NTNU – Trondheim
Norwegian University of
Science and Technology

Evaluation of Telecom Operator Enabled Internet Telephony by Creating a Proof-of-Concept Web Application

Thomas Bruun

Master of Science in Computer Science
Submission date: June 2013
Supervisor: John Krogstie, IDI
Co-supervisor: Svein Yngvar Willassen, Telenor Comoyo AS

Norwegian University of Science and Technology
Department of Computer and Information Science

Preface

The task in this thesis is to evaluate the entry of traditional telecommunication operator services on the internet. This will be done by creating a proof-of-concept web telephony application, allowing users to place phone calls using an application programming interface provided by the Norwegian operator Telenor. The task shall be completed according to the design science research method.

I wish to thank my supervisor John Krogstie at the Department of Computer and Information Science, Norwegian University of Science and Technology for his guidance throughout the work of this thesis. I would also like to thank my company supervisor Svein Willassen at Telenor Comoyo AS, for his help in shaping the scope of this thesis. Finally, I owe an enormous amount of gratitude to Sergey Zyrianov at Comoyo, for implementing the missing features on the server side of Talk+, and for guiding me through the world of voice and networking.

Abstract

Since the beginning of the 21st century, telecommunication operators have witnessed the arrival of internet services that are increasingly taking over traditional voice and messaging markets. Some operators are realizing that they need to bring their traditional services to the internet, in order to keep up with the modern consumer. In this thesis, I create an application allowing phone calls to be placed from an internet browser using a customer's existing phone number. The application extends the existing service Talk+, currently developed as native phone applications by Telenor Comoyo AS. The planning and implementation is conducted by examining modern technologies available to web developers, and by utilizing these in a JavaScript application. The application is evaluated using a set of requirements derived from the existing implementation of Talk+ for iPhone.

The result is a working browser telephony application for outgoing calls, with the same quality of voice and delay as on a cellular phone. The thesis therefore concludes that it is possible to create such an application with the technology available, but underlines that the technology is at an early stage, and that it lacks the support from several large browser vendors.

Oppsummering

Siden starten på det 21. århundre har telefonoperatører vært vitne til at et økende antall internettjenester tar over for tradisjonelle tale- og meldingstjenester. Enkelte operatører innser at de er nødt til å tilby sine tradisjonelle tjenester på internett for å kunne tilfredstille den moderne forbrukeren. I denne hovedoppgaven lager jeg en applikasjon som muliggjør telefoni fra en nettleser, med en kundes eksisterende telefonnummer. Applikasjonen bygger på den den eksisterende tjenesten Talk+, som for tiden utvikles som en mobilapplikasjon av Telenor Comoyo AS. Planleggingen og implementasjonen blir utført ved å undersøke moderne webteknologier som er tilgjengelige, og ved å bruke disse i en JavaScript-applikasjon. Applikasjonen blir utviklet ut fra et sett med krav som er basert på den eksisterende implementasjonen av Talk+ til iPhone.

Resultatet er en fungerende telefoniapplikasjon for utgående samtaler, med den samme kvaliteten på stemme og forsinkelser som mellom to mobiltelefoner. Denne avhandlingen konkluderer derfor med at det er mulig å lage en slik applikasjon med den tilgjengelig teknologien, men understreker at teknologien er på et tidlig stadium, og at den mangler støtte fra flere store nettleserprodusenter.

Contents

Preface i

Abstract ii

Oppsummering iii

List of Figures ix

Abbreviations xi

1 Introduction 1
- 1.1 Three research questions . . . 1
- 1.2 Motivation . . . 2
- 1.3 Thesis outline . . . 3

2 Background 5
- 2.1 Web standards . . . 5
  - 2.1.1 Who defines the web? . . . 6
  - 2.1.2 WebRTC . . . 6
  - 2.1.3 WebSockets . . . 12
  - 2.1.4 WebStorage . . . 12
- 2.2 The telecommunications industry . . . 13
  - 2.2.1 Evolution of telephony . . . 13
  - 2.2.2 Combining old and new technology . . . 14

3 Research Method 17
- 3.1 Design science . . . 17
- 3.2 Artifact & evaluation . . . 18

4 Requirements 19
- 4.1 Determining the requirements . . . 19
- 4.2 Functional requirements . . . 20
  - 4.2.1 Requirements list . . . 20
  - 4.2.2 Requirements details . . . 20

4.3 Non-functional requirements . . . 23
4.3.1 Requirements list . . . 23
4.3.2 Requirements details . . . 24

5 Architecture & Technology 27
5.1 Existing architecture . . . 27
5.1.1 Servers . . . 27
5.1.2 Native clients . . . 30
5.2 Talk+ web application . . . 31
5.2.1 Languages . . . 32
5.2.2 Authentication . . . 33
5.2.3 Signaling . . . 33
5.2.4 Storage . . . 33
5.2.5 Media transfer . . . 34

6 Implementation 35
6.1 Code . . . 35
6.1.1 AppController . . . 37
6.1.2 StorageHandler . . . 37
6.1.3 AppView . . . 37
6.1.4 ServerConnection & CommandHandler . . . 38
6.1.5 MediaEngine . . . 39

7 Evaluation 45
7.1 Functional requirements . . . 45
7.2 Non-functional requirements . . . 49

8 Discussion & Conclusion 53
8.1 Discussion . . . 53
8.2 Conclusion . . . 54
8.3 Further work . . . 56

A Source Code 57
A.1 index.html . . . 57
A.2 app.css . . . 58
A.3 app.js . . . 59
A.4 storageHandler.js . . . 60
A.5 appView.js . . . 60
A.6 serverConnection.js . . . 62
A.7 commandHandler.js . . . 64
A.8 mediaEngine.js . . . 67

B Tests 71

Bibliography **73**

List of Figures

2.1 Example workflow of WebRTC 7
2.2 Example of two servers behind a router employing NAT 9
2.3 WebRTC implementation architecture 11

5.1 A simple overview of the Talk+ architecture 28
5.2 Signaling and media flow in the Talk+ service 30
5.3 The WebRTC core module 31

6.1 Talk+ JavaScript classes, and how they interact with the browser components 36
6.2 Talk+ web application 36
6.3 Talk+ authentication flow 42
6.4 Talk+ call flow 43

B.1 Tests results 71

Abbreviations

OTT	**O**ver-**t**he-**t**op
SMS	**S**hort **M**essage **S**ervice
API	**A**pplication **P**rogramming **I**nterface
IP	**I**nternet **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
NAT	**N**etwork **A**ddress **T**ranslation
STUN	**S**ession **T**raversal **U**tilities for **N**AT
TURN	**T**raversal **U**sing **R**elays around **N**AT
GTURN	**G**oogle **T**raversal **U**sing **R**elays around **N**AT
ICE	**I**ntercative **C**onnectivity **E**stablishment
SDP	**S**ession **D**escription **P**rotocol
SIP	**S**ession **I**nitiation **P**rotocol
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
VoIP	**V**oice **o**ver **I**nternet **P**rotocol
PSTN	**P**ublic **S**witched **T**elephone **N**etwork
PLMN	**P**ublic **L**and **M**obile **N**etwork
WebRTC	**Web** **R**eal-**T**ime **C**ommunication
W3C	**W**orld **W**ide **W**eb **C**onsortium
IETF	**I**nternet **E**ngineering **T**ask **F**orce
ITU	**I**nternational **T**elecommunication **U**nion
DOM	**D**ocument **O**bject **M**odel
HTML	**H**yper**T**ext **M**arkup **L**anguage
CSS	**C**ascading **S**tyle **S**heets
CU-RTC-Web	**C**ustomizable, **U**biquitous

	Real **T**ime **C**ommunication over the **Web**
JSEP	**J**avascript **S**ession **E**stablishment **P**rotocol
ROAP	**R**TCWeb **O**ffer/**A**nswer **P**rotocol

Chapter 1

Introduction

Telecommunication operators have in the past few years witnessed the rise of third-party *over-the-top* (OTT) applications, such as Skype and Whatsapp, who are increasingly taking over the market for traditional telephony and *Short Message Service* (SMS) messaging[1]. For a long time, the operators have been following this trend passively. Some have now begun to explore the opportunities that lie in the use of internet for communication, and have started offering services connecting their traditional telecommunication infrastructure to the web.

1.1 Three research questions

The overall goal in this thesis is to evaluate the entry of traditional telecommunication operator services on the internet. This is conducted by creating a proof-of-concept internet telephony client for the use in desktop internet browsers. The service is built on the existing service Talk+, which is developed by the Norwegian operator Telenor.

Three research questions have been derived from the overall goal.

1. **Is it technically possible to create the application?**
2. **Is the quality of the application good enough, compared to traditional telephony?**
3. **Is this a product that customers will use?**

The main focus in this thesis are the technical aspects of the solution, such as functionality and performance. A greater amount of attention will therefore be given to the first two research questions, than to the user acceptance aspect in the last question.

1.2 Motivation

The primary motivation for this thesis is to help bridge the gap between traditional telephony and internet communication. Telecommunication operators have for too long distanced themselves from new, internet based, technologies, while the younger generation of consumers have switched to cheaper, easier, and more available means of communication[2]. The internet has come to stay, and unlike the phone networks, it is borderless in most parts of the world.

A secondary motivation is interest in the open web platform. The web is not defined by one company and its commercial interest, but by the technology community working together towards open and free standards. The open web standards allow a developer to create *one* application which will work on all major platforms, without depending on the approval of a large cooperation with absolute control over distribution.

1.3 Thesis outline

Chapter 2 presents some background on how the internet is evolving, and what technologies are available to allow browser-based telephony. The same chapter provides an introduction to the telecommunications industry, as well as examples of some of the companies wanting to integrate their traditional telephony services with the internet.

Chapter 3 presents the research method used in this thesis, identifies the *artifact* to improve, and formalizes how that artifact will be evaluated. In **Chapter 4**, all the requirements, both functional and non-functional, are developed and explained in detail.

Chapter 5 first gives a detailed overview of the existing Talk+ architecture and clients, and then presents the technology choices made for the web client. In **Chapter 6**, the system implementation and design is detailed.

In **Chapter 7**, the implemented solution is evaluated, using a requirement fulfillment analysis. Lastly, in **Chapter 8**, a discussion is held, and a conclusion is drawn by revisiting the research questions listed in Section 1.1. A brief discussion on further work is also presented in the last chapter.

Chapter 2

Background

This chapter starts by presenting how the internet is evolving with the help of cooperative standardization. Next, a closer look is taken at some modern web standards, including *Web Real-Time Communication* (WebRTC), the technology that some open web supporters are hoping will define the future of video and audio communication[3]. Next, the chapter moves on to describe the past and present situation in telecommunications, and how traditional services are starting to merge with modern technology, using examples from telecommunication operators around the world.

2.1 Web standards

Before starting on the architecture for the Talk+ web application, we need to know, and understand, the environment in which it will be built. This section starts by explaining how web standards are developed, and then moves on to look at three web standards relevant for this thesis.

2.1.1 Who defines the web?

If a person downloads the latest version of the four most popular internet browsers, chances are high that an arbitrary website will look and behave almost identical in each of the browsers. All browser vendors, i.e. the companies who develop the browsers, aim to implement the web standards designed by the *World Wide Web Consortium* (W3C)[4]. These standards recommend what *Application Programming Interfaces* (APIs) should be available, and their expected behavior, but they do not dictate the implementation.

The vendors are not *required* to implement the standardized APIs. The goal of the W3C is rather to develop standards using a consensus-based decision process, allowing all vendors to participate in the debate. The process is long and thorough, and will not be deeply detailed in this thesis. In short, standards, or recommendations, are composed by *working groups*, which often include members from the major browser vendors. The groups discuss and compose several official *working drafts*. In between the working drafts, *editor's drafts* are released with content that reflects the latest development and discussions. In the end, if a consensus is reached, a *final recommendation* is approved and published[5].

While the W3C define standardized APIs for browsers, the *Internet Engineering Task Force* (IETF) define standards for lower-level protocols[6]. The combined work done by the IETF and W3C is significant in the area of WebRTC, which will be detailed in Section 2.1.2.

2.1.2 WebRTC

WebRTC is a project consisting of three main parts. The first two parts are the browser API and protocol standardization processes, managed by the W3C and the IETF. The third part is the implementation process, which is currently a cooperation between Google, Mozilla and Opera[7]. This section starts by detailing the overall architecture of the WebRTC technology, before briefly describing the implementation process.

W3C standard

WebRTC is one of the newer W3C API proposals, and was originally introduced by Google[8]. The purpose behind the WebRTC technology is to enable real-time flow of media between two or more users, without the need for third-party software. At the time of writing, the latest WebRTC API standards working draft is from August 21st 2012[9], while the latest editor's draft is from March 22nd 2013[3]. In the time between these two drafts there have been six other editor's drafts, indicating that there is still a large amount of activity in the work on this standard.

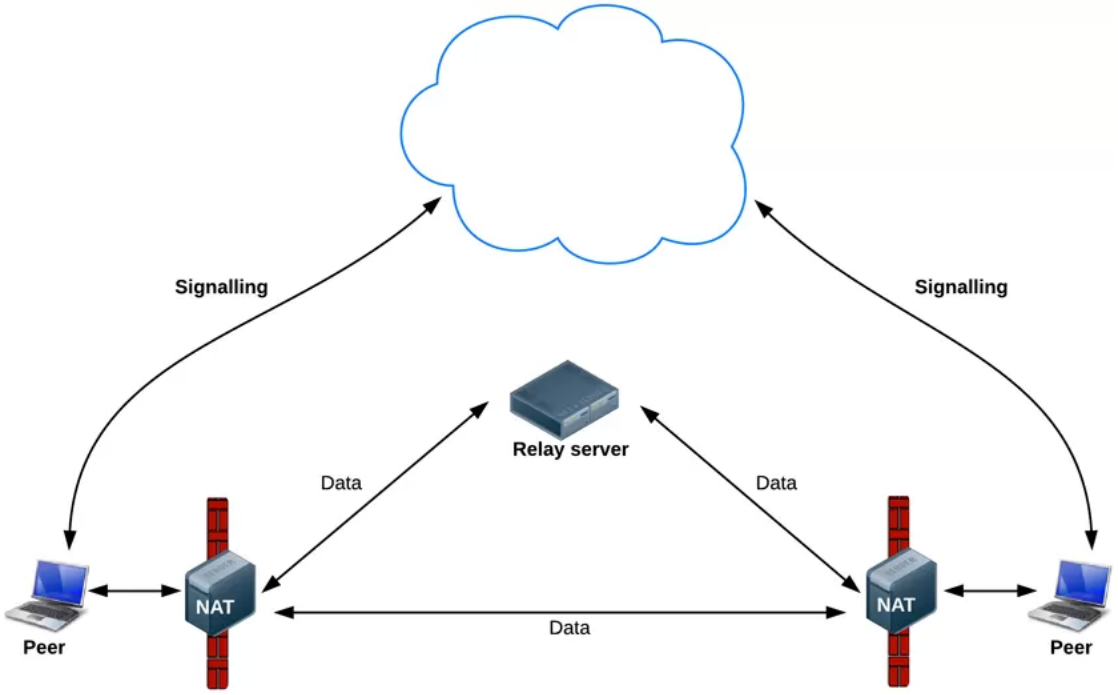

FIGURE 2.1: Example workflow of WebRTC

Source: http://www.html5rocks.com/en/tutorials/webrtc/basics/

A use case often presented when presenting the technology, is audio or video transfer between two users in each their own browser. Figure 2.1 is an example of this workflow. The peer on the left, hereby referred to as the *caller*, wants to initiate a conversation with the peer on the right, the *callee*. The process starts with a procedure called *signaling*.

