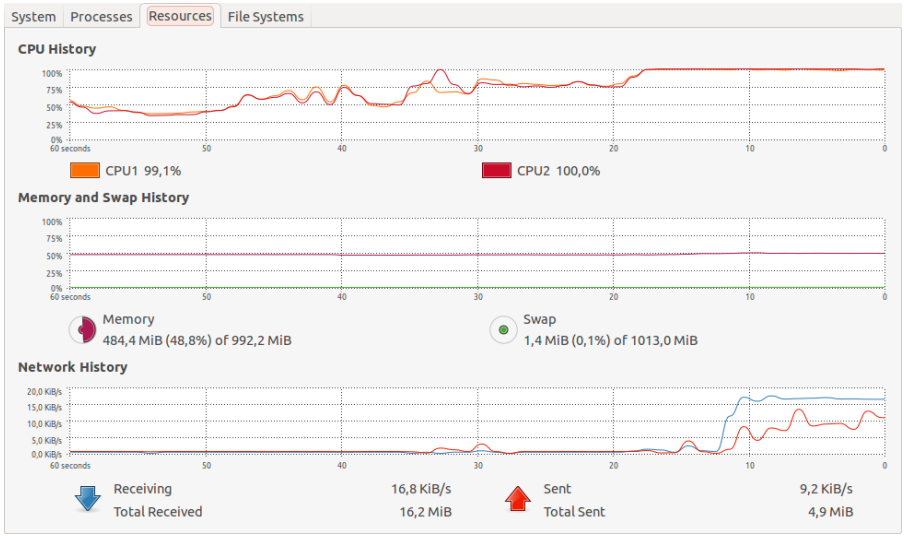


Abbildung E.3.: Das Netbook ist bis an die Leistungsgrenze ausgelastet

Geräte	Setup	Ergebnisse
Ultrabook - Netbook	• Gleich wie bei vorherigem Versuch • Nur Audio, keine Videoübertragung	**Netbook**: • Zunahme CPU Auslastung: ca 20% • Zunahme Memory Verbrauch: nicht spürbar • Zunahme Netzwerk Traffic: 7KiB/s in/out **Ultrabook**: • Zunahme CPU Auslastung: ca. 10% • Zunahme Memory Verbrauch: nicht spürbar • Zunahme Netzwerk Traffic: 8 KiB/s out, 7KiB/s in, Abbruch des Streams nach 30s

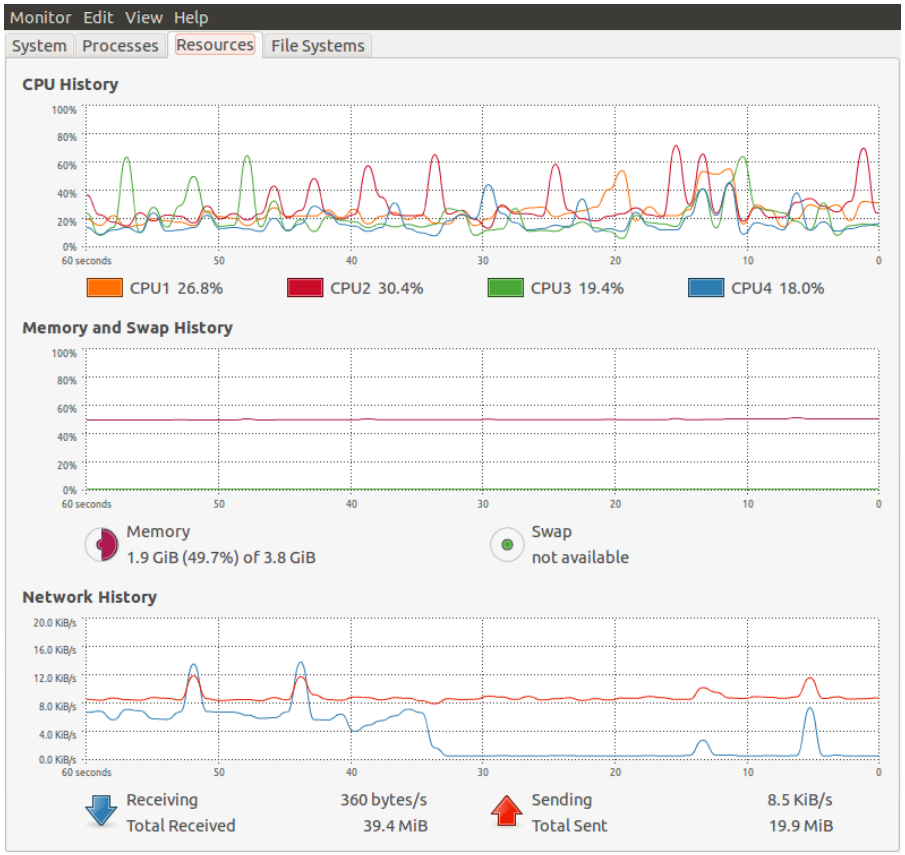


Abbildung E.4.: CPU Belastung Ultrabook

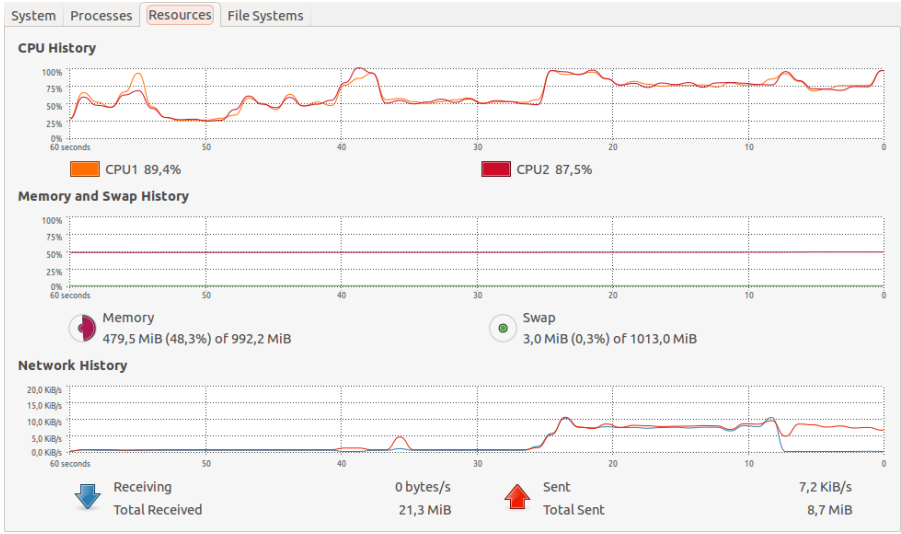


Abbildung E.5.: CPU Belastung Netbook

Geräte	Setup	Ergebnisse
Ultrabook - Mac Book	• 1 Ultrabook, 1 Macbook • Jeweils gleicher Browser • Datenverkehr läuft über HSR Wlan	**Ultrabook**: • Zunahme CPU Auslastung: ca 20% • Zunahme Memory Verbrauch: nicht spürbar • Zunahme Netzwerk Traffic: 50KiB/s, steigend bis 150KiB/s **Qualität**: • Flüssige Audio- und Video Übertragung in beide Richtungen • Stream des Macbook's sind keine Einzelbilder sichtbar • Stream des Ultrabooks zeigt bei schnellen Bewegungen Einzelbilder • Gut geeignet für Bewegungsdarstellung

