Arindra Kumar Das

Web-based real-time communication based collaboration: An evaluation study of the DataChannel

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 08.03.2014

Thesis supervisor:

Prof. Jörg Ott

Thesis advisors:

Ph.D. (Tech.) Jouni Mäenpää

B.Sc. (Tech.) Tomas Mecklin

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Arindra Kumar Das

Title: Web-based real-time communication based collaboration: An evaluation study of the DataChannel

Date: 08.03.2014 Language: English Number of pages:0+72

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor: Prof. Jörg Ott

Advisors: Ph.D. (Tech.) Jouni Mäenpää, B.Sc. (Tech.) Tomas Mecklin

Real-time communication evolved from the traditional telephony network to Internet-based audio/video conferencing tools and instant messaging software. Third-party plug-ins and extensions added the same capabilities to Web-based browser applications.
Vendor-specific heterogeneity in software and hardware requirements brought up interoperability issues. There were no common set of communication protocols. The Web browser as a universal access point to the Web was facing a challenge in real-time communication.
To natively support real-time communication in the Web browser, Web-based Real-Time Communication (WebRTC) was introduced. It provides three main features i.e. access to the user's hardware, peer-to-peer connection between the browsers and arbitrary data exchange through its DataChannel application programming interface (API). These features are added in the browser as one extra protocol stack that can be accessed by a Web application with JavaScript API's. The DataChannel API provides several use cases and it is interesting to examine the behavior of the WebRTC DataChannel from the point of view of developers, enterprises, Web services and service providers. In this thesis we investigate the WebRTC DataChannel API. We design and implement two prototypes of a WebRTC based collaboration application. These were based on the respective specification by the Google Chrome and Mozilla Firefox browsers. A case study is proposed for data communication between two collaborative environments one consisting of an enterprise tool and the other being a non-enterprise tool based on WebRTC DataChannel and IP Multimedia Subsystem. The performance evaluation of the implementation is also carried out by comparing various other tools in a controlled environment.

Keywords: WebRTC, DataChannel, Real-time communication, Enterprise, Collaboration, IP Multimedia Subsystem

When you first start off trying to solve a problem, the first solutions you come up with are very complex, and most people stop there. But if you keep going, and live with the problem and peel more layers of the onion off, you can often times arrive at some very elegant and simple solutions. Most people just don't put in the time or energy to get there.

Steven Paul Jobs

Acknowledgements

Firstly, I would like to express my sincere gratitude to Professor Jörg Ott for being my supervisor and providing me guidance during the period of this thesis.

I am deeply indebted to Tomas Mecklin, my primary instructor at Ericsson Labs, for his enriching thoughts and sharing his profound knowledge. Without his support and encouragement I would not have been able to complete this thesis. I am extremely grateful to Jouni Mäenpää, my manager at Ericsson Labs, for giving me this opportunity to work at Ericsson's Nomadic Lab and his invaluable support through the whole thesis. I have been extremely lucky to have him as my second instructor who apart from normal working hours have also put extra hours during the weekends to go through my thesis for numerous times. Finally, My sincere thanks to all the Nomadic Lab staff especially Salvatore Loreto, for providing me with many valuable research papers required during my thesis.

I extend my heartfelt thanks to the Open source community in general and especially to Jesús Leganés Combarro, Muaz Khan, Sam Dutton and Justin Uberti for contributing rich resources to WebRTC and demos. Thanks to all my friends who have probably read this thesis even more times than me. Thanks for your dedication in helping me.

I owe a big thanks to every single individual who was directly or indirectly involved in completing my thesis. Last but not the least, the first step into this beautiful world of knowledge has been gifted to me by my parents. Thanks for everything mom and dad. I love you.