Signaling

Signaling is used for exchange of information between the peers, such as the user's wish to initiate or end a call, IP addresses, information about media capabilities, and all other information the peers may need to exchange. How the signaling messages are transferred is entirely up to the application. One of the available methods for signaling transportation, is the WebSockets protocol. The protocol is one of the more recent web standards, and will be further detailed in Section 2.1.3.

Although signaling is executed in the manner that the application wishes, there are some steps that needs to be performed in order for media to start flowing between the peers. The steps are described in IETF's *Javascript Session Establishment Protocol* (JSEP) document[10]. As previously mentioned, a conversation consists of a caller and a callee. The first step involves the caller creating an *offer* for the callee. The offer needs to contain a *Session Description Protocol* (SDP), which includes all necessary information about the caller. Examples of information that could be included in the SDP are IP addresses, media codec capabilities, and encryption information. The offer is processed by the callee, matching it to its own capabilities, and creates an *answer*, which is sent back to the caller. If the information in the answer's SDP is successfully processed by the caller, media may begin transferring. This process is further detailed and depicted in Chapter 6, Implementation.

Networking

Signaling must be initiated using an intermediate server. However, one of the principles of WebRTC is that the media shall be able to flow between the caller and callee directly. Establishing a connection between two peers require some knowledge of how the internet works. A frequent issue in networking, is discovering a client's public *Internet Protocol* (IP) address. IPv4, which is set to be replaced by IPv6, is the old standard most frequently used on the internet today[11]. To tackle the problem with IPv4 address scarcity, most office and home networks employ the *network address translation* (NAT) process[12]. This process maps a

unique internal IP address for all devices on the same network, while only using one public IP address, as depicted in Figure 2.2.

There are several different types of NAT, and this thesis does not go into detail on how each works. For some NATs, the client may use a method called *Session Traversal Utilities for NAT* (STUN) in order to discover the computer's publicly reachable IP address. STUN requires the client to send a request to a dedicated STUN server, which returns the client's public IP as seen from the outside. This process requires very little traffic and load on the server. However, some type of NATs do not support the use of STUN for IP discovery, because the external IP will vary from each external observer. The IP discovered by STUN will therefore not be valid. One solution to this problem is to use *Traversal Using Relays around NAT* (TURN)[13]. TURN also uses a third party server, but in stead of using the server for IP discovery, *all data* is relayed through it. In a setting where two clients are exchanging media, this means a substantial load on the TURN server.

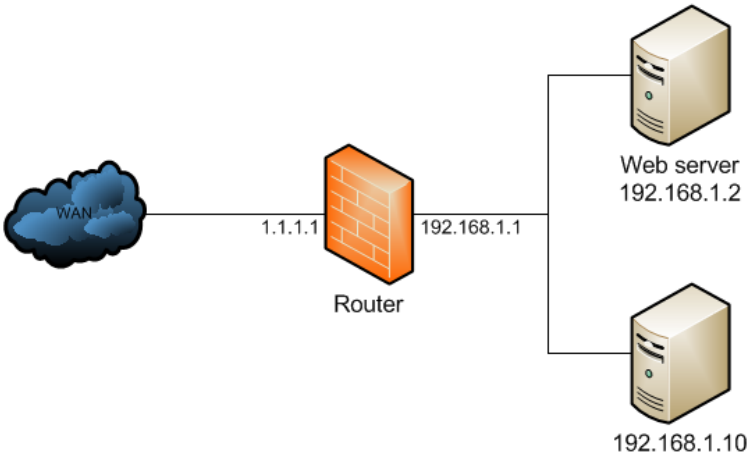

FIGURE 2.2: Example of two servers behind a router employing NAT

Source: http://wiki.mikrotik.com/wiki/Hairpin_NAT

WebRTC uses a method called *Interactive Connectivity Establishment* (ICE) to handle the issues of discoverability[14]. The purpose of ICE is to gather several *candidates*, which is a set of transport addresses consisting of IP addresses and ports. All candidates have been determined as *possible* connection points for reaching the client. Examples of candidates are the public IP of the client's router,

a combination of the public IP and a port mapped by the NAT to the client, and an IP allocated from a TURN server. After the ICE candidate gathering process is over, ICE will order the candidates according to an internal priority algorithm. The goal of this ordering is to place the most direct route to the client first, and routes through one or more relay servers last. After the ordering, the candidates may be sent to the other peer through a signaling offer, and that peer will try each candidate in the received order until it can establish a working connection.

Implementation

Implementation of WebRTC is divided into several areas, as depicted in Figure 2.3. The core component handling complexities such as video and audio encoding, and media transfer, is developed jointly by a group of developers from, among others, Google and Mozilla[7]. The component is open-source and may be used by any browser vendor. This eases the implementation for the individual browser vendors, since they will only need to implement the W3C WebRTC API, and not the whole underlaying stack. The browser vendors work on the API implementation in a varying tempo. Depending on the browser a user has installed, and its version number, a WebRTC application may not work as intended. Evaluations of the browser implementation status, and the overall support for WebRTC, will be performed in Chapter 7 and 8, respectively.

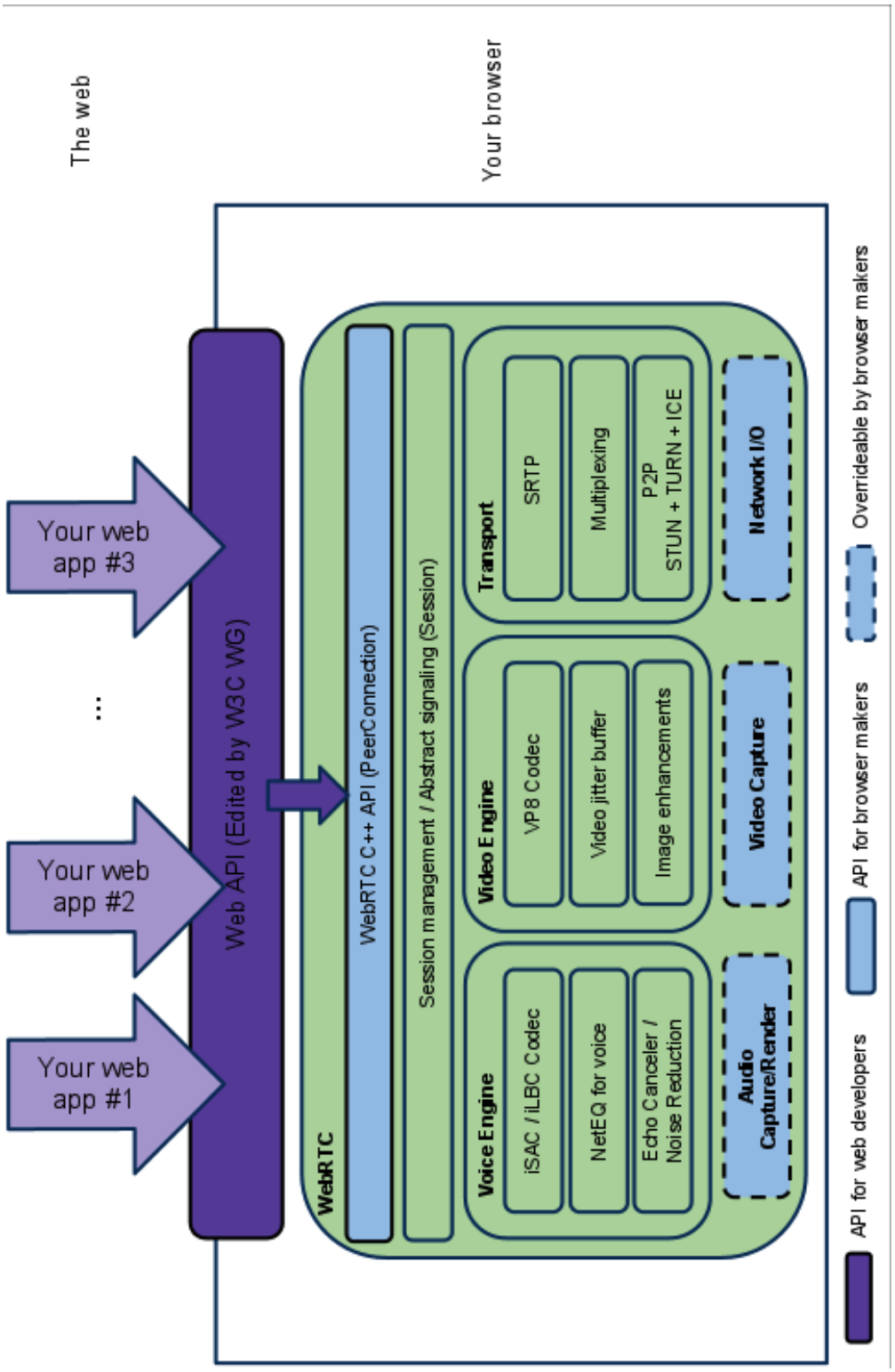

FIGURE 2.3: WebRTC implementation architecture

Source: http://www.webrtc.org/reference/architecture

2.1.3 WebSockets

The *HyperText Transfer Protocol* (HTTP) request-response technique[15] is the standard method for a browser to send and retrieve data from a website. For every request sent to the server, a corresponding response is expected. The server can not send data to the browser whenever it wishes to do so. Applications developed for other platforms than the browser often solve this limitation using a *Transmission Control Protocol* (TCP) socket, which allows for both ends of the connection to send data at any time. TCP is a widely accepted protocol for transporting information over the internet, but is not implemented as a web standard, as is therefore not available to developers of web applications. This limitation is challenged by WebSockets.

The WebSockets API allow browsers to establish an always-on connection to a server. The WebSockets protocol is built on top of TCP, hiding its complexities from the web developers by allowing them to send whole messages, in stead of packets[16][17]. Having an open socket to a server allows for two-way communication with minimum delay, which may in turn increase performance in real-time communication applications. The WebSockets standard was releases as a candidate recommendation in September 2012[17].

2.1.4 WebStorage

Websites have been able to store information on the client's computer for a long time by using *cookies*[18]. Cookies are small key-value files which may store any textual information that the website wishes. When the user visits the same site at a later time, the content of the cookie is sent to the website with *each* request. If the information stored is primarily needed for client-side use, the use of a cookie will send unnecessary amounts of data to the server. Browsers limit cookie access to the website that created the cookie. This limit exists for security reasons, as the cookie may contain sensitive information about the user on that website.

Modern browser standards have introduced other means of storing data on the client. The web standard *Web Storage* details a key-value storage method, similarly to cookies, in which the content is stored locally[19]. One of the Web Storage APIs implemented in most modern browsers, is *localStorage*. One way in which localStorage differs from cookies, is that its content is not passed on to the web server with every request. In a situation where the whole contact list of a user is stored locally in order to quickly retrieve it at a later time, storing it in a cookie would potentially mean a lot of data transferred to the web server with every request.

2.2 The telecommunications industry

2.2.1 Evolution of telephony

Telecommunication is a large field in engineering, and has changed tremendously since its birth in the 19th century[20]. This section concentrates on the state of voice communication in the current century.

Traditional telephony

Up until the 2003 launch of the voice communication software Skype[21], to *call* someone usually referred to the use of a phone over the *public switched telephone network* (PSTN) or *public land mobile network* (PLMN), commonly known as the *landline* and *cellular* networks. In 2003, internet chat and e-mail had existed for several years, but household internet connections had not the capacity to transfer audio back and forth with good enough quality and latency.

Internet telephony

As internet access became faster and more available, *Voice over Internet Protocol* (VoIP) solutions emerged. First out was the internet providers themselves, who offered household phones connected to the internet, rather than to the analogue

PSTN[22]. This meant that the phone traffic could be sent as IP packets as far as possible, before it eventually was switched over to the PSTN and the receiving party. This eased the load on the operators' phone networks, as they only needed to use the PSTN on the last transport leg.

VoIP services continued to expand with the entry of closed-network protocols, such as Skype, which was one of the first successful proprietary VoIP softwares[21]. These protocols are closed-source, allowing only applications approved by the network's owner to access it. In recent years, effort has been made to create open-network protocols, allowing multiple proprietary clients to communicate. The standardization of WebRTC is one such effort.

2.2.2 Combining old and new technology

Telecommunication operators are noticing the increasing customer migration from traditional telephony to internet-based audio and text communication[1]. Now, rather than just having to compete within the same industry, the operators have to think differently in order to hold on to the market. One of the areas some of these operators are now investing money and engineering resources into, is combining their traditional voice systems with modern internet services. This section lists some of the operators who are actively pursuing this new direction.

Telenor

Telenor is a Norwegian telecommunication operator with a long history in telecommunication. Established in 1855 by the Norwegian government under the name Telegrafverket, it is now a worldwide operator with over 170 million mobile subscriptions, and 34.000 employees[23][24].