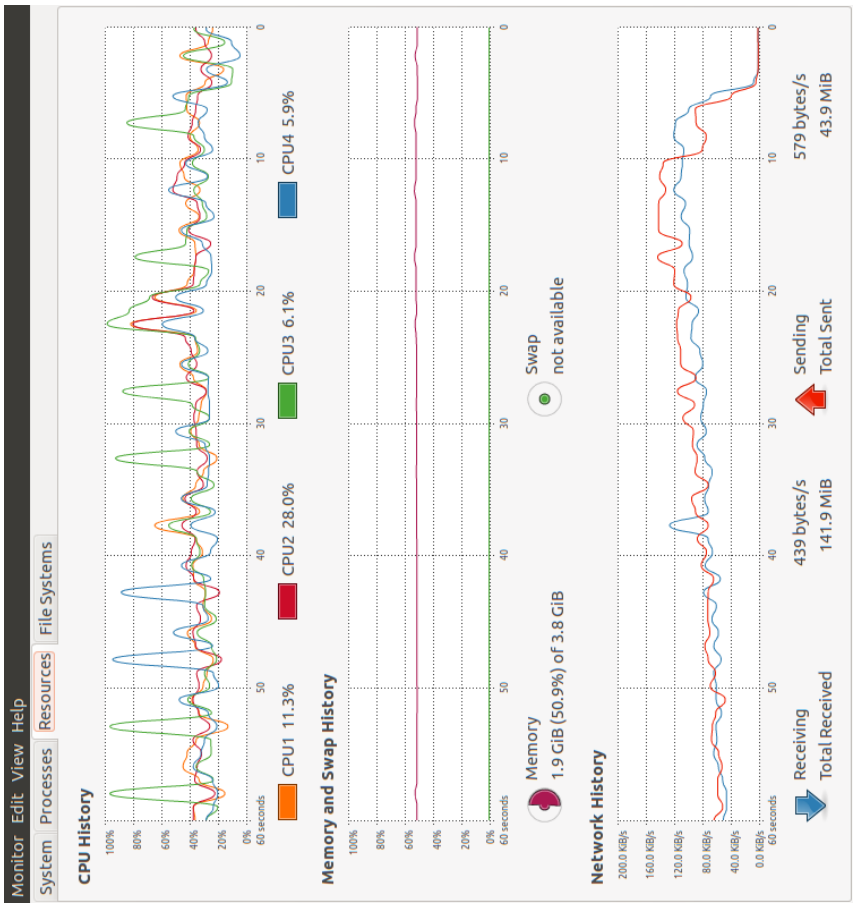


Abbildung E.6.: Ansteigende (automatisch skalierende) Datenrate, da beide Geräte entsprechende Videoauflösungen liefern können

E.2.3. Mobile - Desktop

Geräte	Setup	Ergebnisse
Ultrabook - Tablet	• 1 Ultrabook, 1 Tablet • Beide Geräte im HSR Wlan	**Tablet**: • Zunahme CPU Auslastung: ca 70% **Qualität**: • Tablet kann Video vom Desktop flüssig wiedergeben, auch der Ton wird korrekt und verständlich wiedergegeben • Tablet bringt Leistung nicht um eigenes Video parallel zum remote zu verarbeiten -> eigenes Video freezed • Desktop empfängt entsprechend vom Tablet nur ein Standbild

Geräte	Setup	Ergebnisse
Ultrabook - Smartphone	• 1 Ultrabook, 1 Smartphone • Beide Geräte im HSR Wlan	**Qualität**: • Video kann sowohl auf dem Phone wie auf dem Desktop einigermassen flüssig wiedergegeben werden • Die video Auflösung ist relativ gering • Geeignet für Bewegungsdarstellung

Geräte	Setup	Ergebnisse
Ultrabook - Smartphone	• 1 Ultrabook, 1 Smartphone • Gleiches Setup wie bei vorherigem Versuch	**Qualität**: • Wird das Smartphone gedreht, so verändert die Kamera die Auflösung und refreshed den Stream. • Der Stream wird jeweils neu aufgebaut. Dabei wird er auf 40KiB/s gedrosselt und langsam hochgefahren. • Dieses automatische Quality Scaling funktioniert nur, wenn das Gerät dies unterstützt. • Das Smartphone wird während der Kommunikation ziemlich warm und warnt bald vor schrumpfender Akkuleistung.

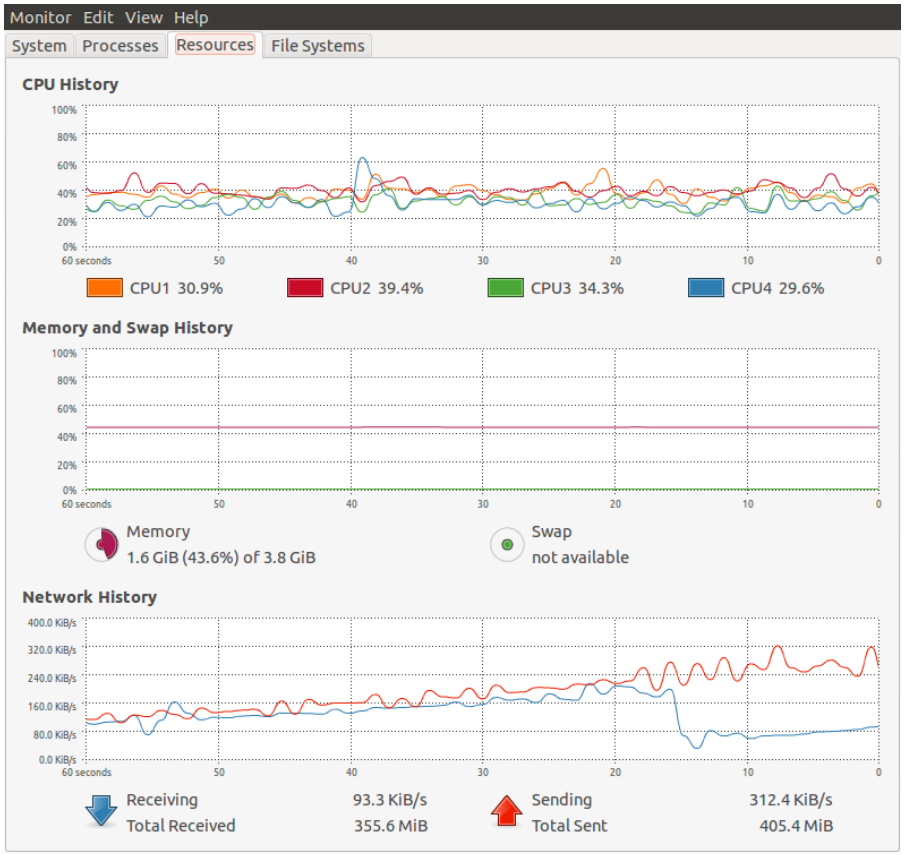


Abbildung E.7.: Zur Sekunde 25 wird das Smartphone von Widescreen Ausrichtung nach Portrait gedreht. Dabei wird die Datenrate auf das bereits mehrfach beobachtete minimum von 40KiB/s gedrosselt und anschliessend langsam hochgefahren.