Otaniemi, 08.03.2014

Arindra Kumar Das

Abbreviations and Acronyms

3GPP	3rd Generation Partnership Project
AJAX	Asynchronous JavaScript and XML
APIs	Application Programming Interfaces
AS	Application Server
ATM	Automated Teller Machine
B2BUA	Back-to-back User Agent
Blob	Binary large object
CSS	Cascading Style Sheets
CSCF	Call Session Control Function
DOM	Document Object Model
DTLS	Datagram Transport Layer Security
ECMA	European Computer Manufacturers Association
GUI	Graphical User Interface
HSS	Home Subscriber Server
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
JXTA	Juxtapose
LAN	Local Area Network
ICE	Interactive Connectivity Establishment
IMS	IP Multimedia Subsystem
IM	Instant messaging
NAT	Network Address Translator
OS	Operating system

PGM	Presence and Group Management
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RTP	Real-time Transport Protocol
SASL	Simple Authentication and Security Layer
SCTP	Stream Control Transmission Protocol
SIP	Session Initiation Protocol
SIMPLE	Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions
SS7	Signalling System No. 7
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TURN	Traversal Using Relays around NAT
UAC	User Agent Client
UAS	User Agent Server
UCOPI	Unified Communications Open Interoperability Program
UDP	User Datagram Protocol
VoIP	Voice over IP
WebRTC	Web Real-Time Communication
WWW	World Wide Web
XHR	XML HTTP Request
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XOR	eXclusive OR

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"WebRTC and HTML5 could enable the same transformation for real time that the original browser did for information" - Phil Edholm

Communication and collaboration are the two main aspects in the growth of human civilization. The Invention of the telephone set an important milestone in the history of communication. Over the last 50 years, we have witnessed the development of many such technologies that have made people connect, share and communicate from miles apart. One of the greatest technological developments is the Internet.

Since its birth, the Internet has gone through a series of reformations. Many technologies have contributed to the development and success of the Internet. One of the most significant contributions to the Internet is the World Wide Web (WWW), commonly known as the Web [76].

The Web is accessed by web browsers [73], installed in client machines (generally, the computer system that makes a request is called the client and the responder to the request is called the server). The web browser is used to retrieve, transmit and traverse information resources on the Web. Web pages reside on servers, which are shown as a response to a request by the client via the 'HyperText Transport Protocol' (HTTP)[66].

Web pages are written in a language known as 'HyperText Markup Language' (HTML)[94] ,which provides rules and the basic building blocks on how to structure and present static content on a web page. Web pages are styled with Cascading Style Sheets (CSS). JavaScript provides dynamic behaviour to web page. JavaScript is a lightweight, interpreted, object-oriented scripting language. The introduction of Extensible Markup Language HTTP Request(XHR) has made JavaScript more dynamic and faster by reloading only the changes within a webpage. HTML manages CSS and JavaScript by the DOM convention (Document Object Model).

HTML5 [72], the latest version of HTML, is still under development. It provides an extension to HTML by introducing new elements and 'application programming interfaces' (APIs)[1] that along with JavaScript are capable of delivering rich web applications.

Web browsers have some basic built-in functions such as fetching and handling external contents, parsing HTML and XML and include also a JavaScript engine. To cope with new technological developments, browsers are continually being updated. The latest updates have added support for HTML5 and its APIs across various devices. These features have encouraged web-based application development to compete with its native counterpart [44].

The development of the browser has added a new dimension to web applications. Traditionally, plug-ins, extensions, add-ons etc.,(these are small pieces of software that add new features to a browser) were used to provide a real-time communication capabilities, but they are vulnerable to all sorts of security issues. For example, installing an extension might introduce vulnerabilities.

Recently, a new technology has been introduced and has already been implemented in the latest versions of major browsers such as Google Chrome, Firefox and Opera. Its aim is to bring real-time communication to the open web platform by simple JavaScript API's [62]. This technology is known as 'Web Real-Time Communication' (WebRTC)[19]. It is still in an initial stage and its standards are currently being drafted by the 'World Wide Web Consortium' (W3C)[23] in collaboration with the Internet Engineering Task Force (IETF)[6][44].

One of the major shifts in the evolution of telephony preceding WebRTC has been the move to IP transport for voice and video telephony. This shift is represented by IP Multimedia Subsystem (IMS). Developed by the 3rd Generation Partnership Project (3GPP),IMS is an infrastructure that provides multimedia services that can run on top of any transport medium that provides IP transport capabilities. [52].