Telenor Comoyo AS, hereby referred to as Comoyo, is an independent subsidiary of Telenor[1]. The company was founded in January 2011, and its purpose is to develop internet services for the consumer market in Telenor. Telenor has tasked Comoyo with creating a internet-oriented telephony service, which they

have named Talk+. The service is aimed at Telenor customers who want to use the internet to call from their personal telephone number to any other phone number. Development on Talk+ is divided into work on server and client components. On the server side, Comoyo receives traffic from all clients, and translates it into the standardized *Session Initiation Protocol* (SIP) protocol for communicating with the Telenor telephony infrastructure[25]. In order to make communication between client and server as consistent as possible, all clients must use WebRTC for media transfer.

Telefonica

Telefonica is a Spanish operator with customers in Europe and South America, and operates under the brand names O2 and Movistar. In February 2013, under the O2 flag, they released a service called TU Go in the United Kingdom[26]. It allows the customer to make phone calls and send SMS from their own number, using their existing Telefonica price plan. The service is currently launched as applications for the mobile operating systems iPhone and Android, as well as the Windows desktop platform.

AT&T

The American telecommunication operator AT&T has existed since the monopoly on telephony in the United States in the 19th century. Serving just over 100 million customers in the United States in 2012, it is the second largest operator in the country[27][28].

AT&T has chosen a somewhat different direction than Telefonica and Telenor. In February 2013, the American operator released the alpha version of a developer library allowing developers to develop web applications with built-in call functionality. The users logs in with an access key unique to every AT&T customer, and proceeds to call using his or her subscription. Their solution also uses the previously mentioned WebRTC technology[29].

Chapter 3

Research Method

This chapter starts by describing the research method chosen for this thesis. It then moves on to identify what the research shall produce, and how it will be evaluated.

3.1 Design science

The research method applied in this thesis is *design science research*. The main idea in this research paradigm is to take an existing *kernel theory*, find a related organizational problem, and try solve the problem using an innovative information technology artifact. The artifact may range from a concrete product, to an informal description[30].

In *Design Science in Information Systems Research*[30], the authors provide a set of seven guidelines for design science research. The guidelines are intended to help understand, execute, and evaluate the research performed. Guideline 1 and 2 were mentioned above, stating that the research must produce an artifact to a known problem. Guideline 3 states that the evaluation of the resulting artifact must thoroughly prove its utility, quality, and efficacy. This is achieved by using well-executed evaluation methods, supported by well-defined metrics. The evaluation methods used in this thesis are detailed later in the chapter.

Guideline 4 states that there must be a clear research contribution. In this thesis, the design artifact itself can be considered a research contribution, as it adds experience to the knowledge base. In guideline 5, it is required that both the construction and evaluation of the artifact is performed rigorously, while still keeping it relevant to the research. Guideline 6 establishes that design is a search process, aimed at finding an effective solution to a problem. It describes the search process as iterative, in which the researcher generate a proposed solution, tests it, and uses the results to generate a new solution. Finally, guideline 7 states that the research must be presentable to both technology aware and management-oriented audiences. For both audiences, the research must show how they may apply the described artifact.

3.2 Artifact & evaluation

Following the design science guidelines, the goal of this thesis is to start with an existing product, and extend its capabilities with help of new technologies. In the context of this thesis, the organization is the telecommunication operator Telenor. The problem Telenor is facing, is that internet services are increasingly expanding into their traditional market with cheaper, more available products.

The proposed solution, or artifact, is identified as an internet hosted Talk+ web application, offering the same functionality as the native clients, built entirely on open web technologies. In Chapter 4, the specific requirements for the application will be detailed. The artifact is to be built as a proof-of-concept application, not a commercially ready product.

The evaluation of the artifact will be performed by first using a requirement fulfillment analysis in Chapter 7. In the analysis, each requirement from Chapter 4 will be compared to the finished product. An overall conclusion on the completeness of the artifact will be drawn in Chapter 8.

Chapter 4

Requirements

This chapter outlays the requirements to the Talk+ web application. The first section details how the requirements have been developed. The following two sections lists and details both the functional and non-functional requirements.

4.1 Determining the requirements

The only formally stated requirements from Comoyo have been that the system should be enable to perform a call from the browser to an arbitrary Norwegian phone number, and that it should use the WebRTC technology.

In addition to these two requirements, the beta version of the Talk+ iPhone application has been examined, resulting in a list of features relevant for the web application. These features have been chosen using a black box observation approach, meaning that they were chosen without looking at the application's code.

It is important to note that the requirements for the Talk+ web application in this thesis, are developed for a prototype application, and not for commercial use. In the event of a commercialization of the application, new requirements should be developed.

4.2 Functional requirements

A functional requirement describes what a system is required to *do*. Relating the requirements to the research questions asked in Chapter 1, the functional requirements shall provide an answer to the first question: **Is it technically possible to create the application?**

4.2.1 Requirements list

ID	Requirement
R1	The user shall be able to log in using Comoyo credentials
R2	The system shall be able to capture input from the user's microphone
R3	The system shall be able to transfer audio from the user to the server
R4	The system shall be able to receive an audio stream from the server
R5	The system shall be able to play back a received audio stream to the user
R6	The user shall be able to perform an outgoing call to a Norwegian phone number
R7	The system shall inform the user when the server rejects a phone number
R8	The user shall be able to perform an outgoing call to a Comoyo user
R9	The system shall inform the user when the server rejects a username
R10	The system shall inform the user if an invalid phone number or username is entered
R11	The user shall be able to place a call to a test number
R12	The user shall be able to view the call log
R13	The application shall inform the user of how much the call is costing
R14	The user shall be able to end the call

4.2.2 Requirements details

R1 - The user shall be able to log in using Comoyo credentials

One of the services Comoyo is creating for Telenor is an global authentication service. This service aims to give one set of credentials to Telenor customers in all of its operating countries, for use with all Telenor services. A customer's account is directly tied to its phone subscription. Comoyo Talk+ is one of the first services integrating with the authentication service, and it is vital for the web application

to support it as well. Without it, the user will not be allowed to perform phone calls.

R2-R5 Audio capture, transfer, and playback

Requirements R2 to R5 address the issues of sending and receiving audio between the client and the Comoyo backend. For a viable calling experience to take place, it is not satisfactory if any of these requirements are unfulfilled.

R6 - The user shall be able to perform an outgoing call to a Norwegian phone number

Requirement R6 states that the user should be able to call to a Norwegian number. The server backend of Talk+ will stop any conversation trying to call a foreign number, or a number connected to a service with special calling rates. Requirement R10 addresses the issue of blocking such numbers in the client.

R7 - The system shall inform the user when the server rejects a phone number

Requirement R7 requires the system to provide the user with feedback if the server rejects a phone number. At this time in the design of the system, it is unknown if the server will provide such specific feedback.

R8 - The user shall be able to perform an outgoing call to a Comoyo user

Similar to requirement R6, this requirement expects the user to be able to call another Comoyo user's username. If the user has a phone subscription connected to the account, the call will be forwarded to the corresponding phone number. The Comoyo username is required to be in the form of an e-mail address.

R9 - The system shall inform the user when the server rejects a username

Requirement R9 requires the system to provide the user with feedback if the server rejects a username. At this time in the design of the system, it is unknown if the server will provide such specific feedback.

R10 - The system shall inform the user if an invalid phone number or username is entered

The server is expected to contain logic for handling invalid input. However, in order to ease the load on the server, and decrease the response time for the user, the client should perform some simple validation of the user's input. A Norwegian phone number is known to be a 8-digit number, optionally prefixed by the country code, *+47*. The Comoyo username is expected to be a valid e-mail address, and is therefore possible to validate.

R11 - The user shall be able to place a call to a test number

The Talk+ iPhone application gives the user the opportunity to place a test call in order to verify that the service is working. The web application should also allow the user to do the same.

R12 - The user shall be able to view the call log

Users of mobile phones have come to expect that all outgoing and incoming calls are placed in a call log. The call log shall note when the call took place, who the call was with, and how long it lasted. The call log should be persistent, allowing the user to access the log on different computers, and in different browsers on the same computer.

R13 - The application shall inform the user of how much the call is costing

In the current state of implementation, the Talk+ service does not subtract from the calling minutes included in the user's subscription. This is currently a technical restriction, which requires all users to pre-purchase credit used for calling. It is therefore a requirement that the application informs the user on how much the

current call is costing per minute, and how much the call has currently cost. After the call is over, the user must also be informed of how much the total cost of the conversation was.

R14 - The user shall be able to end the call
After having initiated a call, the user shall be able to end the call both before and after the call has been answered. When the user ends the call, the remote party shall no longer hear audio from the user's microphone.

4.3 Non-functional requirements

Functional requirements formalize the expected functionality in a system, but not how the system is expected to operate. To fill this gap one may use a set of non-functional requirements. A non-functional requirement describes how a system is required to *behave or perform*. The following non-functional requirements shall be used to answer the second research question asked in Chapter 1: **Is the quality of the application good enough?**

In telecommunications, the clients have little or no control over the transmission delay, connection to the server, and audio quality. Many of the following non-functional requirements address such areas. Those requirements should be viewed as a measure of how the Talk+ service is performing *as a whole*, not solely in the web application.

4.3.1 Requirements list

ID	Requirement
F1	The user shall be able to call using the three most popular internet browsers
F2	The user shall be able to hold a five minute call without losing connection
F3	The participants shall not experience noticeable voice transmission delays
F4	The time from a call is started to media flow has begun, must not exceed 11 seconds
F5	The system shall establish calls with a success rate of 90%

4.3.2 Requirements details

F1 - The user shall be able to call using the three most popular internet browsers

When creating a web application, a developer will normally test whether the website looks and behaves the same across all the major browsers. A user should not have to use a specific browser when visiting a web site, which should also apply to the Talk+ web application. In the scope of this thesis, only the three most used browsers will be considered. According to StatCounter, those browsers are Google Chrome, Microsoft Internet Explorer, and Mozilla Firefox[31].

F2 - The user shall be able to hold a five minute call without losing connection

As a measure of stability and quality of the service, the user shall be able to hold a five minute conversation without losing the phone connection, given that the underlaying network connection is stable.

F3 - The participants shall not experience noticeable voice transmission delays

An important topic in voice communication is voice delay. According to a *International Telecommunication Union* (ITU) recommendation document, 400 milliseconds is regarded as an upper threshold for voice delay. The ITU is the United Nations specialized agency for information and communication technologies. The recommendation states:

"While delays above 400 ms are unacceptable for general network planning purposes, it is recognized that in some exceptional cases this limit will be exceeded."[32]

An upper limit of what voice delay is acceptable in the application should therefore be set at 400 ms. A problem arises with measuring this delay. 400 milliseconds is a short time, and should be measured automatically. This would require the ability to measure on both the phone and in the web application, which requires

equipment too sophisticated given the scope of this thesis. In order to have voice delay included as a requirement, a subjective metric was chosen. The test will be performed by comparing the delay in Talk+, to the delay between two mobile phones.

F4 - The time from a call is started to media flow has begun, must not exceed 11 seconds

Unlike voice transmission delay, the delay described in this requirement is measurable in the web client. In the context of this thesis, the user is said to initiate a call when the user submits the phone number or username in the client. Media flow is said to begin when the browser initiates playback of the remote audio. The timestamp of both events can be recorded, and the time difference automatically calculated.

There is no readily available statistics or requirements from the ITU on the delay described above. In order to arrive at a justifiable requirement, a series of calls where placed from one cellular phone to another. Half the calls were performed from the first phone, and the other half from the second phone. Both phones were using the same operator, and the calls were performed in the same area. The delay was manually measured using stopwatch software on a computer. After eight phone calls, the average delay was measured to 11 seconds.

F5 - The system shall establish calls with a success rate of 90%

A successful call is defined as a call that is established by the user, accepted by the callee, and provides remote audio in both ends of the call.

Chapter 5

Architecture & Technology

This chapter consist of two parts. The first part details the existing architecture of the Talk+ service on the server side, how the web application will have to interact with the servers in order to function, as well as a brief introduction to the architecture of the existing clients. The second part of this chapter is dedicated to planning the Talk+ web application. Server interaction and technology choices are addressed, before the next chapter details the implementation of the application.

5.1 Existing architecture

5.1.1 Servers

In order for a call to be placed from a Talk+ client to a phone, several backend components must be used. Some are required by the WebRTC standard, others are specific to the requirements of Talk+. This section examines the components used, without exposing sensitive information about the components' inner functionality. Figure 5.1 provides a simplified overview of the components, and how they interact.

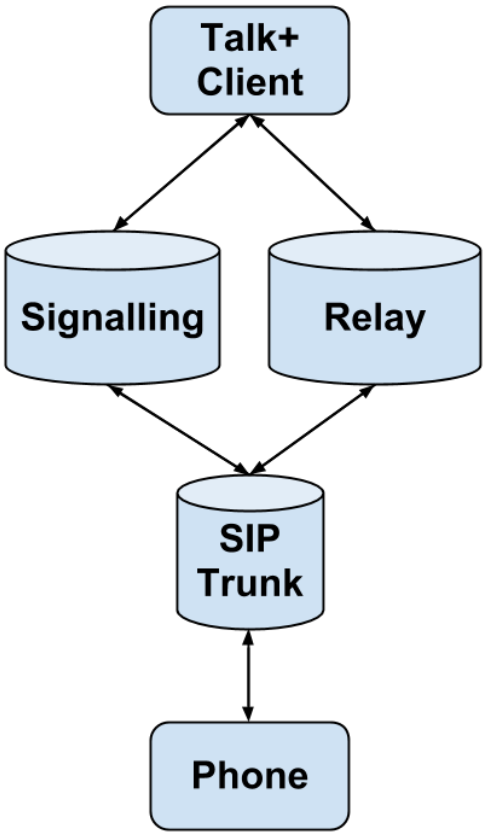

FIGURE 5.1: A simple overview of the Talk+ architecture

As detailed in Chapter 2, WebRTC requires a signaling mechanism in order to set up a connection between two peers. Comoyo uses *one* endpoint for signaling in *all* of its communications products. Although the service is used for several other purposes, such as authentication, SMS sending, and contacts retrieval, this thesis refers to it as the *signaling* server, for simplicity. The signaling server may communicate with native desktop and mobile applications, browser applications, and other platforms which supports communication over the supported protocols. The server supports communication through both a TCP socket, and using WebSockets, as presented in Section 2.1.3.

Chapter 2 briefly described how WebRTC is built for allowing direct media communication between clients. In Talk+, the callee is a node on the PSTN, which does not use the WebRTC protocol. Therefore, there must exists a way for media to be gathered by Comoyo, and relayed to the phone. For this purpose, Comoyo

uses a relay server, for which they issue a username and password per user session. The existing Talk+ clients, which will be detailed in Section 5.1.2, are sending media to this server.