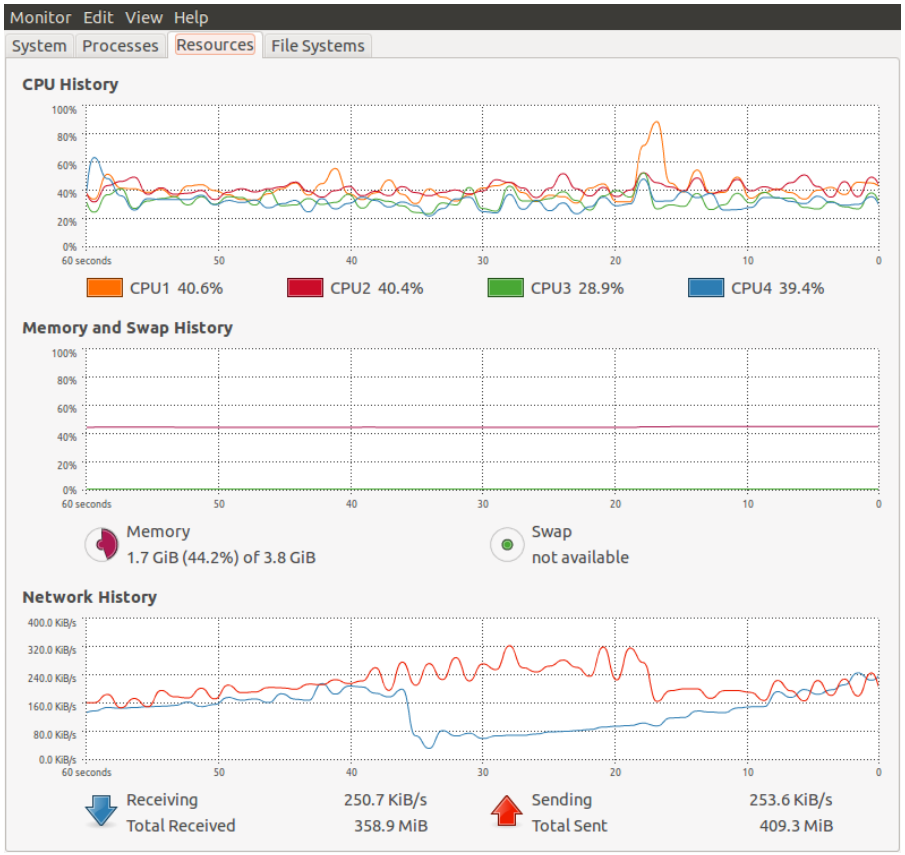


Abbildung E.8.: Die Datenrate wird bis auf 300KiB/s hochgefahren, wenn die Geräte dies liefern können. Zur 15. Sekunde findet eine Neuausrichtung des Smartphones statt.

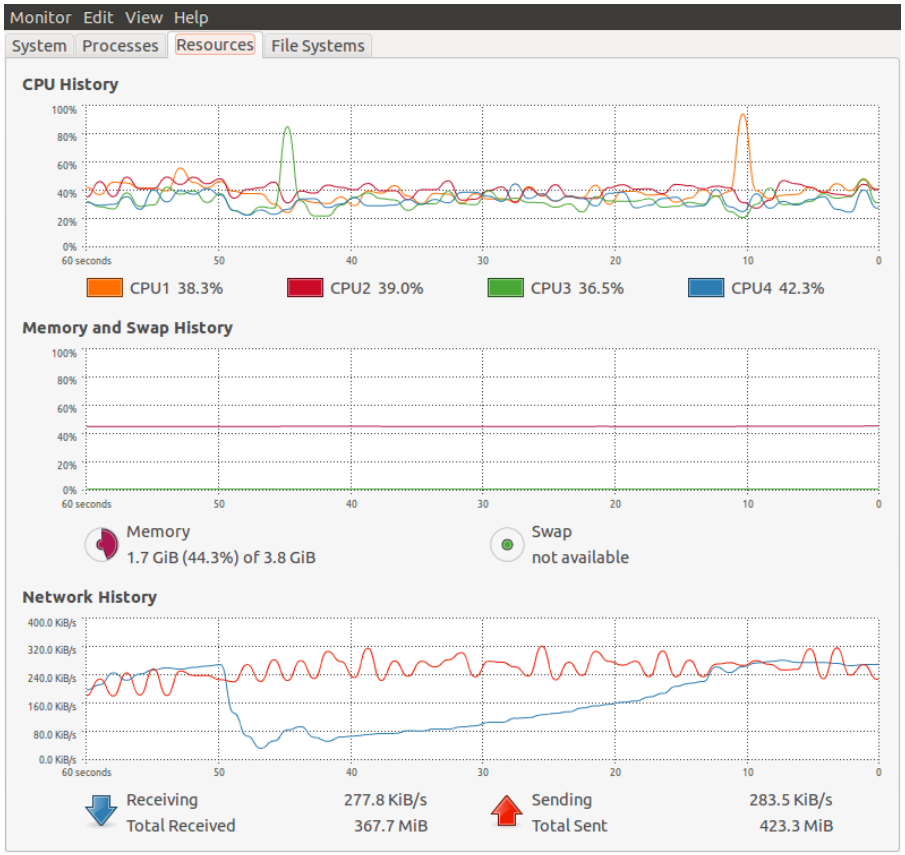


Abbildung E.9.: Drehung des Smartphone nach Widescreen. Nach 40s erreicht die Auflösung wieder die ursprüngliche Auflösung.