1.1 Problem Statement

With enterprises becoming more global, the Internet reaching remote areas and devices becoming cheaper, the need for a universal collaboration tool is very important. Andrew McAfee coined the term "Enterprise 2.0" [87], which describes how Web technologies can be used within enterprises for collaboration, knowledge sharing and communication. Many tools have emerged that make collaboration more powerful and convenient. Due to vendor specific enterprise collaboration tools like Microsoft Lync, Cisco Unified Communication, etc. and a variety of access devices, the integration of new tools into the current infrastructure has become difficult.

A new kind of web-based real-time collaboration tool is therefore needed; one that is more interactive and responsive, free from plug-ins, is universal and can easily integrate with existing tools. Implementation of WebRTC in future web applications along with IMS as an infrastructure might change the way people use smart phones

[1]An API, or application programming interface, specifies the ways a program can interact with an application

and other communication tools. It is now assumed that the enhancement of the browser and the ongoing process of adding real-time communication to the web platform will combine communication and data. This will create opportunities for enterprises, service providers, Web services and application developers to improve the user experience.

WebRTC comprises of three important API's: getUserMedia, PeerConnection and DataChannel. Among these, the DataChannel API is responsible for the exchange of non-media data. Since the standardization process of the WebRTC API is still ongoing, there are several open questions that still need to be answered. For example, will it still be feasible to integrate the DataChannel API within the existing enterprise collaboration tools that have issues like security, poor user experience and performance? Will IMS be the next generation of infrastructure that can support real-time multimedia technologies? How can the collaboration tool be accessed from different devices? Will it be scalable and interoperable? We will attempt to provide answers in this thesis.

1.2 Research objectives and scope

The main objective of this thesis is to research existing multimedia technologies, investigate the WebRTC DataChannel and to learn how it is used to integrate with existing enterprise collaboration tools.

A prototype collaboration tool will be implemented to show how data can be transferred via WebRTC enabled browsers using the WebRTC DataChannel API. Further, the performance of the prototype will be analysed by comparing its performance to some popular collaboration tools that exist today.

To understand the interoperability, a case study has to be carried out and for this, an architecture will be designed. This architecture will demonstrate the interoperability of Microsoft Lync and a native WebRTC client (in a browser) to transfer peer to peer data via IP Multimedia Subsystem (IMS).

1.3 Structure of the thesis

This thesis consists of 5 chapters, structured as follows: Chapter 2 provides a background regarding this thesis along with related developmental work. Chapter 3 gives an overview about Web based multimedia technologies that are used in the implementation of our collaboration tool. Chapter 4 discusses the design and implementation of the proposed collaboration tool and describes the case study which proposes to integrate a WebRTC enabled browser and enterprise collaboration tools with IMS as the infrastructure. Chapter 5 presents a comparative study of various file transfer collaboration tools compared to our own implementations. Finally, Chapter 6 concludes by summarizing our thesis and listing future work items.

Chapter 2

Background

In this chapter a literature review for this thesis is presented. Collaboration tools and their importance in the context of an enterprise are first described. Various communication architectures and related technologies and protocols upon which these collaboration tools are based are descibed next. The concept of IP multimedia subsystem (IMS) is then described. This will be used later in this thesis for the proposed case study (Chapter 5).

2.1 Collaboration and enterprise

Collaboration is the process and methods used where individuals work together towards a common goal. Research has shown that collaborative work is more productive compared to working alone [112]. Remote and virtual collaboration in the Internet age has been made possible by various collaboration tools that facilitate easy and efficient communication for individuals distributed over various locations and in different time zones. Some of the early collaboration tools included email, bulletin boards and Internet Relay Chat (IRC). These are often referred to as groupware [99].

Based on time, collaboration is divided into synchronous and asynchronous. Synchronous collaboration tools require a team to work at the same time. Examples include instant messaging clients, application sharing software and whiteboards. Asynchronous tools allow teams to work at different times. Example include email, bulletin boards and weblogs.

Based on the communication architecture, collaboration is divided into three major architectures i.e. client-server, peer-to-peer and hybrid architectures. In the client-server architecture, collaboration related data is stored on the server. Clients usually log on to the server using their native applications. On the other hand, peer-to-peer architecture involves direct collaboration between clients. FilesOverMiles, JetBytes and PipeBytes are some peer-to-peer architecture based collaboration tools for sharing files [85]. Hybrid architecture is the third kind of architecture that is the combination of peer-to-peer and client-server architectures. The most common example is Skype. We will discuss these architectures in more detail Section 2.2.