The relay server has a direct connection to the Telenor SIP Trunk, in which all digital voice signals are forwarded to the Telenor PSTN. From the SIP Trunk, the signal is handled by Telenor, leaving the relay server as the endpoint for the Talk+ server infrastructure. The flow of events from the browser to the phone is depicted in Figure 5.2.

According to the WebRTC specification, connecting to a relay using credentials should be performed using the TURN mechanism, as described in Section 2.1.2. However, when the Talk+ project began, TURN was not a part of the specification, so the clients used a similar mechanism, called *Google TURN* (GTURN), which is not available in the web API. Therefore, TURN support had to be added in the Comoyo relay server before the implementation of the Talk+ web application could begin.

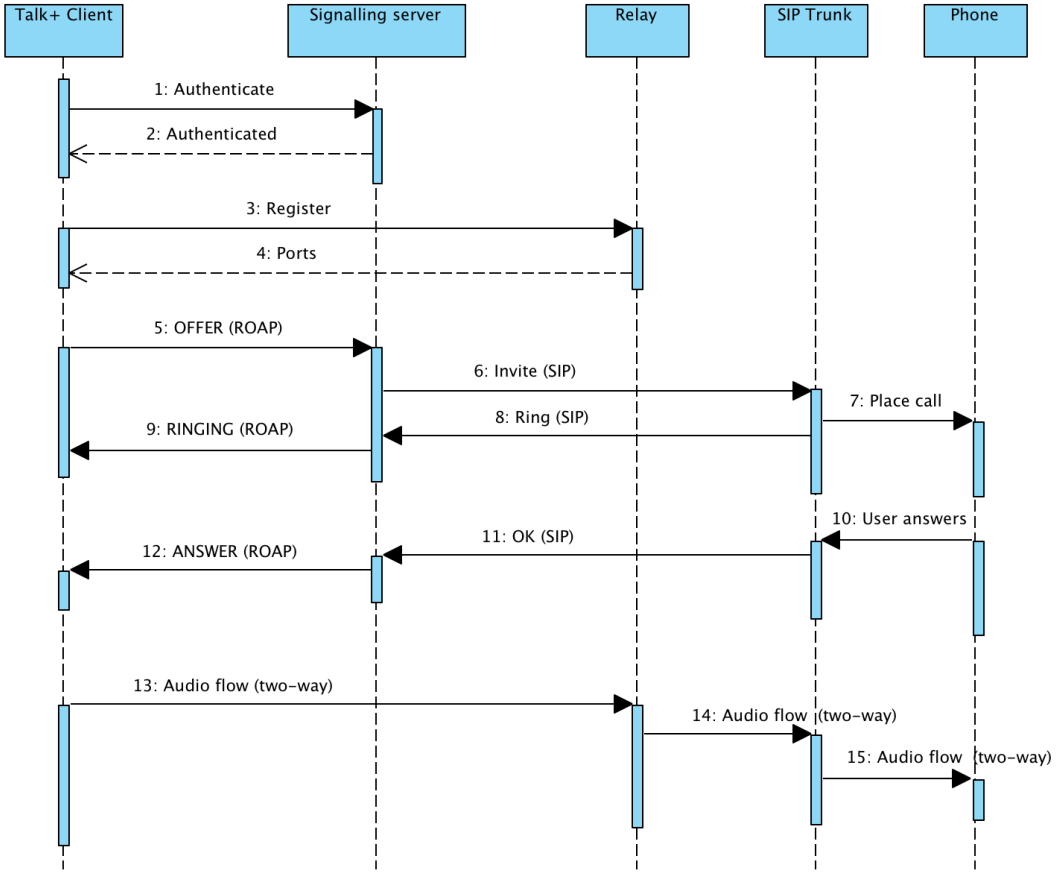

FIGURE 5.2: Signaling and media flow in the Talk+ service

5.1.2 Native clients

At the time of writing, there are two Talk+ clients in a closed testing phase. One is an application for Google's Android operating system, and the other is developed for Apple's iOS platform. Clients for desktop operating systems are currently under development. The clients are all developed in the native programming language of their respective operating system, while they share one core component. This component contains the WebRTC core module from the open-source project presented in Section 2.1.2, and implements its own, non-standard, API on top of this core. The API may used by the various clients, the same way a web application would interact with the W3C standardized API.

The version of WebRTC core used in the Talk+ clients is a forked version from 2012. A forked version means that the code is copied into a separate project, and

maintained separately. Later changes in the original code will not automatically be transferred to the forked code. Figure 5.3 shows how the Talk+ clients are using a detached version of WebRTC. This workflow is well suited for Comoyo, as they do not need to worry about clients using different WebRTC versions. This control is lost when implementing a web browser version of the application. Although the browser vendors use the same core, their individual API implementations available to the web developers may not be consistent, as explained in Section 2.1.1.

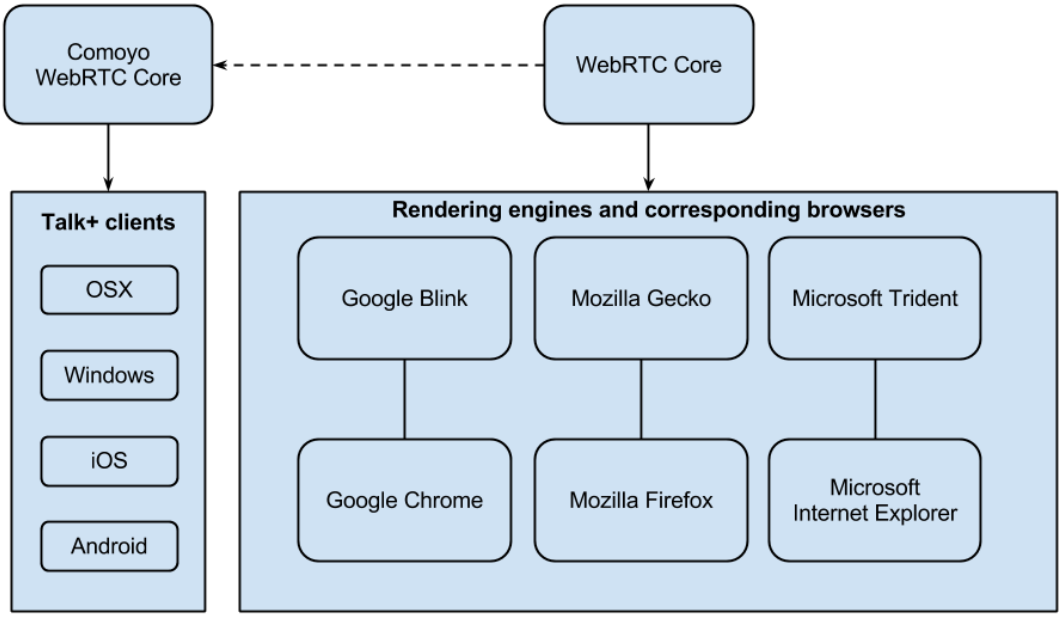

FIGURE 5.3: The WebRTC core module

5.2 Talk+ web application

This section discusses the technology choices for the browser-based implementation of Talk+. Chapter 6 details how these technologies are used and organized in the code.

5.2.1 Languages

Hyper-Text Markup Language

Hyper-Text Markup Language (HTML) is a formal markup language used to describe the content and structure of a website. It is defined by the W3C, and is accepted by all internet browsers[33]. All modern browsers support version 4 of HTML, but since 2007, W3C has been working on the next version, often referred to as HTML5. The HTML5 standard was completed in 2012, and several browsers have implemented most of the specification[34].

Cascading Style Sheets

Cascading Style Sheets (CSS) is used for styling of the various elements in the HTML[35]. This includes colors, positioning, hiding elements, to mention just a few of the uses of CSS. Chapter 4 specifies no requirements to the layout or visual style of the application. Therefore, only a small amount of CSS will be used in order for the application to have a visual expression.

JavaScript

When writing applications for use in browsers, the common approach is to use the JavaScript programming language. JavaScript is the native programming language across all modern browsers, and implements the EcmaScript standard[36]. The language has direct access to the browser's Web APIs, such as WebSockets and WebRTC, as well as the website's *Document Object Model* (DOM) for manipulating its content[37]. Due to its tight integration with the browser, JavaScript will be used in the implementation of this thesis' artifact.

Alternative methods of writing applications for browsers are to rely on third-party software in the browser, often referred to as plugins. Java Applets, Adobe Flash, and Microsoft Silverlight are examples of plugins which may work in the browser. The advantage of applications written for such software, is that they are, unlike browsers, developed by *one* vendor, and will therefore perform the same regardless of the browser. The software may also have access to deeper levels of

the user's operating system, such as hardware and external storage[38]. A largely debilitating factor of third-party plugins, is that they are not necessarily supported by the browser or operating system that the user is using. For example, Apple does not support the Flash or Silverlight plugins on their phones and tablets[39].

5.2.2 Authentication

Requirement R1 requires the application to allow authentication using Comoyo credentials, as the user will not be able to perform calls until authentication has taken place. The authentication process requires communication with the Comoyo signaling server. In order for the user not being required to log in each time he opens the application, there will also need to exist a way for the application to store credentials. Sections 5.2.3 and 5.2.4 will plan the signaling and storage components in the implementation.

5.2.3 Signaling

Section 5.1.1 establishes that the signaling server at Comoyo may communicate either through a TCP socket, or a WebSockets connection. Since TCP is not available as a web API (see Section 2.1.3), the Talk+ web application must rely on WebSockets as the transportation protocol for communicating with the signaling server. The WebSockets API is available in the latest version of the five most popular desktop browsers[40], and is therefore not expected to cause significant compatibility issues.

5.2.4 Storage

In the Talk+ web application requirements, detailed in Chapter 4, requirement R8 states that *The user shall be able to view the call log.* The call log of the user may, with extensive use of the application, grow to be very large. It would not make sense to pass this information on to the server at every request, while it still would

be preferable to have the information stored on the client in order to reduce load time when accessing the application. With this requirement in mind, and the fact that the browser compatibility for localStorage is very high[41], the conclusion is that localStorage should be used for all persistence in the Talk+ web application.

5.2.5 Media transfer

In Chapter 2 it became clear that the architecture of Talk+ on the web would need to use WebRTC for media transfer. In order for the application to exchange media with the existing Talk+ backend, there are no alternatives. One of the key differences between the use of WebRTC in the native clients and the web application, is that the native clients all share the same implementation of the API, while the web application will need to work on the most popular browsers, in which the API implementations are unique.

Chapter 6

Implementation

This chapter details the final implementation of the Talk+ web application. Section 6.1 starts by detailing the overall design of the system, and moves on to explain the different classes in detail.

6.1 Code

All client side code is available in Appendix A. Figure 6.1 shows all the classes in the project, how they are connected, as well as where the application interfaces with the different browser components. The arrows indicate where the instance of the class resides. The dotted lines from a class to a browser component specifies that the class is the owner of the component. The figure presents the goal that every browser component in use shall be directly handled by one responsible class. For example, if a class wants to send a message over WebSockets to the signaling server, the message must be passed to the ServerConnection class.

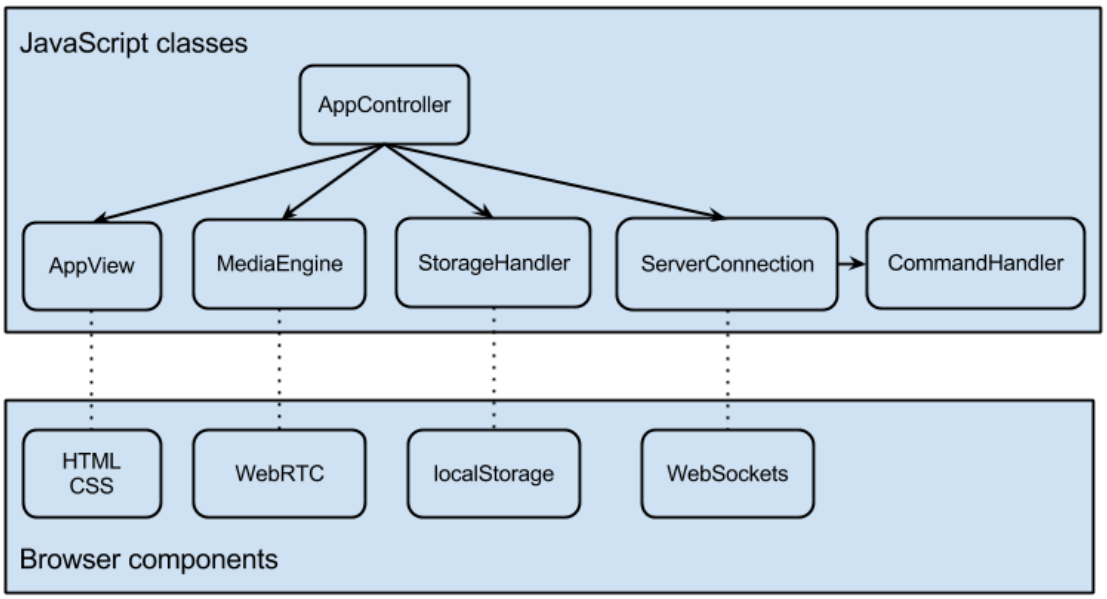

FIGURE 6.1: Talk+ JavaScript classes, and how they interact with the browser components

The implemented solution is a single-page HTML5-dependent web application. This means that the application only has one HTML file, and that all changes to the content or structure on the web page are achieved through the use of JavaScript. A single-page solution is beneficial for an application which relies heavily on JavaScript, since there are no page refreshes that will reset the state of variables. The HTML and CSS used in the application may be viewed in Appendix A.1 and A.2, respectively. Figure 6.2 shows the user interface for the implemented application.