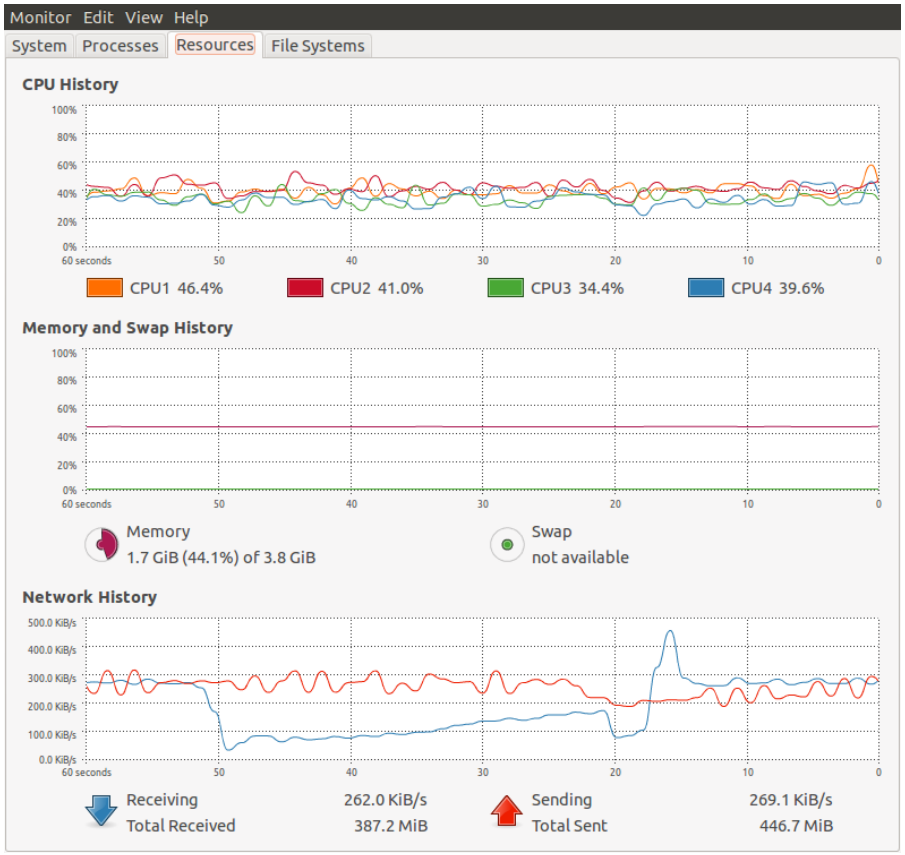


Abbildung E.10.: Ein Kurzer Pufferunterlauf zur Sekunde 40 führt daraufhin zu einem Peak. Anschliessend pendelt sich die Datenrate bei 300 KiB/s ein.

Abbildung E.11.: Das Ultrabook kann einige wenige KiB/s liefern, das Smartphone sogar nut 1KiB/s.

E.2.4. Out of Network

Geräte	Setup	Ergebnisse
Ultrabook - Smartphone	• 1 Ultrabook, 1 Smartphone • Smartphone direkt über Home Wlan angebunden • Ultrabook über Home Wlan mit HSR VPN angebunden	**Ultrabook**: • Zunahme CPU Auslastung: ca 20% • Zunahme Memory Verbrauch: nicht spürbar • Zunahme Netzwerk Traffic: 1-5KiB/s • Sprache gut unverständlich **Qualität**: • Video stockend oder flüssig, je nach dem ob die Pakete gut durchkommen oder nicht. • Smartphone wird etwas warm • Video Verzögerung von bis zu ca. 4s • Sprache verständlich oder unverständlich, je nach dem wie gut die Übertragungsrate ist.

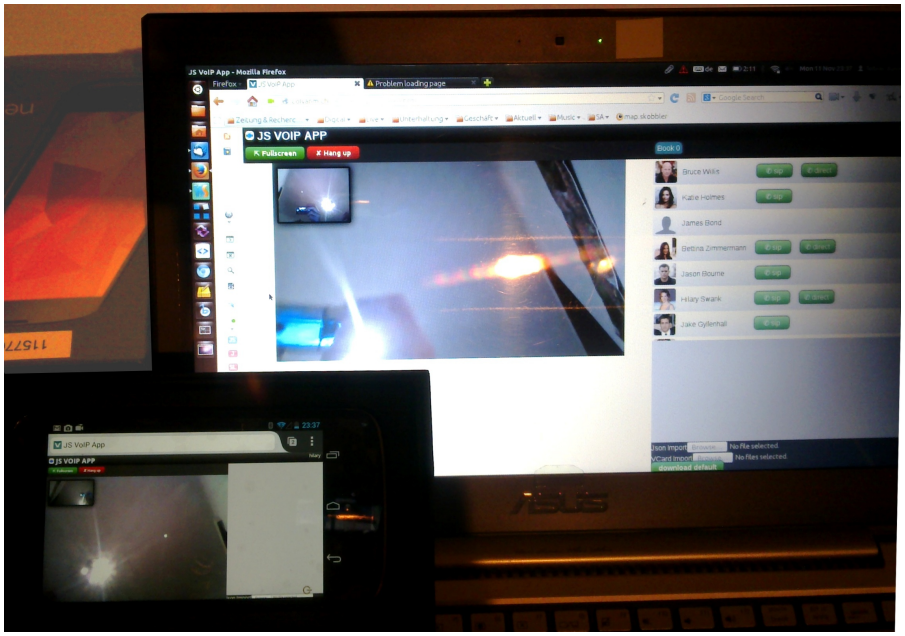


Abbildung E.12.: Videokommunikation zwischen Smartphone und Ultrabook

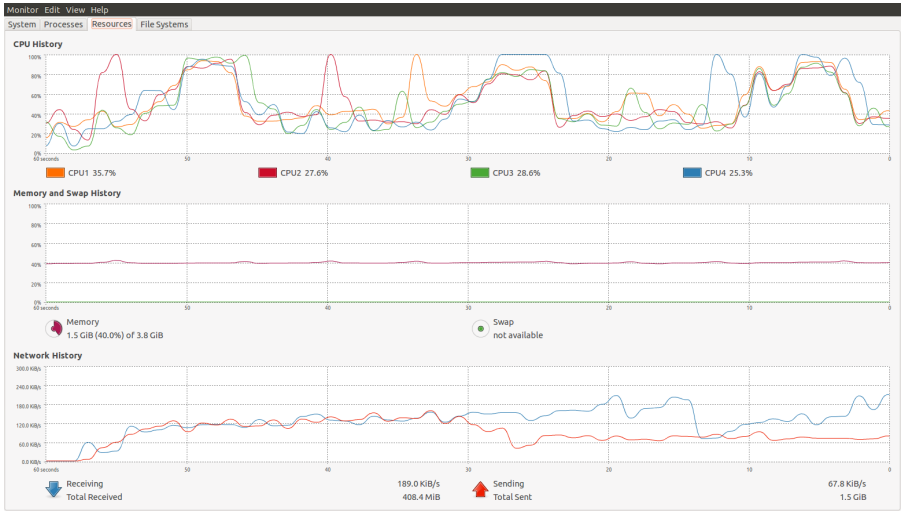


Abbildung E.13.: Die Datenrate fährt hoch bis über 100 KiB/s. Drehung des Smartphones zur Sekunde 55

E.2.5. Festnetz - Mobilfunknetz

Geräte	Setup	Ergebnisse
Ultrabook - Netbook	• 1 Ultrabook, 1 Netbook • Netbook über UMTS (Stadt) angebunden • Ultrabook über Home Wlan angebunden • 1. Mal Video+Audio, 2. Mal nur Audio	**Ultrabook**: • Zunahme Netzwerk Traffic bei Video: ca. 32-70KiB/s • Zunahme Netzwerk Traffic bei Audio: ca. 8KiB/s **Qualität (Video)**: • Video stockend • Audio verzögert und nur knapp verständlich **Qualität (nur Audio)**: • Audio flüssig, kaum verzögert und gut verständlich