Collaboration is very important in an enterprise, because apart from the importance discussed earlier, an enterprise needs to build a corporate culture, train new employees and bring employees from various sections into contact with each other and give them access to the information they need to accomplish their tasks.

Enterprises already have plenty of communication media such as e-mail, instant messaging, Intranet, telephones, software for document sharing & knowledge management. Even so, they are looking for new techniques of collaboration to transfer knowledge, ideas and views more efficiently and in a way that was not previously possible.

Enterprises generally see technologies for communication as falling within two categories. i.e. channels and platforms. Channels are meant to create and distribute digital information. Channels can be emails and instant messaging. Platforms are more public and widely visible like an Intranet, corporate websites and information portals [88].

Many concepts were developed to facilitate collaboration. The initial enterprise collaboration tools included applications like Lotus notes, KM Systems etc. KM Systems also designed a database to capture human knowledge by receiving "brain dumps" to gather employee experiences, expertise, learnings, insights and other types of knowledge [87].

2.2 Communication architecture

Over the years more complex models and architectures were developed to support the highly collaborative nature of the Internet. A generic web application is capable of five basic functionalities: browsing, reading, editing, adding and deleting [79]. These actions are performed by request and response methods. Collaboration via the Internet today can happen in many ways such as distributed computing, collaborating in creating media or software, conversing online and organizing people into online communities [46].

The network architectures of collaborative tools found today fall under two categories i.e. client-server models and peer-to-peer (P2P) architectures. Another architecture, the hybrid architecture, is used in many collaborative tools today to combine the advantages of the above architectures. The World Wide Web (WWW) is based on a client-server architectural model, although originally, it was assumed to be like a peer-to-peer (P2P) model [51]. We will discuss these architectures in more detail in the following subsections.

2.2.1 Distributed systems: Peer-to-peer architecture

During the last few years, peer-to-peer architecture-based applications such as distributed file systems have been contributing a large fraction of traffic on the Internet [114]. Features like fault tolerance, availability, scalability and performance are the reasons that have added to the growth of distributed file sharing systems compared to traditional centralized file systems [113].

'Peer' is a term used to refer to the participating computer system in a peer-to-peer model. Therefore, the peer-to-peer model (figure 2.1) refers to direct communication between two computer systems, where the computer system or peer that sends a request behaves as the client and the responding peer behaves like a server. This makes the peer able to act like both a client and server according to the behaviour it exhibits during communication [46]. All peers are of equal standing in the network [114]. The infrastructure is created as a sum of peers and their contributing resources, thus maintaining a decentralized infrastructure. The member peers connect dynamically and are ad-hoc in nature [104].

Due to the autonomous and open nature of peer-to-peer applications, they sometimes face challenges in providing the levels of availability, privacy, confidentiality, integrity and authenticity that are often required. Apart from this they also have the challenge of figuring out a mechanism for organizing the peers in such a way that they can cooperate [104]. However, more complex peer-to-peer applications might include provisions for security, anonymity, fairness, increased scalability and performance, as well as resource management and organization capabilities [43].

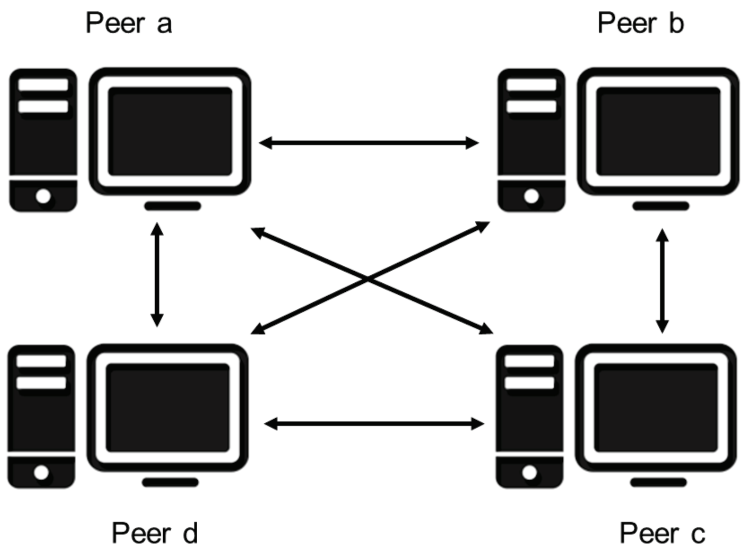

Figure 2.1: Peer-to-peer model

Peer-to-peer applications are classified into various categories: collaboration, distributed computation, platforms and content distribution. These are briefly discussed below [55]