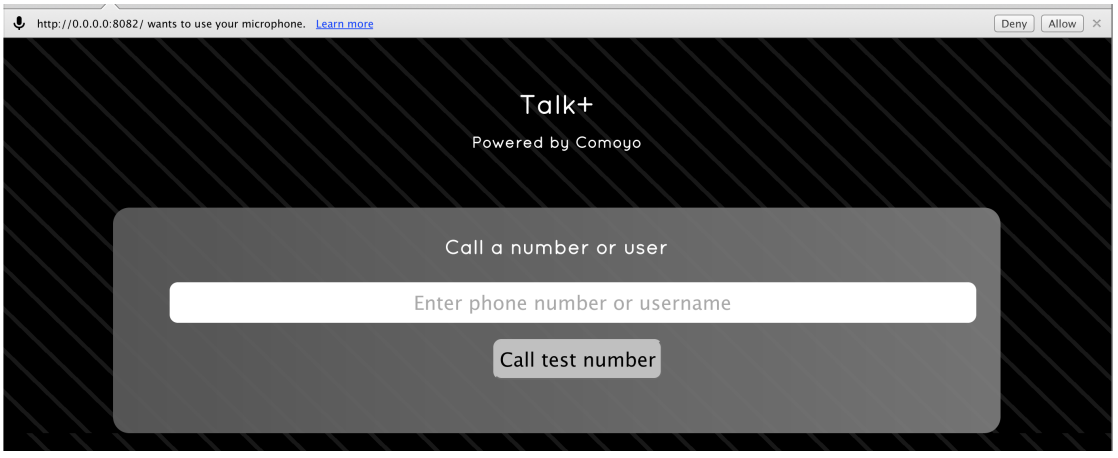

FIGURE 6.2: Talk+ web application

The classes described in this section work together in providing a working service to the user. In order to visualize how the classes are connected, two common use cases have been depicted as sequence diagrams. Figure 6.3 (page 42) shows the flow of data between classes during authentication. In order to simplify the figure, the storage class has been left out. Similarly, Figure 6.4 depicts the flow of events and data when a user initiates a phone call from the application.

6.1.1 AppController

The AppController class, listed in Appendix A.3, is the base class of the application. Its sole responsibility is to host the controllers for storage, server communication, and media transfer, as well as the application's view. The AppController is instantiated at the root level of the application. Most browsers treat the *window* object as the root object. Thus, when the AppController is instantiated, it is automatically attached to the *window* object.

6.1.2 StorageHandler

In Chapter 5, the decision was made to use localStorage for persistence. The StorageHandler class, listed in Appendix A.4, wraps around the localStorage API, while exposing the same functions for setting and retrieving data. The class extends localStorage with the ability to store some values in memory, in addition to on disk, for faster retrieval. In order to accomplish in-memory storage, the StorageHandler class adds the value to a local variable. When retrieving a value, the class first check if the value is already stored in a variable, and falls back to querying the localStorage.

6.1.3 AppView

The AppView is the only component of the application that is allowed to read or write to the DOM (see Appendix A.5). All other classes should call the appropriate

AppView function if they wish to access a DOM element, or if they wish to alter either the layout, structure, or content of the web page. The most important function of the class is to react to the user's actions. When the user logs in, submits a number, ends a call, or presses any other button, the AppView captures the event, and forwards the action to the appropriate class.

6.1.4 ServerConnection & CommandHandler

The ServerConnection class, shown in Appendix A.6, has several responsibilities. Its main responsibility is to maintain the WebSockets connection to the signaling server. This includes opening the connection, and sending regular empty messages to the server in order to keep the connection open. The class is also responsible for parsing incoming, and formating outgoing, messages. When another class wants to send a message on the socket, it must call the ServerConnection *sendMessage* function with a command name, and an object containing the message body. The function formats the command, and its content, in the structure required by the server, and emits the message on the socket.

When a message arrives on the socket from the server, the class' *onmessage* function will parse the message, and retrieve its command name and attached data. If a command name is successfully parsed, the function will try to invoke a function with the same name in the CommandHandler class. For example, if the command *com.telenor.sw.footee.common.th.ClientRegistrationResponse* is received, the *onmessage* function extracts the command name *ClientRegistrationResponse*, and checks if CommandHandler has the function *handleClientRegistrationResponse*. If the function exists, it is called with the data received from the server as a parameter.

In addition to handling the different incoming commands that are received on the socket connection to the server, the CommandHandler class (see Appendix A.7) is also responsible for building all outgoing commands, and forwarding them to the ServerConnection class.

6.1.5 MediaEngine

The MediaEngine class, shown in Appendix A.8, is responsible for all aspects of the WebRTC technology, making it a fairly large and complex class. The following section summarizes the code, and does so based on the class functions.

Create RTCPeerConnection

The W3C standard defines the interface *RTCPeerConnection*[3], which the MediaEngine instantiates to the variable *peerConnection*. As soon as the user is authenticated, and the client has received the TURN server IP and credentials, *peerConnection* instance is constructed. The *RTCPeerConnection* interface specifies several *EventHandlers*, which are functions developers may override. These functions are later called by the browser, if a corresponding event occurs.

One of the events specified in the W3C standard, *onicecandidate*, is invoked when a new ICE candidate is found. ICE candidates are, as defined in Section 2.1.2, sets of IPs and ports that are possible routes to the local client from a remote client. The *onicecandidate* EventHandler will also notify when the search for new ICE candidates is complete. The implementation of the Talk+ web application will not allow a call to be placed before all ICE candidates have been found. Therefore, if a call is placed before the ICE gathering is complete, the call information may be saved and automatically start when the application detects the last candidate.

Another important event handler that the implementation overrides, *onaddstream*, is called when a remote stream is added to the *peerConnection*. The implementation forwards this stream to the AppView, which initiates playback of the stream in the DOM.

Request microphone

After the *peerConnection* instance is created, the application will ask the user permission to capture audio from the computer's microphone. If the user grants

the application this permission, is continues by calling the MediaEngine class function *createOffer*.

Create offer

The *createOffer* function encapsulates the corresponding function on the *peerConnection* object. It starts by adding the microphone audio stream to the *peerConnection* object, which is responsible for maintaining both the local and remote streams. It then calls *peerConnection*'s *createOffer* function, and submits a success function as a parameter, which is fired if the offer is successfully created. The process of creating the offer is performed entirely the browser. The success function takes a *RTCSessionDescription* object as a parameter. This object contains information on whether this is an offer or an answer, as well as the SDP (see Section 2.1.1). The RTCSessionDescription object is added to *peerConnection* by passing it to its *setLocalDescription* function.

Initiating call

The process of initiating a call starts with the creation and sending of a signaling message. If the ICE gathering is complete, the *startCall* function creates a *offer* signaling message, and includes the SDP of the local session description. There are two other important attributes in an offer. The first attribute is named *offererSessionId*, and is a unique call identifier. The attribute is therefore incremented by the application if a new call is created. The second attribute is named *sequence*, and starts with a value of 1 for every new *offererSessionId*. If a new offer or answer is sent by the application during the same call session, the attribute is incremented by 1.

The format of the signaling message is described by the *RTCWeb Offer/Answer Protocol* (ROAP) protocol, which is the protocol recognized by the Comoyo signaling server[42]. ROAP used to be the standard signaling protocol in WebRTC, but has later been replaced by the JSEP (see Section 2.1.2), which allows for a more free formating of signaling messages[43].

Ending call

Ending a call is simpler than the initiation, but the signaling message depends on the state of the call. If the call has been initiated by an offer message, and the client has received a signaling message indicating that the recipient's phone is ringing, then the previously mentioned *offererSessionId* must be included in the message. If the call is initiated, and the recipient has answered, the call must be shut down using a *sessionToken*. This token is received by the application together with the signaling message stating that the call is answered.

Handle signaling message

All incoming messages from the Comoyo signaling server is decoded in the CommandHandler class, as passed to MediaEngine's *handleSignalingInstantMessage* function. In a Talk+ call flow, the Comoyo server will emit one of the following three message types: *RINGING*, *ANSWER*, or *SHUTDOWN*. The ringing message does not contain any other useful information, and is simply used by the MediaEngine to alert the AppView of a changed state. The same applies to the shutdown message. When a message of this type is received, the AppView class is signaled to stop the playback of remote audio. When an answer message is received, it contains the remote party's session description. This is submitted to the *peerConnection* instance. If the SDP is correctly parsed, the browser will automatically set up the media connection to the Comoyo server, and start transferring media.

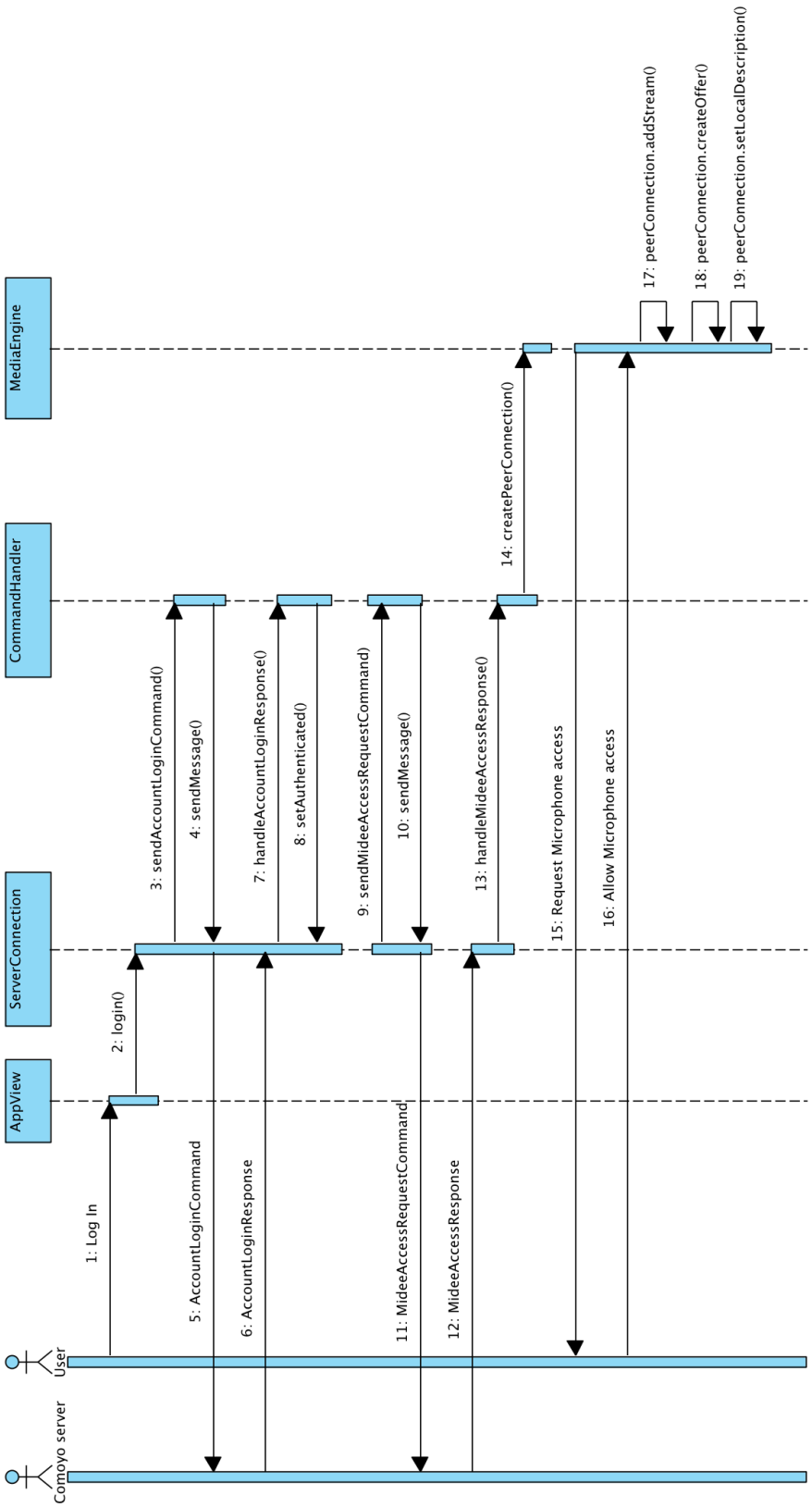

FIGURE 6.3: Talk+ authentication flow

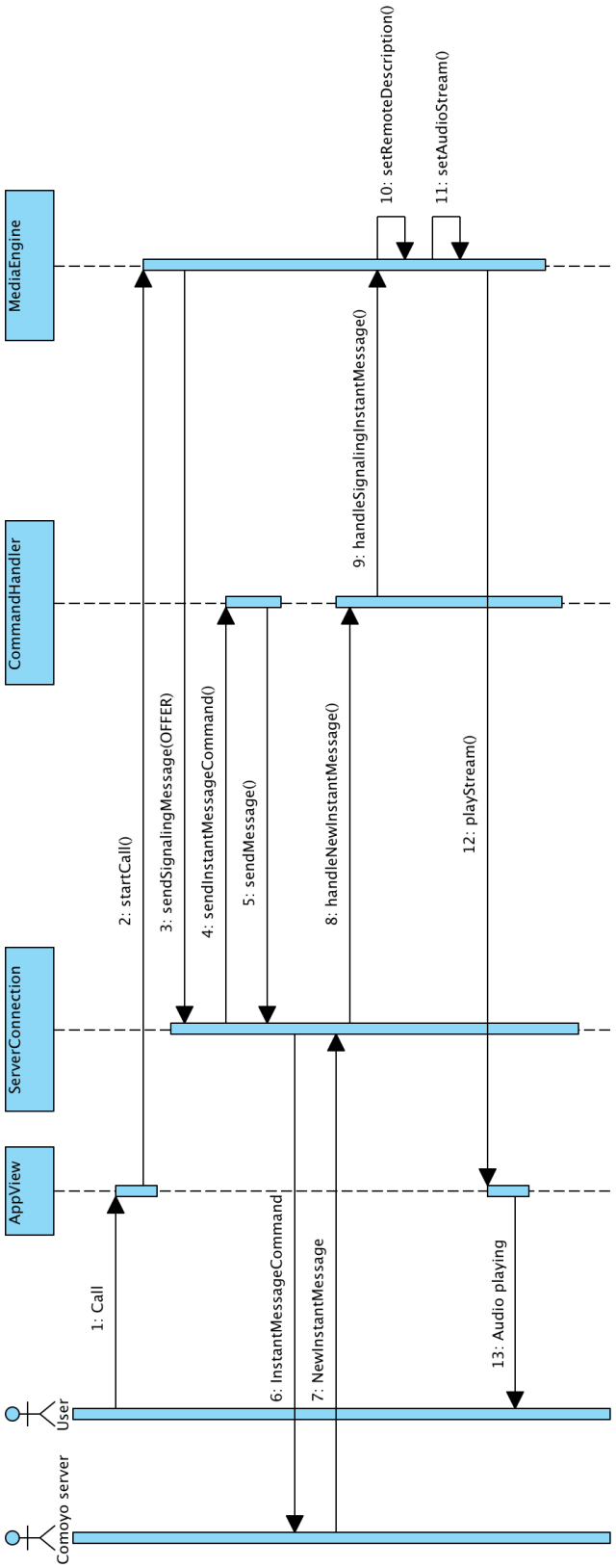

FIGURE 6.4: Talk+ call flow

Chapter 7

Evaluation

This chapter presents the evaluation of the implemented artifact. The evaluation is performed by determining which of the requirements stated in Chapter 4 that are fulfilled. The requirements are tested using automatic measuring where possible, and by using manual methods in the remaining cases. The results of tests involving measurements are documented in Appendix B.