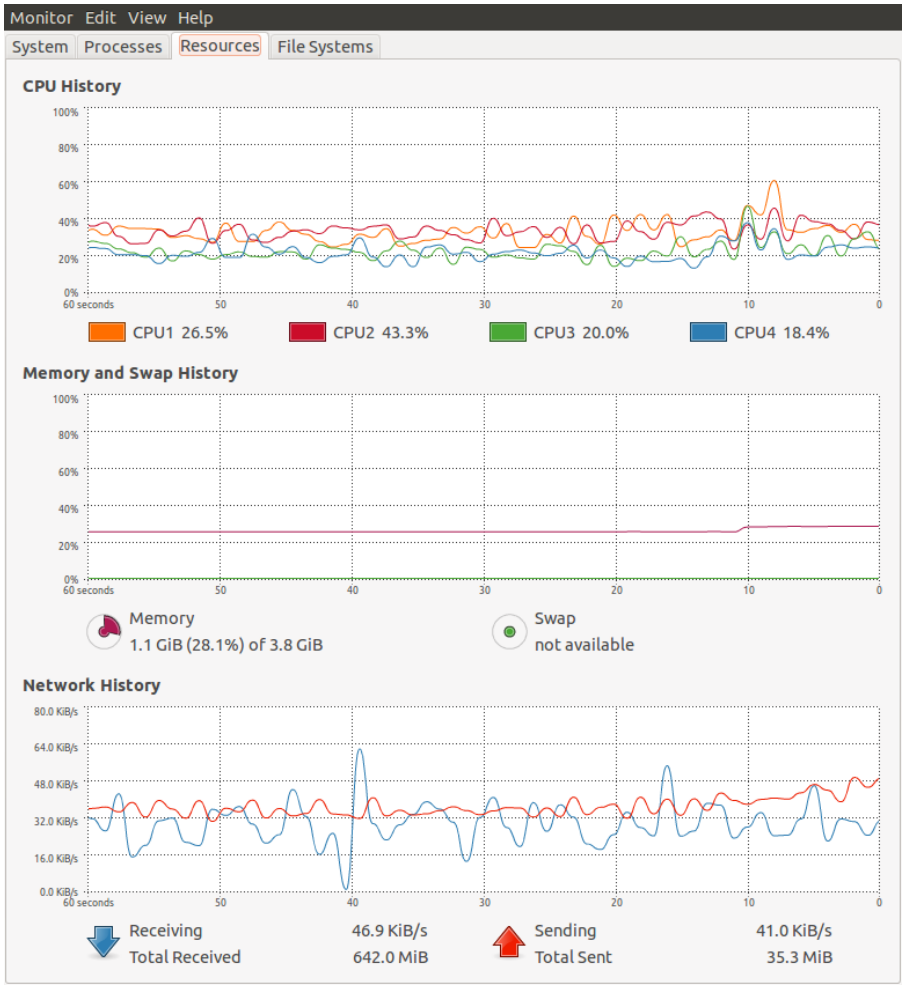


Abbildung E.14.: Ultrabook: Die Videowiedergabe ist nach dem Start noch relativ schmalbandig

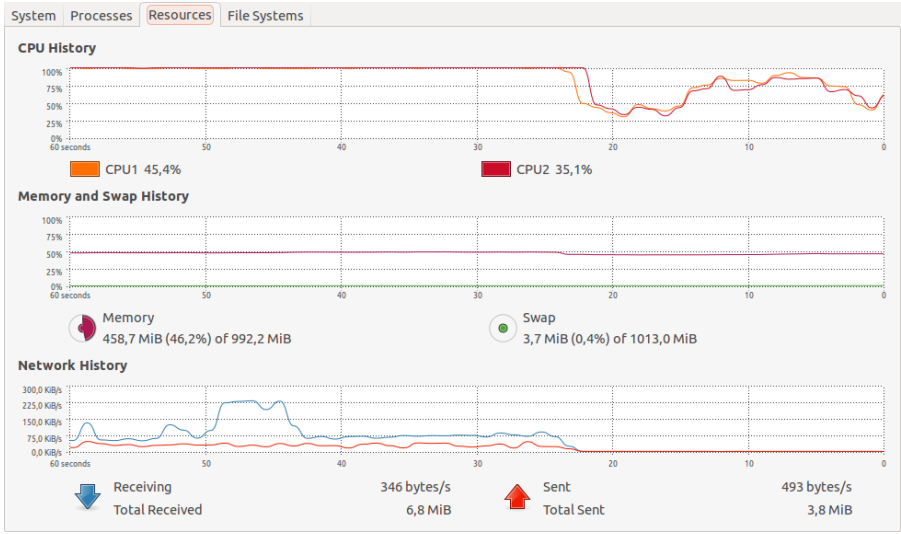


Abbildung E.15.: Netbook: Die Datenrate vom Ultrabook ist etwas höher als die des Netbook.

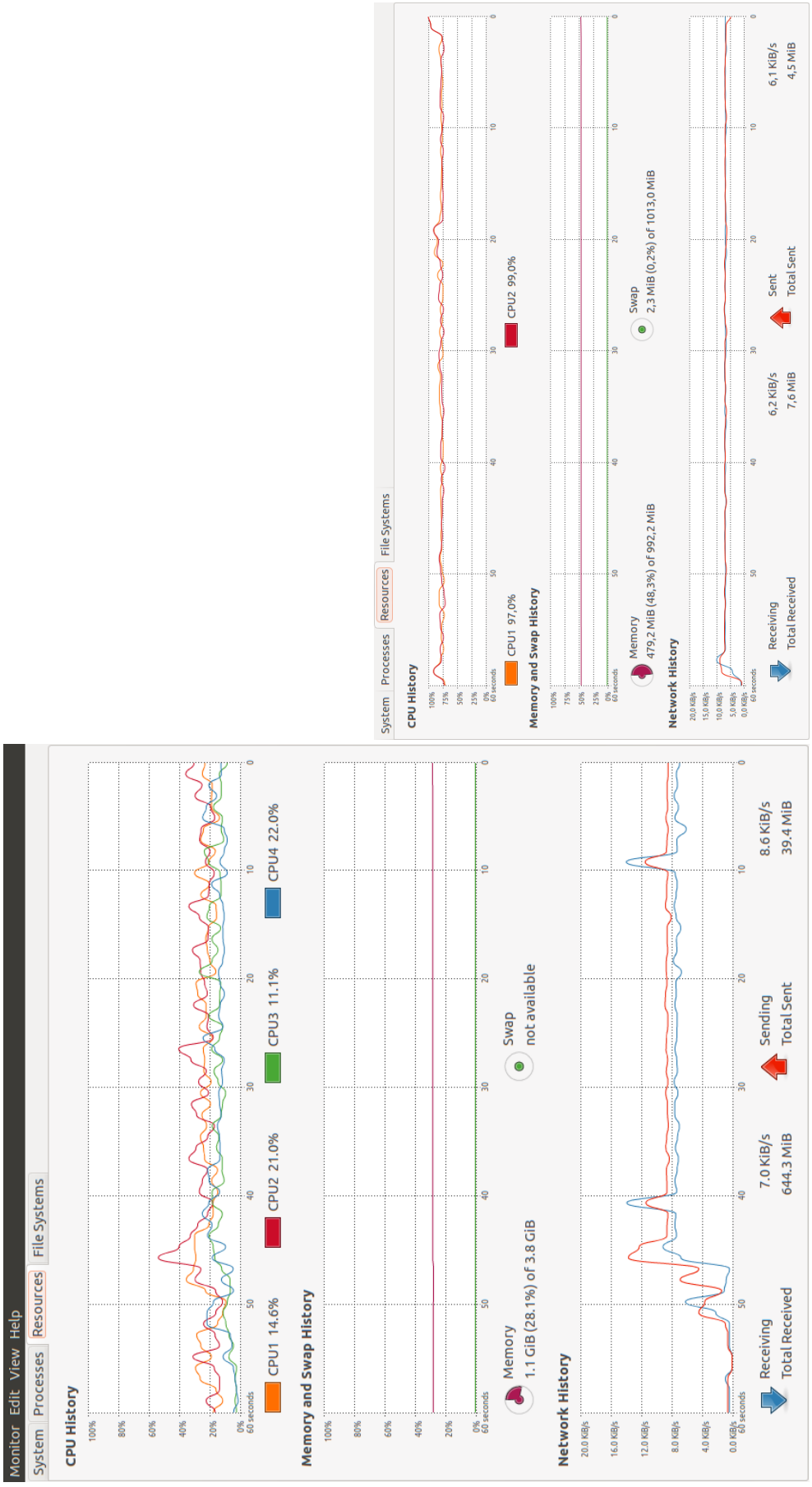
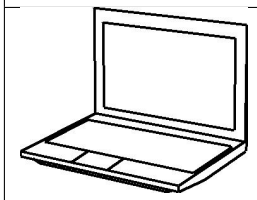
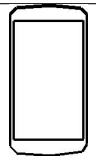