- **Collaboration:** The application provides an infrastructure for peer-to-peer communication. This includes real-time communication and collaboration between peers such as chat/IRC and instant messaging [43].
- **Distributed computation:** This helps in load sharing between peers to enable the entire system to provide high processing power (CPU cycles). A distributed computation system distributes big tasks into small units and returns the results after execution.
- **Platforms:** A P2P platform gives the application developer leverage by providing an infrastructure that provides peer-to-peer functionalities upon which applications can be built. An example of such a platform is JXTA (Juxtapose) [70].
- **Content distribution:** Most peer-to-peer applications that run over the Internet are typically content distribution applications. These applications act as platforms that allow peers to function as distributed storage mediums by contributing, searching, and obtaining digital content. The main advantages of such applications are their ability to function, scale, and self organize without the need of a central server or management. Examples of such systems include Napster and Bit torrent.

2.2.2 Client-server architecture

The client-server based model (figure 2.2) comprises two components: a server providing services and a client that makes requests for these services. A single server can provide services to many clients that request its services.

A client is the user-oriented computing system that usually has a graphical user interface (GUI) for the users to access, operate and interact with. Each process that runs on the client is represented as an individual resource and all the operations are facilitated by the operating system (OS). Examples include desktop computers, tablets, mobile phones, laptops, Automated Teller Machines (ATMs) etc. [83]

A server is a computing system that acts upon the requests from clients by delivering the requested service in the form of responses. It also acts as a centralized system to relay information between two clients. It handles many logical functions to seamlessly deliver the requested resources. It can also act as client to another server to execute certain tasks. Examples include the Google servers, the Amazon servers, sensors, actuators etc. [83]

Client-server architectures have many advantages. The system can be easily scaled to support clients by adding high performance server hardware (but network bottlenecks should be taken into consideration) [83]. These architectures are considered secure because they are run on well maintained servers with redundancy mechanisms [113].

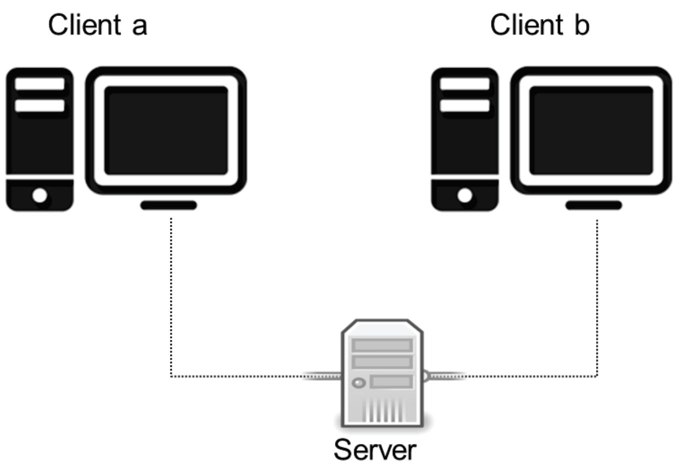

Figure 2.2: Client-server model

2.2.3 Hybrid architecture

The hybrid architecture comprises of elements from both client-server and peer-to-peer architectures. In this architecture the exchange of a file is usually a direct communication between the peers or clients. A server is generally used to discover and setup a connection between the peers. Further, the server is used to make the application more robust, as it caches the updates temporarily. These cached updates that can be used in case of a failure or when the users are temporarily unavailable. [113].

Napster and Skype are examples of hybrid architecture based applications. In Napster, a server node indexes the files from a set of users in a centralized manner. Users can search for files from a server and download them directly from the holding peer.