The requirements are specified in Chapter 4, and are split into functional and non-functional requirements. This chapter addresses the requirements in the same manner, starting with the functional requirements.

After the evaluation of each requirement type, a table summarizes the results by using a color scheme. Green indicates that the requirements is fulfilled, while red indicates that it is unfulfilled. Yellow is used to indicate partial fulfillment of the requirement, or that some external factor impeded the implementation.

7.1 Functional requirements

R1 - The user shall be able to log in using Comoyo credentials
If the user is not logged in, the application displays a form in which the user may enter his or her credentials. The credentials are sent to the Comoyo servers for

verification. If the authentication is successful, the credentials form is hidden from the user.

R2 - The system shall be able to capture input from the user's microphone

By using the standardized web API *getUserMedia*, the application requests access to the user's microphone. The feature was tested, and found to be working in both Firefox and Chrome. An issue arose on systems where multiple input audio devices were available, and the wrong device was selected in the browser settings. After the setting was corrected, the issue was no longer observed.

R3 - The system shall be able to transfer audio from the user to the server

By using a series of manual tests, in which calls were placed to a cellular phone, it was verified that the client's microphone audio was transfered to the phone, via the Comoyo relay server.

R4 - The system shall be able to receive an audio stream from the server

By using the same test approach as R3, it was verified that the system received a remote audio stream from the Comoyo relay server.

R5 - The system shall be able to play back a received audio stream to the user

Requirement R5 is satisfied by the use of the HTML5 audio element. The browser is not aware of audio output capabilities, nor the system volume. Audio playback may therefore appear not to work on some systems. Correcting this issue is considered to be the user's responsibility, and the requirement is therefore fulfilled.

R6 - The user shall be able to perform an outgoing call to a Norwegian phone number

At the time of writing, the Talk+ is in a closed testing phase, disabling access to the service for normal Telenor subscribers. Given that the authenticated user possesses the correct calling rights in the Comoyo backend, this requirement is fulfilled. The first successful call from the web application was placed on March 20th, in the experimental version of Google Chrome. As of May 6th, two-way dialog also works in the stable version of Chrome.

R7 - The system shall inform the user when the server rejects a phone number

Whenever the server rejects the phone number, it returns a *shutdown* signaling message. However, the message does not contain any information about *why* the call was rejected. Various circumstances could cause a rejected call, such as insufficient credit balance, an improperly formatted number, a blacklisted number, or an internal server error. As a result, the web application only informs the user that the call ended, without specifying the reason. The application therefore only partially fulfills this requirement.

R8 - The user shall be able to perform an outgoing call to a Comoyo user

The server accepts Comoyo usernames in the same way that it does with phone numbers. Requirement R8 was therefore fulfilled by expanding the solution for R6, allowing users to supply text in the recipient input field.

R9 - The system shall inform the user when the server rejects a username

The server does not respond to non-existing usernames, leaving the application in a waiting state. This behavior is considered to be a fault in the server backend. One solution to the absence of response from the server, is to have an internal timeout in the system, notifying the user to verify the supplied username, and to try again. This alternative solution has not been implemented.

R10 - The system shall inform the user if an invalid phone number or username is entered

The user is only allowed to type in an 8-digit Norwegian phone number, or a Comoyo username. The phone number is validated by removing any non-digits, and subsequently testing if the length is eight digits. The test allows the number to be prefixed by the Norwegian country code.

The Comoyo username is required to be in the form of an email address. The format of an email address is defined by IETF[44], and the application verifies that the submitted username conforms to this standard. If the submitted callee is neither a valid phone number, nor an email, the application informs the user of the invalid input.

R11 - The user shall be able to place a call to a test number

The application contains a button labeled *Test call* that, when pressed, places a call to a predefined number. The purpose of the test number is to verify that a connection is successfully established, and that remote media is received.

R12 - The user shall be able to view the call log

As of May 2013, the Comoyo signaling service does not offer a command for retrieving the call log of a user account. It is up to the individual clients to save the log locally, as is practiced by both the Android and iPhone native clients. This presents an issue for the web application. One of the main advantages of a web application, is that it is available globally, on any device, as long as the browser supports the technologies used. However, locally persisted data will not be available between the browsers. Until the Talk+ service provides a centralized call log, the value of a call log in the Talk+ web application is very limited, and has thus been left out of the current implementation.

R13 - The application shall inform the user of how much the call is costing

The Comoyo user account used to test the service during development is exempt from call costs, and the server therefore does not announce any information about cost to the client. It has therefore not been possible to examine the cost information format normally sent by the server, and this has been left out of the current implementation.

R14 - The user shall be able to end the call

There are two states in which a call can be ended by the client. The first state occurs during the ringing phase, before the callee has answered. The second state occurs during an active conversation. Ending calls in both states are supported by the client.

Functional requirements overview

ID	Requirement
R1	The user shall be able to log in using Comoyo credentials
R2	The system shall be able to capture input from the user's microphone
R3	The system shall be able to transfer audio from the user to the server
R4	The system shall be able to receive an audio stream from the server
R5	The system shall be able to play back a received audio stream to the user
R6	The user shall be able to perform an outgoing call to a Norwegian phone number
R7	The system shall inform the user when the server rejects a phone number
R8	The user shall be able to perform an outgoing call to a Comoyo user
R9	The system shall inform the user when the server rejects a username
R10	The system shall inform the user if an invalid phone number or username is entered
R11	The user shall be able to place a call to a test number
R12	The user shall be able to view the call log
R13	The application shall inform the user of how much the call is costing
R14	The user shall be able to end the call

7.2 Non-functional requirements

F1 - The user shall be able to call using the three most popular internet browsers

As established in Section 4.3.2, the three most popular browsers today are Firefox, Chrome, and Internet Explorer. Each browser is individually evaluated below.

Microsoft has, at the time of writing, not implemented WebRTC in their browser, Internet Explorer. Critical to the W3C WebRTC standard, Microsoft is working on an alternative proposal, named *Customizable, Ubiquitous Real Time Communication over the Web* (CU-RTC-Web)[45]. In March 2013, Microsoft released a prototype implementation of CU-RTC-Web, as a plugin for Internet Explorer 10 in Windows, and Chrome on Mac OSX[46]. However, as CU-RTC-Web is an unofficial standard, support for Internet Explorer was not considered while developing the Talk+ web application.

When the implementation of the application began, the browser with the most complete implementation of the WebRTC API was Google Chrome. At the time, the most important difference between Chrome and Firefox, was that Chrome had support for the TURN protocol (see Section 2.1.2). Chrome was therefore the first browser to allow calling, and does so also in the latest stable release (Chrome 27).

Although Mozilla Firefox did not have support for TURN when the implementation started, it did have support for several of the APIs in the WebRTC standard, although they were turned off by default. With this in mind, supporting Firefox in the Talk+ web application was put on hold. After monitoring the activity in Mozilla's WebRTC team on a regular basis in the following months, TURN support in Firefox was added to their experimental nightly release in the last week of April 2013. At that time, the development phase of the thesis was considered to be concluded. Therefore, only minor attempts were made to adapt the application for Firefox. These attempts proved unsuccessful.

F2 - The user shall be able to hold a five minute call without losing connection

Several long conversations have been performed during the length of this project. In the first working implementation of two-way calling, there was a limit of two minute long calls imposed by the server. As soon as this limit was removed, several

five minute conversations with a cellular phone have been performed, without the loss of connection.

F3 - The participants shall not experience noticeable voice transmission delays

As detailed in Section 4.3.2, measuring the voice transmission delay requires more advanced equipment than deemed necessary for this thesis. A manual test, in which several phone calls were placed from Talk+ to a mobile phone, were compared to a series of calls made between two cellular phones. Talk+ was running in the browser on a broadband connection. No noticeable difference in delay was experienced. This delay is expected to increase if the internet connection is provided through a cellular network, or if the traffic on the broadband connection from other applications or computers is high.

F4 - The time from a call is started to media flow has begun, must not exceed 11 seconds

Three cases were considered: Calling the test number, an automated customer service number, and a cellular phone.

The test number proved to be the most responsive, with an average delay of 0,33 seconds. The technical details of the test number is unknown to the author, but an assumption can be made that the test call is optimized for fast response. The second test was conducted by calling the call centers of five large Norwegian businesses, all with automatic answering machines. All five answered within 3-5 seconds, with an average of 3,7.

The cellular phone proved to be the least responsive, with an average response time of 10.87 seconds. This test included the need for a human to manually accept the incoming call, which naturally added to the delay, although the call was accepted as soon as the phone was ringing. It is reasonable to assume that the cellular phone delay is caused by the increased complexity of call routing.

The requirement of 11 seconds was manually measured using phone calls over the cellular network. It is therefore reasonable to compare this delay, to the delay measured from the web client to a cellular phone. Both tests averaged at the same delay, and it is therefore concluded that the Talk+ web application performs within the required delay limit.

F5 - The system shall establish calls with a success rate of 90%

This requirement was tested by performing ten phone calls from the application to a cellular phone. In order to ensure thorough testing, all calls were placed without reloading the application, and the calls alternated between being ended on the phone and in the application. The test uncovered a flaw in the system, in which the user could not place a new call if the previous call was terminated on the phone. The issue was fixed, and further testing resulted in purely successful calls.

Non-functional requirements overview

ID	Requirement
F1	The user shall be able to call using the three most popular internet browsers
F2	The user shall be able to hold a five minute call without losing connection
F3	The participants shall not experience noticeable voice transmission delays
F4	The time from a call is started to media flow has begun, must not exceed 11 seconds
F5	The system shall establish calls with a success rate of 90%

Chapter 8

Discussion & Conclusion

This chapter begins by discussing the current support of the WebRTC standard, and attempts to predict the future of this technology. Next, the research questions from Chapter 1 are revisited, and then used to draw a conclusion to this thesis. The chapter concludes by detailing how Comoyo expects to proceed with the work performed in this thesis.

8.1 Discussion

According to usage statistics recorded by StatCounter, the top four browser vendors, Microsoft, Apple, Mozilla, and Google, claim 97 per cent of the browser market[31]. Out of these four, only the two latter are supporting the official WebRTC standard proposal. Microsoft, who possesses about 28 per cent of the market, believes the direction of the current standard is fundamentally wrong, and have therefore proposed, and partially implemented, their own standard. In the almost two years that the W3C has worked on the standard, Apple has not made an official comment on it.

Mozilla and Google, with their 60 per cent of the market, of which 80 per cent are using the latest browser version, are both working intensely on implementing the latest WebRTC draft in their browsers. Both vendors are very open about their

implementation progress, and have managed to generate some media attention during the Spring of 2013. In February, the companies co-authored a press release, announcing that their independent implementations of the WebRTC API were now able to communicate across their browsers. The news generated publicity for both companies, which they used to encouraged developers to start building applications that use the technology.

Microsoft is the owner of the VoIP service Skype, and therefore have an economic incentive in proposing a standard allowing them to bring their existing service to the browser. If that is the reason behind their counter-proposal, then it is not likely that they will retract it in the near future.

In my view, Microsoft is a key missing partner needed in order for WebRTC to become the de facto standard in browser media exchange. In order for Microsoft to shift its support, it is likely that one of two scenarios should occur. In the first scenario, Microsoft decides to retract their proposal, and start implementing the current proposed API, without any external factors forcing the decision. It is difficult to estimate the likeliness of this scenario, but from reading the discussions in the W3C Working Group, it does not seem imminent.

In the second scenario, in which Microsoft refuses to retract its proposal, Mozilla and Google must trust web developers to develop popular WebRTC services, while disregarding that their product will only be supported in about half of the potential customers' browsers. If one or more highly successful services are created with the WebRTC technology, the users of Microsoft and Apple's browsers are likely to put pressure on them to implement the same support. If they fail to listen to their users, they may see many of them migrating away from their browsers.

8.2 Conclusion

Chapter 1 establishes three research questions for evaluating the entry of traditional telecommunication operator services on the internet. The questions were to

be answered by creating a proof-of-concept telephony client for the use in desktop internet browsers. In this section, these research questions are addressed.

Is it technically possible to create the application?

In order to determine if the application was technically possible to create, a set of functional requirements were developed. Ten out of the fourteen requirements are considered fulfilled, one is not fulfilled, and the remaining three are either partially fulfilled, or their fulfillment is impeded by an external factor. Although not all the functional requirements have been satisfied, the application is able to establish a call from a browser to a phone on the PSTN, which has been the overall technical goal of this thesis.

Is the quality of the application good enough, compared to traditional telephony?

In Section 4.3, a set of non-functional requirements were developed to provide a qualitative evaluation of the application. Four out of the five requirements are addressing factors such as the call delay, voice quality, and stability of the application. These requirements are found to be fulfilled, and may therefore be considered equal to, or better than, the quality of a traditional phone call.

The remaining requirement addresses the user's freedom to perform calls from the three most common browsers. Out of the three, only two browsers are supporting the technology required to use the application. In the current state of implementation, the application will only work in one of these two browsers. This reduces the availability for a substantial number of users, and must be viewed as a considerable drawback for the quality of the application.

Is this a product that customers will use?

In its current state, Talk+ is not a service aimed at replacing the customer's phone. One of Telenor's strategies is to promote the iPhone and Android applications as services providing low calling rates when abroad. With the addition of a web

browser version of the product, traditional telephony is extended by allowing calls to be placed from a desktop computer. The implemented solution is to be regarded as a proof-of-concept application, and has therefore not been subject to any user acceptance tests. However, without the freedom to call from an arbitrary browser, it is difficult to expect a widespread use of the service.

8.3 Further work

Telenor Comoyo is investing considerable resources in the Talk+ service. After being presented with the results in this thesis, some thoughts were given on possible further work on a Talk+ browser application. In the long term, Comoyo's vision is to create a web-centric telephone that is able to receive incoming calls, browse the user's contact list, and eventually replace the traditional telephony application on cellular phones.