Abbildung E.16.: Ultrabook: Audiodatenrate nach dem Verbinden

Abbildung E.17.: Netbook: Audiodatenrate nach einiger Zeit (eingependelt)

F. Firewall Testing

F.1. Testgeräte

Abbildung	Gerät	Hardware	Software
	Ultrabook, Asus UX 31	Intel Core i7 2x 1.8 GHz, 3.8GiB Memory	Ubuntu 12.04 64Bit, Browser: Firefox 25
	Netbook, Samsung NC 10	Intel Atom 1.6 GHz, 992MiB Memory	Ubuntu 12.10 32Bit, Browser: Firefox 23
	Smartphone, Google Nexus 4	NQualcomm Snapdragon S4 Pro 2x 1.5 GHz, 2GB Memory	Android 4.3 32 Bit, Firefox 25

F.2. Testszenarien und Ergebnisse

F.2.1. VPN - Home-WLAN

Geräte	Setup	Ergebnisse
Home-WLAN - Home-WLAN-HSR-VPN	• Smartphone über Home-WLAN, Cablecom 10er-Leitung • Ultrabook über Home-WLAN, Cablecom 10er-Leitung, Verbunden mit HSR-VPN	• Media-Stream kommt ohne Verzögerung zustande • Kein Unterschied zu Locale Round

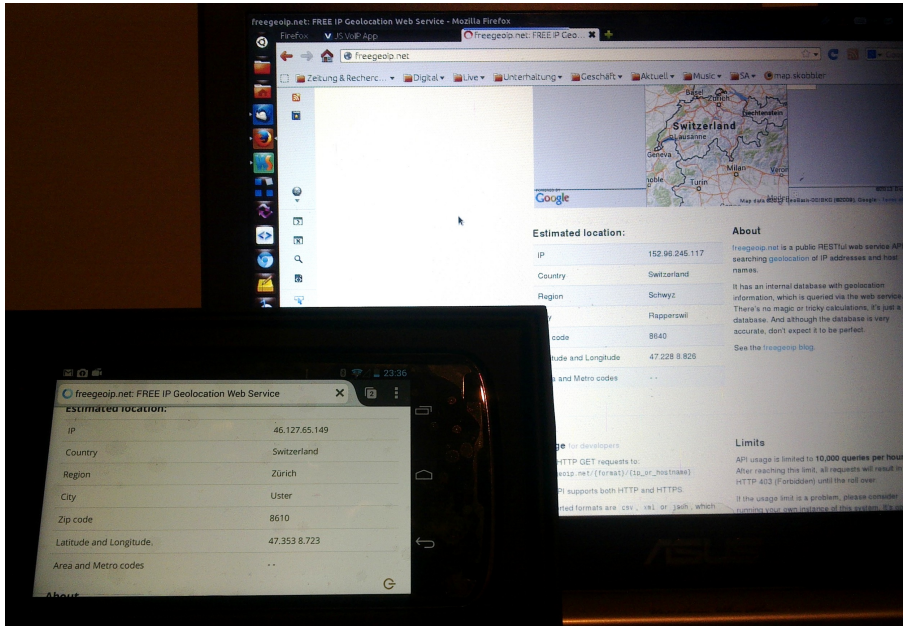


Abbildung F.1.: freegeoip.net zeigt, das das Ultrabook über die HSR verbunden ist und das Smartphone direkt im lokalen Wlan hängt.

F.2.2. HSR - Mobile

Geräte	Setup	Ergebnisse
Mobile Network - HSR-Notebook-WLAN	• Netbook über Mobilfunk, Sunrise Mobile Prepaid (max. 256 Kbps) • Ultrabook über HSR-Notebook-WLAN	• Media-Stream kommt nicht zustande • Im Wireshark sind keine UDP-Pakete sichtbar • Vermutung: HSR blockt „dynamic port“-„dynamic port“-Verbindungen um Filesharing zu unterbinden • Auch die apprtc.appspot.com-Demo-App kommt nicht durch

F.2.3. Home-WLAN - Mobile

Geräte	Setup	Ergebnisse
Mobile Network - Home-WLAN	• Netbook über UMTS (Stadt) • Ultrabook über Home-WLAN	• Media-Stream wird erfolgreich aufgebaut

G. Qualitätsmanagement

Zur Sicherung der Qualität wurden verschiedene Strategien kombiniert. Zur automatisierten Qualitätskontrolle wurden Unittests eingesetzt. Die Performance und das Verbindungsverhalten durch Firewalls hindurch wurde durch systematische Tests ermittelt. Die Funktion der Benutzeroberfläche wurde durch regelmässige manuelle Tests in Firefox und Chrome überprüft. Die Connection und die Videokommunikation wurden durch regelmässige manuelle Tests zu zweit mit zwei Geräten getestet.