2.3 Network Address Translator (NAT) Traversal mechanisms

An Internet Protocol (IP) address is a unique address which is used to identify hardware interfaces on a network. The IP address follows the Internet Protocol standard. IP addresses are categorized as public and private. A public IP address is an address used to access devices across the Internet or other public networks. Public IP address are globally unique. A private IP address is an address used to access devices in a local network. Private addresses are reusable across different local networks but cannot be used in public networks.

Originally, IP addresses were designed to be globally reachable. Due to extensive use of the Internet, IP addresses were in shortage [116]. The major problems were address depletion and scaling in routing. To address these problems, short-term and long-term solutions were devised. The long-term solution was a proposal for a new Internet Protocol version (IPv6) with more addresses that would solve the problem of IP address depletion. The short-term solution was to reuse IP addresses. This would be done by a process called Network Address Translation (NAT) [64].

Network address translation (NAT) is a mechanism to connect a local network to the public Internet by mapping (translating) a private IP address to a public IP address. NAT hides an entire address space behind a single public IPv4 address, making it possible for many clients behind the NAT to share the same public IPv4 address. NAT hides the host behind it, thereby restricting external hosts from reaching it. This provides security. NAT can support one or several home networks behind the same public IPv4 address [116]. Furthermore, the Internet is a public network that is not secure and cannot be trusted. An internal network connected to the public Internet allows the anyone to access and interact with the internal network, giving them an opportunity to attack it. To protect the internal network from external threats, an intermediate system between the networks was introduced. This intermediate system that prevents unauthorized communication is called a **firewall**. It can also be used by an organization to enforce a security policy on the traffic between its internal network and the Internet [90].

Both a NAT and a firewall can create serious problems in peer-to-peer networks. In the client-server network architecture, the NAT and firewall are not an issue because the client initiates the session to a globally reachable IP address. In the peer-to-peer (P2P) network architecture, both peers may be hidden behind NATs. P2P networks, due to their nature, are vulnerable and by default firewalls will block untrusted incoming requests from other peers. This makes P2P communication impossible, without the use of additional mechanisms [9] [67].

STUN (Session Traversal Utilities for NAT) is a NAT traversal protocol. It is used by a peer to determine the IP and port address assigned to it by the NAT for its corresponding private IP address and port. It is also used for connectivity checking between peers [97].

STUN is a client-server protocol that supports two kinds of transactions: request/response transactions, where a client sends a request to the server and the server responds, and indication transactions, where either the client or server sends an indication that generates no response [97].

A STUN message consists of a fixed header that includes a method, a class, and the transaction ID. The method indicates which of the various requests or indications the STUN message is. The class indicates whether the message is a request, a success response, an error response or an indication. The transaction ID is a randomly selected 96-bit number. For request/response transactions, the transaction ID identifies the server's response to a particular request from the client. For indications, the transaction ID is usually used in debugging [97].

STUN client sends a request to a server that is generally located in the public Internet or in an ISP's network when offered as a service known as a STUN server. The request arrives at the STUN server passing through one or more NATs between

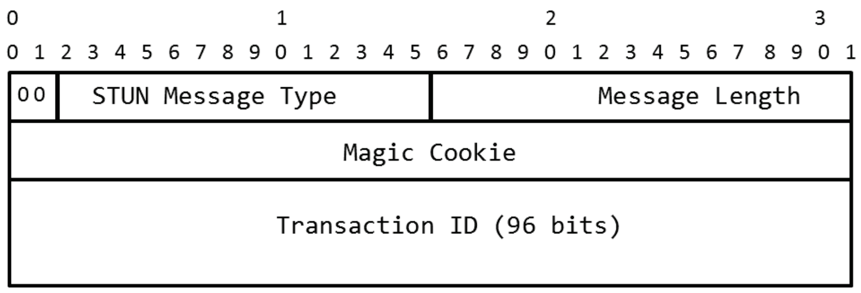

Figure 2.3: Format of STUN Message Header [97]

the STUN client and the STUN server. The NAT modifies the source transport address (that is, the source IP address and the source port) of the packet when the request is passed through it. Thus, the server sees the public IP address and port of the NAT closest to the server as the source transport address of the request. This is called a reflexive transport address. The STUN server copies this reflexive transport address into an XOR-MAPPED ADDRESS attribute in the STUN response and sends the response back to the STUN client. As this packet passes back through the NAT, the NAT will modify the destination transport address in the IP header, but the transport address in the XOR-MAPPED-ADDRESS attribute within the body of the STUN response will remain unaffected. In this way, the client can learn its reflexive transport address allocated by the outermost NAT with respect to the STUN server [97]. It uses this address to tell the other computers on the Internet where it can be contacted. To guarantee that the NAT binding does not expire the computer sends a 'keep alive' request to the STUN server periodically [92].