In the short term, the application should be reimplemented with the intent of creating a commercial product. The first step in achieving this is to create a set of user requirements, which will be used, together with a set of user acceptance tests, to determine if the product is ready for a commercial launch. Second, the application must be reimplemented with a focus on writing maintainable, as well as testable, code. The application should be browser agnostic, meaning that the development is not be targeted towards one specific browser. Lastly, visual and interaction designers should create a user interface design for the application, in order to provide an optimal user experience.

Appendix A

Source Code

A.1 index.html

```
<!DOCTYPE html>
<html>
<head>
    <link href="assets/css/app.css" type="text/css" rel="stylesheet">
    <title>Talk+</title>
</head>
<body>
    <header>
        <h1>Talk+</h1>
        <h3>Powered by Comoyo</h3>
    </header>
    <section id="content">
        <h1 id="call_indicator">Call a number or user</h1>
        <ol>
            <li>
                <input type="text" name="callee"
                       placeholder="Enter phone number or username"></input>
            </li>
            <li>
                <button id="testCall">Call test number</button>
            </li>
            <li>
                <button id="endCall" hidden>Hang up</button>
            </li>
        </ol>
        <form id="login">
            Email: <input name="username" type="email"></input>
            Password: <input name="password" type="password"></input>
            <input type="submit" value="Log in"/>
        </form>
    </section>
    <!-- The audio element will eventually contain
```

```
        the remote user's audio stream -->
    <audio id="phone" src="null"></audio>
</body>

<!-- Load scripts last for loading performance -->
<script src="assets/js/lib/jquery-1.9.1.min.js"></script>
<script src="assets/js/storageHandler.js"></script>
<script src="assets/js/commandHandler.js"></script>
<script src="assets/js/serverConnection.js"></script>
<script src="assets/js/mediaEngine.js"></script>
<script src="assets/js/appView.js"></script>
<script src="assets/js/app.js"></script>
</html>
```

A.2 app.css

```
@font-face {
    font-family: ComoyoRegular;
    src: url(../fonts/ComoyoQuickRegular.ttf?);
}

body {
    padding: 0;
    margin: 0;
    background: repeating-linear-gradient(45deg, black 4px, #303030 10px);
    font-family: ComoyoRegular;
    color: white;
}

header {
    position: relative;
    border-bottom: 30px green;
    width: 100%;
    margin: 0;
    padding: 50px 0;
}

h1, h3 {
    display: block;
    text-align: center;

}

body > section {
    background: linear-gradient(90deg, rgba(94, 94, 94, 0.9),
                                rgba(128, 128, 128, 0.9));
    padding: 20px;
    margin: 0 10%;
    border-radius: 20px;
}

body > section > form {
    text-align: center;
}
```

```
body > section > ol {
    display: block;
    list-style: none;
}

body > section > ol > li {
    font-size: 24pt;
    padding: 10px;

}

body > section > ol > li > label {
    float: left;
    width: 350px;
    height: 100%;
}

body > section > ol > li > input {
    box-sizing: border-box;
    border-radius: 10px;
    border: none;
    text-align: center;
    font-size: 18pt;
    width: 100%;
    height: 50px;
}

body > section > ol > li > button {
    font-size: 18pt;
    text-align: center;
    margin-left: 400px;
    border-radius: 10px;
    width: auto;
    height: 50px;
}
```

A.3 app.js

```
(function() {
    function AppController() {
        this.serverConnection = new ServerConnection();
        this.storage = new StorageHandler();
        this.mediaEngine = new MediaEngine();
        this.serverConnection.connect();

        this.appView = new AppView();
        this.callee = '';
    }

    app = new AppController();
})();
```

A.4 storageHandler.js

```
function StorageHandler() {
    this.sessionKey = null;
    this.clientId = null;
    this.userId = null;
    this.store = window.localStorage;
}

StorageHandler.prototype.setItem = function(key, value, callback) {
    this[key] = value;
    this.store.setItem(key, value);
    if (callback !== undefined) {
        callback();
    }
};

StorageHandler.prototype.getItem = function(key) {
    var item;
    // If we have already set this item, we use that one
    if (this[key] !== null && this[key] !== undefined) {
        return this[key];
    }

    // If the item is not set, we try to fetch it from storage
    item = this.store.getItem(key);
    if (item !== null && item !== '') {
        this[key] = item;
        return item;
    }

    // If the item is not set anywhere, we return null
    return null;
}
```

A.5 appView.js

```
function AppView() {
    var that=this;
    $('input[name="callee"]').keypress(function(e){
        if (e.which === 13) {
            app.callee = e.currentTarget.value;
            if (that.isValidInput(app.callee)) {
                e.preventDefault();
                console.log("Callee set to " + app.callee);
                app.mediaEngine.startCall();
            }
            else {
                that.stateChange('INVALID');
            }
        }
    });
```

```
    $('#login').submit(function(e){
        e.preventDefault();
        app.storage.setItem('username',
            e.currentTarget.elements.namedItem('username').value);
        app.serverConnection.login(
            e.currentTarget.elements.namedItem('password').value);
    });

    $('#testCall').click(function(e){
        app.callee = '+470047004700';
        app.mediaEngine.startCall();
        $('#testCall').hide();
    })

    $('#endCall').click(function(e){
        app.mediaEngine.endCall();
        that.stateChange('SHUTDOWN');
    })
}

AppView.prototype.isValidInput = function(callee) {
    // First, check if the input is a valid phone number
    var stripped_number = callee.replace(/[^0-9]+/g,'');

    // RFC 2822 email regex credit:
    // http://www.hacksparrow.com/javascript-email-validation.html
    var email_regex = RegExp(['^(([^<>()[\\]\\\\.,;:\\s@\\"]',
        '+(\\.[^<>()[\\]\\\\.,;:\\s@\\"]+)*)|',
        '(\\".+\\"))@((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]',
        '{1,3}\\])|(([a-zA-Z\\-0-9]+\\.)+[a-zA-Z]{2,}))$'].join(''),'i');


    if ((stripped_number.length === 10 &&
        stripped_number.split(47).reverse()[0].length === 8) ||
        (stripped_number.length === 8)) {
        return true;
    }
    // Continue to check if the input is an RFC 2822 standardized email
    else if (email_regex.test(callee)) {
        return true;
    }
    return false;
}

AppView.prototype.hideLoginForm = function() {
    $('#login').hide();
}

AppView.prototype.stateChange = function(state) {
    switch (state) {
        case 'RINGING':
            $('input[name="callee"]').prop('disabled', true);
            $('#endCall').show();
            $('#call_indicator').text('Ringing...');
            break;

        case 'ANSWER':
```

```
            $('input[name="callee"]').prop('disabled', true);
            $('#endCall').show();
            $('#call_indicator').text('Answered!');
            break;

        case 'PROCESSING':
            $('input[name="callee"]').prop('disabled', true);
            $('#endCall').show();
            $('#call_indicator').text('Connecting...');
            break;

        case 'INVALID':
            $('#call_indicator').text('Invalid number or username');
            break;

        case 'SHUTDOWN':
            $('#call_indicator').text('Call a number or user');
            $('input[name="callee"]').prop('disabled', false);
            $('#endCall').hide();
            $('#testCall').show();
            break;
    }

}

AppView.prototype.playStream = function(stream) {
    var phone;
    phone = document.getElementById('phone');
    phone.src = window.webkitURL.createObjectURL(stream);
    phone.play();
}

AppView.prototype.stopStream = function(stream) {
    var phone;
    phone = document.getElementById('phone');
    phone.pause();
}
```

A.6 serverConnection.js

```
function ServerConnection() {
    var that = this;

    this.authenticated = false;
    this.ws = null;
    this.commandHandler = new CommandHandler(this);

    this.ping = function() {
        setTimeout(function() {
            that.ws.send("\0");
            that.ping();
        }, 30*1000);
    }
};
```

```
ServerConnection.prototype.connect = function() {
    var that = this;
    this.ws = new WebSocket('wss://edgee-ws-api-staging.comoyo.com:443');

    this.ws.onclose = function() {
        console.log("Server connection closed");
    };
    this.ws.onopen = function() {
        var timeout;
        console.log("Server connection opened.");
        that.commandHandler.sendClientRegistrationCommand(that.sendMessage);
        that.ping();

    };
    this.ws.onmessage = function(message) {
        console.log("Server message received");
        // Handle incoming commands and relay the data
        var command, data, commandSuffix;

        if (message.data === "") {
            console.log("Received ping");
            return;
        }

        data = JSON.parse(message.data.substring(0, message.data.length - 1));
        commandSuffix = Object.keys(data)[0].split('.').reverse()[0];
        command = "handle" + commandSuffix;

        // Call the handle function for the incoming command, if it exists
        if (typeof(that.commandHandler[command]) !== "undefined" ||
            that.commandHandler.hasOwnProperty(command)) {
            that.commandHandler[command](data);
        }
    }
};

ServerConnection.prototype.close = function() {
    console.log("Closing edgee connection");
    if (typeof(this.ws) !== "undefined" || ws.hasOwnProperty('close')) {
        this.ws.close();
    }
};

ServerConnection.prototype.sendMessage = function(command, message) {
    var fullCommand, data;

    fullCommand = "com.telenor.sw.adaptee.th." + command;
    data = {}
    data[fullCommand] = message;
    this.ws.send(JSON.stringify(data));
};

ServerConnection.prototype.readyState = function() {
    return this.ws.readyState;
};
```

```
ServerConnection.prototype.login = function(password) {
    this.commandHandler.sendAccountLoginCommand(null, password);
};

ServerConnection.prototype.sendSignalingMessage = function(message) {
    this.commandHandler.sendInstantMessageCommand(null, message);
};

ServerConnection.prototype.setAuthenticated = function(authenticated) {
    if (authenticated === true) {
        this.commandHandler.sendSubscriptionCommand();
        this.commandHandler.sendMideeAccessRequestCommand();

        app.appView.hideLoginForm();
    }
};

function SignalingMessage() {
    this.contentBody = null;
    this.recipientUserName = app.callee;
    this.originatingUserName = app.storage.getItem('username');
    this.contentType = "application/sdp";
}
```

A.7 commandHandler.js

```
function CommandHandler(serverConnection) {
  this.serverConnection = serverConnection;
}

CommandHandler.prototype.sendSubscriptionCommand = function(callback) {
  var message = {
    subscriptionInformation: {
      subscribeToInstantMessagesOfType: 'application/sdp'
    }
  }
  this.serverConnection.sendMessage('SubscriptionCommand', message);
}

CommandHandler.prototype.sendServiceRequestCommand = function(callback) {
  var message = {
    serviceId: 'talkplus'
  }

  this.serverConnection.sendMessage('ServiceRequestCommand', message);
}

CommandHandler.prototype.sendClientRegistrationCommand = function(callback) {
  var message;
  if (app.storage.getItem('clientId') !== null) {
    this.sendAccountLoginCommand();
    return;
  }
  message = {
```

```
    clientInformation: {
      imsi:"talkplusweb",
      imei:"talkplusweb"
    }

  };
  this.serverConnection.sendMessage('ClientRegistrationCommand', message);
}


CommandHandler.prototype.handleClientRegistrationResponse = function(data) {
  var clientId =
  data['com.telenor.sw.footee.common.th.ClientRegistrationResponse'].clientId
    app.storage.setItem('clientId', clientId);
    this.sendAccountLoginCommand();

  }

  CommandHandler.prototype.sendAuthenticateSessionCommand =
  function(callback, message) {
    var message = {
      authenticateSessionInformation: {
        userId: app.storage.getItem('userId'),
        clientId: app.storage.getItem('clientId'),
        sessionKey: app.storage.getItem('sessionKey')
      }
    }
    this.serverConnection.sendMessage(
      'AuthenticateSessionCommand', message);
  }

  CommandHandler.prototype.handleAuthenticateSessionResponse =
  function(data) {
    var response =
    data['com.telenor.sw.footee.common.th.AuthenticateSessionResponse'];
    this.serverConnection.setAuthenticated(response.authenticated);
    if (response.authenticated) {
      console.log("Logged in!");
    }
    else {
      console.log("Could not authenticate with token, resetting");
      app.storage.setItem('userId', ''),
      app.storage.setItem('clientId', ''),
      app.storage.setItem('sessionKey', '')
    }
  }

  CommandHandler.prototype.sendAccountLoginCommand =
  function(callback, password) {
    var message;

    if (app.storage.getItem('sessionKey')) {
      this.sendAuthenticateSessionCommand()
      return;
    }
    message = {
      accountLoginInformation: {
```

```
      userName: app.storage.getItem('username'),
      password: password,
      clientId: app.storage.getItem('clientId')
    }
  }
  this.serverConnection.sendMessage('AccountLoginCommand', message);
}

CommandHandler.prototype.handleAccountLoginResponse = function(data) {
  var response = data['com.telenor.sw.footee.common.th.AccountLoginResponse'];

  this.serverConnection.setAuthenticated(response.loggedIn);

  app.storage.setItem('sessionKey', response.sessionKey);
  app.storage.setItem('userId', response.userId);
}

CommandHandler.prototype.sendInstantMessageCommand =
function(callback, message) {
  var message = {
    instantMessage: {
      contentBody: message.contentBody,
      recipientUserName: message.recipientUserName,
      contentType: message.contentType,
      originatingUserName: message.originatingUserName
    }
  }
  this.serverConnection.sendMessage('SendInstantMessageCommand', message);
}

CommandHandler.prototype.handleNewInstantMessage = function(data) {
  var instantMessage;

  instantMessage =
  data['com.telenor.sw.footee.common.th.NewInstantMessage']
  ['instantMessage'];

  app.mediaEngine.handleSignalingInstantMessage(instantMessage);

}

CommandHandler.prototype.sendMideeAccessRequestCommand =
function(callback) {
  var message = {
    request: {
      token: 'abc'
    }
  }
  this.serverConnection.sendMessage('MideeAccessRequestCommand', message);
}

CommandHandler.prototype.handleMideeAccessResponse = function(data) {
  var rtc, data;
  data = data['com.telenor.sw.footee.common.th.MideeAccessResponse'];
  app.mediaEngine.createPeerConnection(data.rtp)
}
```