Activity	2013-38	2013-39	2013-40	2013-41	2013-42	2013-43	2013-44	2013-45	2013-46	2013-47	Total
Design		0.50	1.00	1.00				1.00	0.50	1.00	5.00
Development				1.00	20.50	15.25	15.50	13.50	12.50		78.25
Projectmanagement	1.00	1.50	7.25	1.75	3.75	2.50	1.25	3.50		1.50	24.00
Inform	1.50	12.00	6.00	5.75	6.00	4.00	8.00	3.00			46.25
Experiment		5.25	16.50	10.50	13.50	3.00	16.00	4.00	12.00	1.50	82.25
Meeting	2.50	2.00	6.50	4.00	1.75	1.50	1.00	2.00	1.50		22.75
Configure	1.50				9.00	3.00		4.00			17.50
Documentation	0.50	3.00	0.25	1.00		1.50	2.25	1.00	2.00		11.50
Maintanance					1.00						1.00
Quality Management					3.00	1.00	5.50	3.00			12.50
Total	7.00	24.25	37.50	25.00	58.50	31.75	49.50	35.00	28.50	4.00	301.00

Abbildung G.1.: Die Aktivität „Qualitätsmanagement“ fasst Testing und Reviewing zusammen.

G.1. Automatisierte Tests

Abbildung G.2.: mit QUnit erfolgreich ausgeführte Tests

Unittests Die Applikationsdomain wird mit Unittests getestet und die fortbleibende Funktion bei Änderungen damit nachgewiesen.

G.2. Systematische Tests

Performancetest Die Performance wurde durch eine umfangreiche Performanceanalyse untersucht und dokumentiert. Dabei wurden Mobilgeräte wie Desktopgeräte getestet.

Siehe Anhang E

Firewall Tests Das Verhalten der durch WebRTC genutzten Protokolle durch Firewalls hindurch wurde in einem dreiteiligen Firewall-Test ermittelt. Dabei wurde das Verhalten der HSR Firewall, von Firewalls in Mobilfunk- und von Firewalls in Heimnetzanbindungen untersucht.

Siehe Anhang F

G.3. Manuelle Tests

Connection & Videoverbindung Die Connection stellt die am schwierigsten zu testende Komponente und gleichfalls eine der komplexeren dar, weil sie die RTCPeerConnection, die RTCDataChannel und die UserMedia Schnittstelle verwendet. Die Schnittstellen interagieren mit dem User und mit Netzwerkevents. Testen mit Unittests wäre nur möglich gewesen, wenn für diese Schnittstellen komplette Dummies gebaut worden wären.

Aus diesem Grund wurde als Teststrategie manuelles Testen gewählt. Dabei wurden Verbindungsaufbau und -abbau jeweils in jeder Richtung mit Firefox, Chrome und Firefox-Chrome kombiniert sowie allen Variationen von Auf- und Abbaureihenfolgen zu zweit mit zwei Geräten durchgetestet.

Dies wurde in jeder Iteration mindestens einmal durchgeführt um die Funktionalität zu überprüfen.

Kontaktbuch Import Die Verarbeitung der Kontaktbuchdaten und das Kontaktbuchmanagement haben wir durch Unittests abgedeckt. Beim Import hätten wir ebenfalls einen Dummy bauen müssen. Daher wurde auch diese manuel getestet.

G.4. Reviews

Regelmässige Reviews stellten die Qualität von Dokumentation und Code sicher. Für die Reviews wurde die Commit-Kommentarfunktion von GitHub genutzt.

Zusätzlich hat uns Herr Bläser ein Code Review durchgeführt.

H. Schnittstellen & Protokolle

Channel Beschreibt die Channelschnittstelle. Implementatoren eines eigenen Channels müssen sich an diese Schnittstelle halten sowie den Channel in der appConfiguration eintragen.

Contactbook Beschreibt die Contactbookschnittstelle. Implementatoren eines eigenen Contactbooks müssen sich an diese Schnittstelle halten sowie das Contactbook in der appConfiguration eintragen.

datachannelMessage Beschreibt den Aufbau der Messages, die über den Data-Channel der PeerConnection gesendet werden.

signalingChannel Beschreibt den Aufbau der Messages, die über den Signaling-Channel gesendet werden.

Class Channel

Defined in: Channel.js.

Class Summary	
	Channel(account) A channel connects the client with a signaling server.

Field Summary	
	implementInterface Channel Interface implementation declaration ('Model/Interfaces /ChannelInterface').

Method Summary	
	addReceiveListener(listener) Adds a listener to receive messages.
	removeReceiveListener(listener) Removes a listener from the receivers list.
	send(message) Sends a message over the channel.
	start() Starts the channel: begins receiving messages, listens for server push or whatever.
	stop() Stops the channel: closes the channel connection and removes all listeners.

Class Detail

Channel(account)

A channel connects the client with a signaling server. It is used for transporting signaling messages between the client (peer) and a central server. Peers can send messages over the channel and register listeners to be notified on a message arrival. A Channel does not run its service after initialization. The service has to be started on 'start'.

Parameters:

{Account} **account**
- A channel account (Model/Domain/Account) of a user (Model/Domain/User). Every Account is bount to it's channel. The channel will be injected with its own account.

Field Detail

implementInterface

Channel Interface implementation declaration ('Model/Interfaces/ChannelInterface'). The channel management checks for this declaration and will throw ans Error, if the declaration is missing or functions are missing.

Method Detail

{boolean} **addReceiveListener**(listener)

Adds a listener to receive messages. Adds the listener only if it's not yet registered.

```
var listener = new MyClass(param1, param2);
listener.notify = function(message) { console.log(message); };

channel.addReceiveListener(listener);
```

Parameters:

{notify: function({ receiver: string; message:string })} **listener**
- An object implementing a notify(message) method

Returns:

{boolean} listener was added or not

---

removeReceiveListener(listener)

Removes a listener from the receivers list.

Parameters:

{notify: function({receiver: string; message: string })} **listener**
- An object implementing a notify(message) method

---

send(message)

Sends a message over the channel.

Parameters:

{receiver: string; message: string} **message**
- A message object

---

start()

Starts the channel: begins receiving messages, listens for server push or whatever.

---

stop()

Stops the channel: closes the channel connection and removes all listeners.

Class Contactbook

Defined in: Contactbook.js.

Class Summary	
	Contactbook() A contactbook defines import and storage logic for a specific format and source.

Field Summary	
	data Contains the contactbook data.
	dataSourceType Defines the type of the contactbook.
	implementInterface Contactbook Interface implementation declaration ('Model/Interfaces /AddressbookInterface').
	type Contactbook path definition.

Method Summary	
	count() Gets the number of contactbook entries.
	getEntries() Gets the contactbook entries.
	load(source, successCallback) Loads the contactbook data initial or on every application start.

Class Detail

Contactbook()

A contactbook defines import and storage logic for a specific format and source. The contactbook ist stored in the localstorage. After restarting the application, the contactbooks will be reinstanciated and injected with the data from the storage.

Field Detail

data

Contains the contactbook data.

---

dataSourceType

Defines the type of the contactbook. This information is used to restore the contactbook correctly from storage. The type has to correspond with the configuration in 'appConfiguration.contactbookImport.file'.

```
Addressbook.dataSourceTypes.file
```

---

implementInterface

Contactbook Interface implementation declaration ('Model/Interfaces/AddressbookInterface'). The contactbook

management checks for this declaration and will throw ans Error, if the declaration is missing or functions are missing.