The **Traversal Using Relays around NAT (TURN)** is a NAT traversal protocol standardized by IETF. It is an extension of STUN. TURN is needed since under some circumstances it can be impossible for a host located behind a NAT to communicate directly with other hosts (peers). In such circumstances, an intermediate node can act as a communication relay between the hosts. TURN can be used to control a communication relay that acts as an intermediate node to the host.

The host can control the process of relaying through the TURN server by obtaining an IP address and port on the server, called the relayed transport address. Thus, when a peer sends a packet to the relayed transport address, the server relays the packet to the host and when the host sends a data packet to the server, the server relays it to the appropriate peer using the relayed transport address as the source. [84].

Always using a TURN relay would be enough to traverse nearly all firewalls. However, using a TURN server is expensive and also brings extra overhead by adding a new entry to the data path. In cases where there is no other way to establish a connection, a TURN relay must be used but in other cases depending upon the cost and traffic it is preferable to try to avoid the use of a relay.

ICE (Interactive Connectivity Establishment) is a NAT traversal solution standardized by the IETF. ICE is designed for UDP-based media streams but can be extended for other transport protocols like TCP [96]. ICE combines various NAT traversal protocols creating many possible paths to connect two peers and chooses the best path [9]. ICE is based on STUN and TURN. The IP address and port used to communicate with another peer is called an ICE candidate. There are three kinds of ICE candidates [28]:

- Host candidates that refer to the local IP address.
- Reflexive or STUN candidates that refer to the IP address of the peer's NAT (assuming they are only behind a single NAT). These candidates are determined by an external STUN server when requested by the peer.
- Relay or TURN candidates that refer to the IP address of a relay server allocated for use by the peer.

These ICE candidates can be exchanged via XHR/WebSocket/WebSync between peers. Once the exchange of ICE candidates are completed, both the peers start the ICE connectivity checks to determine the best possible candidate pair that can be use to communicate. After both the peers agrees upon a single candidate pair, the connectivity check stops and media can be sent/received using that candidate pair.

2.4 Signaling Protocols

Signaling is the mechanism of transferring control information for the purpose of establishing, maintaining and terminating a communication session between users in a network. The protocols that facilitate signaling are known as signaling protocols. Examples of signaling protocols include SIP, XMPP/Jingle, SS7, H.323 etc. In this thesis, we will be discussing SIP (Session Initiation Protocol) and XMPP (extensible Messaging and Presence Protocol), as they are the predominant signaling protocols used by major vendors for real time multimedia communications.

SIP (Session Initiation Protocol) [98] is an application-layer signaling protocol that supports voice, video and multimedia sessions. It is standardized by the Internet Engineering Task Force (IETF). The current version of the standard is SIP/2.0. The main SIP specification is RFC 3261[98].

Method	**Purpose**
ACK	Acknowledges the reception of a final response to an INVITE request
BYE	Terminates a session
CANCEL	Cancels the previous request sent by a client
INFO	Carries session-related control information
INVITE	Establishes a session
MESSAGE	Allows the transfer of instant messages
NOTIFY	Notifies a SIP node about an event that has occurred
OPTIONS	Queries a server about its capabilities
PUBLISH	Publishes event state
REGISTER	Informs a proxy server about a binding between an AoR (Address of Record) and a contact address
SUBSCRIBE	Requests asynchronous notifications of an event
UPDATE	Updates the parameters of a session

Table 2.1: Examples of SIP methods

SIP uses a client-server model and defines two message types i.e. request and response, for establishing, modifying and terminating sessions. Each transaction consists of a request and at least one response. SIP defines two types of responses: final and provisional. Examples of SIP requests are listed in Table 2.1. INVITE messages carry session description in their body. INVITE messages have all the session description information required to negotiate between participants for agreeing on a common set of compatible media types. Session Description Protocol (SDP) offers a standard representation for such session description information, irrespective of how that information is transported. Usually, SDP must convey enough information to enable applications to join a session. A SDP session description includes: session name and purpose, time(s) the session is active, the media comprising the session and information needed to receive those media (addresses, ports, formats, etc.).