A.8 mediaEngine.js

```
// Find the browser specific API
RTCPeerConnection = window.webkitRTCPeerConnection ||
  window.mozRTCPeerConnection || window.RTCPeerConnection;

navigator.getUserMedia = navigator.webkitGetUserMedia ||
  navigator.mozGetUserMedia || navigator.getUserMedia;

RTCSessionDescription = window.mozRTCSessionDescription ||
  window.RTCSessionDescription

function MediaEngine() {
  this.peerConnection = null;
  this.localMediaStream = null;
  this.callWaitingForIce = false;
  this.sessionId = 1460;
  this.sequence = 1;
  this.sessionToken = null;
  this.turnInfo = null;
};

MediaEngine.prototype.createPeerConnection = function(turnInfo) {
  var that=this, config;
  if (turnInfo !== undefined) {
    this.turnInfo = turnInfo;
  }
  config = {
    iceServers:
    [
    {
      url:'turn:'+that.turnInfo.server+':6768',
      credential: that.turnInfo.password,
      username: that.turnInfo.user
    }]
  }
  console.log(config);
  try {
    // As of May 11th, Chrome Stable does not support the username field
    // To leverage this, we fall back to the old format if the new fails
    that.peerConnection = new RTCPeerConnection(config);
  }
  catch (e) {
    config.iceServers[0] = {
      url:'turn:'+that.turnInfo.user+'@'+that.turnInfo.server+':6768',
      credential: that.turnInfo.password
    }
    that.peerConnection = new RTCPeerConnection(config);
  }


  that.peerConnection.onicecandidate = function(e) {
    // If the last candidate is null,
    // we know the ICE gathering has finished
    if (e.candidate === null) {
      console.log("ICE gathering complete")
```

```
      if (that.callWaitingForIce) {
        that.startCall();
      }
      return
    }
    console.log("icecandidate: " + e.candidate.candidate);
  }

  that.peerConnection.oniceconnectionstatechange = function(e) {
    if (that.peerConnection.iceConnectionState === "connected") {
      console.log("Connected to Talk+ relay!");
    }
  }

  that.peerConnection.onaddstream = function(e) {
    console.log("Remote stream added")
    app.appView.playStream(e.stream);
  }

  if (that.localMediaStream === null) {
    that.requestMicrophone();
  }
  else {
    that.createOffer();
  }
}

MediaEngine.prototype.startCall = function() {
  var that = this, contentBody, message;
  app.appView.stateChange('PROCESSING');

// Got all ICE candidates, let's send the offer
if (that.peerConnection.iceGatheringState === "complete") {
  this.callWaitingForIce = false;
  console.log("Sending offer");
  message = new SignalingMessage();
    contentBody = {
      messageType: "OFFER",
      sdp: that.peerConnection.localDescription.sdp,
      offererSessionId: ++that.sessionId,
      answererSessionId: '',
      seq: that.sequence
    }

    message.contentBody = window.btoa(JSON.stringify(contentBody));
    app.serverConnection.sendSignalingMessage(message);
    // console.log(contentBody);
  }
  else {
    console.log("Waiting for more ICE candidates")
    this.callWaitingForIce = true;
  }
};

MediaEngine.prototype.endCall = function() {
  var message, contentBody;
  console.log("Ending call");
```

```
  app.appView.stopStream();
  this.peerConnection.close();
  this.createPeerConnection();

  message = new SignalingMessage();
  contentBody = {
    messageType: "SHUTDOWN",
  }
  // If the call has started, supply the call session token
  if (this.sessionToken !== null) {
    contentBody.sessionToken = this.sessionToken;
  }
  // If the call has not started, supply the offerer session ID
  else {
    contentBody.offererSessionId = this.sessionId;
  }

  message.contentBody = window.btoa(JSON.stringify(contentBody));
  app.serverConnection.sendSignalingMessage(message);
  this.sessionToken = null;
}

MediaEngine.prototype.createOffer = function() {
  var that=this;
  console.log("Creating offer");

  this.peerConnection.addStream(this.localMediaStream);
  this.peerConnection.createOffer(function(sessionDescription) {
      console.log(sessionDescription);
      that.peerConnection.setLocalDescription(sessionDescription);
    }, function(e){}
  );
}

MediaEngine.prototype.handleSignalingInstantMessage = function(message) {
  var contentBody, sessionDescription;
  var that = this;
  contentBody = JSON.parse(window.atob(message.contentBody));
  console.log("Got " + contentBody.messageType)

  // Update the UI to reflect  the current state
  app.appView.stateChange(contentBody.messageType);

  switch (contentBody.messageType) {
    case 'RINGING': {
      break;
    }
    case "ANSWER": {
      console.log(contentBody);
      sessionDescription = new RTCSessionDescription({
        type: 'answer',
        sdp: contentBody.sdp
      });

      that.peerConnection.setRemoteDescription(sessionDescription,
        function(){
          console.log("Remote description set!");
```

```
          that.sessionToken = contentBody.setSessionToken;
        }, function(e){
          console.log("Remote description could not be set: " + e)
        }
        );

      break;
    }
    case "SHUTDOWN": {
          app.appView.stopStream();
          this.peerConnection.close();
          this.createPeerConnection();
          break;
        }
        default: {
          console.log("Unrecognized command: " + contentBody.messageType);
        }
      }
    };

    MediaEngine.prototype.requestMicrophone = function() {
      var that = this;

  // Request access to the microphone
  navigator.getUserMedia({audio: true}, function(localMediaStream) {
    that.localMediaStream = localMediaStream;
    that.createOffer();
  }, function(error){console.log(error);});
}
```

Appendix B

Tests

Test ID	Time to cell phone answer (sec)
TCP1	10.44
TCP2	10.43
TCP3	11.70
TCP4	10.51
TCP5	10.07
TCP6	11.97
TCP7	10.97
Average	**10.87**

Test ID	Time to test number answer (sec)
TT1	0.32
TT2	0.28
TT3	0.47
TT4	0.30
TT5	0.30
Average	**0.33**

Test ID	Phone to phone (sec)
P2P1	15
P2P2	10
P2P3	7
P2P4	7
P2P5	8
P2P6	16
P2P7	10
P2P8	15
Average	**11**

Test ID	Time to call center answer (sec)	Callee
TCC1	3.58	Telenor 09000
TCC2	4.33	Telenor 05000
TCC3	3.88	Storebrand
TCC4	3.64	Nordea
TCC5	3.14	Netcom
Average	**3.71**	

FIGURE B.1: Tests results

Bibliography

[1] Telenor. The Comoyo difference, May 2011. URL http://www.telenor.com/news-and-media/articles/2011/the-comoyo-difference/. [Last accessed May 29, 2013].

[2] Herald Sun. Internet messaging outnumbers SMS messages for the first time, May 2013. URL http://www.heraldsun.com.au/technology/smartphones/internet-messaging-outnumbers-sms-messages-for-the-first-time/story-fni0c1du-1226650844946. [Last accessed May 29, 2013].

[3] W3C. WebRTC 1.0: Real-time Communication Between Browsers, March 2013. URL http://dev.w3.org/2011/webrtc/editor/webrtc.html. [Last accessed May 29, 2013].

[4] W3C. W3C MISSION, 2012. URL http://www.w3.org/Consortium/mission. [Last accessed May 29, 2013].

[5] W3C. W3C Process Document, 10 2005. URL http://www.w3.org/2005/10/Process-20051014/tr. [Last accessed May 29, 2013].

[6] Internet Engineering Task Force. About the IETF. URL http://www.ietf.org/about/. [Last accessed May 29, 2013].

[7] WebRTC Initiative. WebRTC, 2012. URL www.webrtc.org. [Last accessed May 29, 2013].

[8] Harald Alvestrand. Google release of WebRTC source code, June 2011. URL `http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html`. [Last accessed May 29, 2013].

[9] W3C. WebRTC 1.0: Real-time Communication Between Browsers, August 2012. URL `http://www.w3.org/TR/webrtc/`. [Last accessed May 29, 2013].

[10] IETF. Javascript Session Establishment Protocol, February 2013. URL `http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03`. [Last accessed May 29, 2013].

[11] Mike Leber. Global IPv6 Deployment Progress Report, May 2013. URL `http://bgp.he.net/ipv6-progress-report.cgi`. [Last accessed May 29, 2013].

[12] IETF. IP Network Address Translator (NAT) Terminology and Considerations, August 1999. URL `http://tools.ietf.org/html/rfc2663`.

[13] IETF. Traversal Using Relays around NAT (TURN), April 2010. URL `http://tools.ietf.org/html/rfc5766`. [Last accessed May 29, 2013].

[14] Internet Engineering Task Force. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, April 2010. URL `http://tools.ietf.org/html/rfc5245`. [Last accessed May 29, 2013].

[15] W3C. Hypertext Transfer Protocol – HTTP/1.1, June 1999. URL `http://www.w3.org/Protocols/rfc2616/rfc2616.html`.

[16] IETF. The WebSockets Protocol, December 2011. URL `http://tools.ietf.org/html/rfc6455`. [Last accessed May 29, 2013].

[17] W3C. The WebSockets API, September 2012. URL `http://www.w3.org/TR/websockets/`. [Last accessed May 29, 2013].

[18] IETF. HTTP State Management Mechanism, April 2011. URL `http://tools.ietf.org/html/rfc6265`. [Last accessed May 29, 2013].

[19] W3C. Web Storage, April 2013. URL http://dev.w3.org/html5/webstorage/. [Last accessed May 29, 2013].

[20] International Telcommunication Union. 50 YEARS OF EXCELLENCE, 2006. URL http://www.itu.int/itudoc/gs/promo/tsb/88192.pdf. [Last accessed June 3, 2013].

[21] Skype. About Skype, 2013. URL http://about.skype.com. [Last accessed May 29, 2013].

[22] Joe Hallock. A Brief History of VoIP, November 2004. URL http://www.joehallock.com/edu/pdfs/Hallock_J_VoIP_Past.pdf. [Last accessed May 29, 2013].

[23] Nærings- og handelsdepartementet. Aktivt eierskap, April 2011. URL http://www.regjeringen.no/nb/dep/nhd/dok/regpubl/stmeld/2010-2011/meld-st-13-2010-2011/6/2/7.html?id=637192. [Last accessed May 29, 2013].

[24] Telenor. Norsk historie, 2013. URL http://www.telenor.com/no/om-oss/var-historie/norsk-historie/. [Last accessed May 29, 2013].

[25] IETF. SIP: Session Initiation Protocol, June 2002. URL http://www.ietf.org/rfc/rfc3261.

[26] Telefonica Digital. Telefónica launches TU Go in the UK, March 2013. URL http://blog.digital.telefonica.com/?press-release=telefonica-o2-tu-go. [Last accessed May 29, 2013].

[27] FierceWireless. Grading the top 10 U.S. carriers in the fourth quarter of 2012, March 2013. URL http://www.fiercewireless.com/special-reports/grading-top-10-us-carriers-fourth-quarter-2012? [Last accessed May 29, 2013].

[28] AT&T. The History of AT&T. URL http://www.corp.att.com/history/. [Last accessed May 29, 2013].

[29] Kevin Daly. WebRTC: The Power of AT&T Calling - In Your App, 2013. URL `http://developer.att.com/home/community/conference/ThePowerOfATTCalling-InYourApp.pdf`. [Last accessed May 29, 2013].

[30] Jinsoo Park Sudha Ram Alan R. Hevner, Salvatore T. March. Design Science in Information System Research. *MIS Quarterly*, 28(1):75–105, March 2004.

[31] StatCounter.com. Top 5 Browsers on May 2013. URL `http://gs.statcounter.com/#browser-ww-monthly-201305-201305-bar`. [Last accessed May 26, 2013].

[32] Jinsoo Park Sudha Ram Alan R. Hevner, Salvatore T. March. Transmission Systems and Media, Digital Systems and Networks, May 2003. URL `http://www.itu.int/rec/T-REC-G.114-200305-I/en`. [Last accessed May 29, 2013].

[33] W3C. HTML 4.01 Specification, December 1999. URL `http://www.w3.org/TR/html4/`.

[34] W3C. HTML5, December 2012. URL `http://www.w3.org/TR/html5/`. [Last accessed May 29, 2013].

[35] W3C. Cascading Style Sheets (CSS), May 2011. URL `http://www.w3.org/TR/css-2010/`. [Last accessed May 29, 2013].

[36] W3C. JAVASCRIPT WEB APIS. URL `http://www.w3.org/standards/webdesign/script`. [Last accessed May 29, 2013].

[37] W3C. Document Object Model (DOM), January 2009. URL `http://www.w3.org/DOM/`. [Last accessed May 29, 2013].

[38] Dan S. Wallach Chris Grier, Samuel T. King. How I Learned to Stop Worrying and Love Plugins, 2009. URL `http://w2spconf.com/2009/papers/s1p1.pdf`. [Last accessed May 29, 2013].

[39] Steve Jobs. Thoughts on Flash, April 2010. URL `http://www.apple.com/hotnews/thoughts-on-flash/`. [Last accessed May 29, 2013].

[40] Can I use... Can I use Web Sockets?, April 2013. URL `http://caniuse.com/websockets`. [Last accessed May 30, 2013].

[41] Mozilla Developer Network. DOM Storage guide, May 2013. URL `https://developer.mozilla.org/en-US/docs/Web/Guide/DOM/Storage`. [Last accessed May 29, note =.

[42] IETF. RTCWeb Offer/Answer Protocol (ROAP), October 2011. URL `http://tools.ietf.org/html/draft-jennings-rtcweb-signaling-01`. [Last accessed May 29, 2013].

[43] IETF. [rtcweb] Draft minutes for 82.5 meeting, February 2012. URL `http://www.ietf.org/mail-archive/web/rtcweb/current/msg03435.html`. [Last accessed May 29, 2013].

[44] Internet Engineering Task Force. Internet Message Format, April 2011. URL `http://tools.ietf.org/html/rfc2822`. [Last accessed May 29, 2013].

[45] Microsoft. Customizable, Ubiquitous Real Time Communication over the Web (CU-RTC-Web), August 2012. URL `http://html5labs.interoperabilitybridges.com/cu-rtc-web/cu-rtc-web.htm`. [Last accessed May 29, 2013].

[46] Microsoft. CU-RTC-Web Roaming, 2013. URL `http://html5labs.interoperabilitybridges.com/prototypes/cu-rtc-web-roaming/cu-rtc-web-roaming/info`. [Last accessed May 29, 2013].