---

type

Contactbook path definition. Require.js uses this information to reinstanciate the contactbook after restarting the application. The path should be relatively to the 'Src' directory and the class is declared without '.js'.

```
'Model/Domain/Addressbook/AddressbookRemoteJson'
```

Method Detail

{int} **count()**

Gets the number of contactbook entries.

Returns:

{int}

{array} **getEntries()**

Gets the contactbook entries.

Returns:

{array} A list of contactbook entries. Should return a reference to a member and not a copy cause the UI will track changes on this list.

---

load(source, successCallback)

Loads the contactbook data initial or on every application start. Ever contactbook type uses an own signature of load!

Parameters:

{string|array|url} **source**
- A string containing the content of the file the user uploaded (contactbook type 'file') or a list of strings containing the content of the files the user uploaded (contactbook type 'directory') or the url (string) of the remote contactbook data (contactbook type 'online').

{function} **successCallback** *Optional*
- This function has to be called after successful loading of the online source (contactbook type 'online' only).

Namespace datachannelMessage

Defined in: DatachannelMessage.js.

Namespace Summary	
	datachannelMessage The datachannel message is sent over the datachannel of the peer connection to the other peer.

Field Summary	
	message - The content of the message.
	messageType - The type of the message.

Namespace Detail

datachannelMessage

The datachannel message is sent over the datachannel of the peer connection to the other peer. Datachannel messages can contain various properties. The property 'messageType' and message are mandatory. A datachannel is encrypted and bound to the two peers. Authentication is not needed, cause only this peers have access to the messages.

```
peerConnection.dataChannel.send({
        messageType: 'system',
        message: 'bye'
});
```

```
peerConnection.dataChannel.send({
        messageType: 'user',
        message: 'Hello Bruce. How are you?'
});
```

Field Detail

{string} **message**

- The content of the message. System messages contain system commands like 'bye'. User messages contain chat messages.

{string} **messageType**

- The type of the message. Types: 'system', 'user'.

Namespace signalingMessage

Defined in: SignalingMessage.js.

Namespace Summary	
	signalingMessage The signaling message is sent over the signaling channel to the other peer via the signaling server.

Field Summary	
	sdp - The session description protocol generated by the RTCPeerConnection.
	sender - The senders username oder nickname.
	type - The type of the message.

Namespace Detail

signalingMessage

The signaling message is sent over the signaling channel to the other peer via the signaling server. Signaling messages can contain various properties. The property 'type' is mandatory.

```
signalingChannel.send({
        type:"bye",
        sender: "brucewillis"
});
```

```
signalingChannel.send({
        type:"candidate",
        label:0,
        id:"audio",
        candidate:"a=candidate:2391379543 1 udp 2113937151 152.96.232.113 34319 typ host
});
```

```
signalingChannel.send({
        type:"offer",
        sdp:"v=0\r\no=Mozilla-SIPUA-26.0 24254 0 IN IP4 0.0.0.0\r\ns=SIP Call\r\nt=0 0\r
        sender:"brucewillis"
});
```

Field Detail

{string} **sdp**

- The session description protocol generated by the RTCPeerConnection.

{string} **sender**

- The senders username oder nickname. Used to show the callee who is calling and to addressing the signaling

channel answer correctly.

{string} **type**

- The type of the message. Type: 'offer', 'answer', 'candidate', 'bye'.

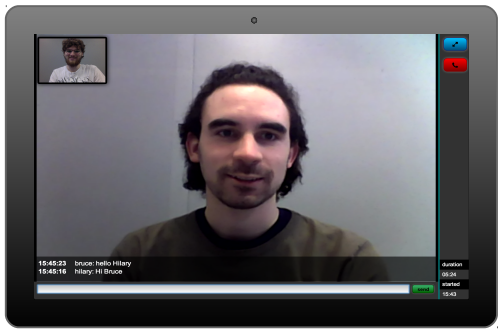
HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

A Practical JavaScript-Only Video-Over-IP Communication Platform

Jannis Grimm, Tobias Blaser

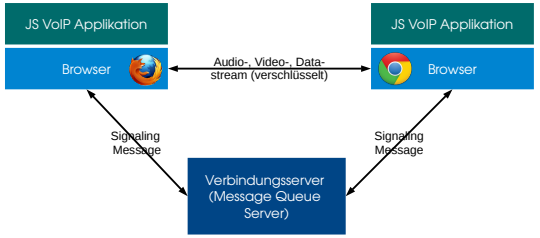
Betreuer: Prof. Dr. Luc Bläser

Studienarbeit HS 2013
Abteilung Informatik



Im Rahmen der Studienarbeit «A Practical JavaScript-Only Video-Over-IP Communication Platform» wurde eine Video-Kommunikationsapplikation entwickelt, die im Browser läuft und ausschliesslich in JavaScript geschrieben wurde.
Die Applikation unterstützt den Import von Kontaktbüchern und bietet Schnittstellen für eigene Kontaktbuchformate.
Die Applikation bietet zudem eine Schnittstelle für einen beliebigen Verbindungsservice und enthält eine Referenzimplementation.

Funktion



Benutzer verbinden sich mit einem Verbindungsservice. Um Anzurufen sendet die Applikation über den verbundenen Verbindungsservice eine Anfrage an den andern Teilnehmer. Kommt eine Antwort zurück, bauen beiden Seiten den Peer-to-Peer Stream auf starten die Kommunikation.

Erkenntnisse

- ✓ Die neusten Versionen der modernen Browser unterstützen die Technologie.
- ✓ Auch die Browser-Versionen für Tablets und Mobiles unterstützen den Standard.
- ✓ Bandbreite, Qualität und Format werden automatisch an Verbindung, Prozessorleistung und Kamera angepasst.
- ✓ Die Codierung des Mediastreams und die Verschlüsselung sind momentan noch sehr resourcenhungrig.
- ✓ Videokommunikation und Datenübertragung (Chat) funktionieren browserübergreifend.

Features

- ✓ Plattformunabhängig
- ✓ Ohne Software Installation
- ✓ Erweiterbar
- ✓ Responsive
- ✓ Modernste Web-Technologie
- ✓ Kommunikation verschlüsselt
- ✓ Open Source
- ✓ Firewalls und NATs passierbar
- ✓ Kabel- und Mobilfunknetz
- ✓ Chat

Schlussfolgerungen

- ✓ Es ist möglich nur mit WebTechnologie eine Videokommunikations-Plattform aufzubauen.
- ✗ Die existierenden SIP-Server sind noch nicht bereit, um vom Browser aus als Verbindungsservice genutzt zu werden.
- ✓ Die Applikation wurde so konzipiert, das sie später um Dateiaustausch, Konferenzschaltung oder Screensharing erweitert werden könnte.
- ✓ Auf genug modernen Geräten lässt sich die Technologie bereits heute in vollem Umfang nutzen.