Table 2.2 shows an example of session description for a conference (audio and video) that uses RTP to transport media from IP address 192.0.0.2 using ports 49180 and 52052, respectively.

Line	**Description**
v=0	The version of the protocol is "0" [71].
o=jdoe 3289054256 88 IN IP4 192.0.0.2	jdoe is the originator. The session ID is 3289054256 and the version is 88. The session was created on machine 192.0.0.2.
s=SDP Implementation	A short textual session name
i=A Seminar on the session description protocol	Further information about the session.
u= `http://www.conferenceurl.com/sdpseminar.html`	A link to extra information about the session.
e=jdoe@url444.jdoe.com (John Doe)	The email address of jdoe, including full name.
c=IN IP4 124.191.8.1	The IP address at which the sender of the SDP wishes to receive media.
t=3487140000 3487143600	The starting and the ending times of the session.
a=recvonly	The sender only wants to receive media.
m=audio 49170 RTP/AVP 0	The sender of the SDP wishes to receive audio on port 49170 using RTP Profile for Audio and Video Conferences with minimal Control running over UDP. The final zero is extra parameter information for RTP/AVP.
m=video 51372 RTP/AVP 31	The sender of the SDP wishes to receive video on port 51372 using RTP Profile for Audio and Video Conferences with minimal Control running over UDP. The final 31 is extra parameter information for RTP/AVP.

Table 2.2: Example of SDP session description

Every SIP user (entity) has an address called a SIP URI (Uniform Resource identifier). The format of SIP URIs is similar to email addresses (user@domain).

SIP networks have end nodes known as user agents. They act either as a client or a server. The User Agent Client (UAC) sends requests and receives responses. The User Agent Server (UAS) receives requests and sends responses. A back-to-back user agent (B2BUA) is an intermediary that simultaneously acts as a UAS and UAC. It links a UAS to a UAC but appears as an end node to both communicating parties.[1]

The SIP architecture also includes a number of servers to provide routing, registration and authentication/authorization services. These are summarized as follows:

- **Registrar Server:** This acts as a directory which keeps track of all the UA. It registers the network location of a user agent who has logged onto the network.

It obtains the IP address of the user and associates it with their SIP URI on the system.

- **Proxy Server:** This is a network relay host that proxies requests and responses during session setup and termination. It acts as both a server and a client and can modify a SIP request before passing it further.
- **Redirect Server:** The Redirect server is used to redirect clients to the user agent they are attempting to contact. If a user agent makes a request, the redirect server can respond with the IP address of the user agent being contacted.
- **Presence Server:** It stores and provides presence information of clients.

Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE) was added as an extension to SIP to support instant messaging and presence services.

XMPP (eXtensible Messaging and Presence Protocol)[102], also known as Jabber is an XML (eXtensible Markup Language) based protocol developed in 1998 by Jeremie Miller and is currently standardized by the IETF working group and is specified in RFC 3920 and 3921 [77]. It is a set of open source technologies providing various services such as instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication and generalized routing of XML data [25]. It has a decentralized architecture enabling anyone to run an XMPP server for communication purposes and provides built-in security by Simple Authentication and Security Layer (SASL) and Transport Layer Security (TLS) [25]. The VoIP extension to XMPP is known as Jingle and was developed by Google [77]. While the data is sent over XMPP, Jingle aids XMPP clients in signaling of multimedia sessions by negotiating a connection between the participants to provide media transport over Real-time Transport Protocol (RTP) [24].

A comparison between XMPP/Jingle and SIP/SIMPLE is made below to list the important differences:

- SIP and XMPP have very different purposes. SIP's core functionality include negotiation between endpoints for creating, managing and terminating a session between the endpoints whereas XMPP provides a streaming pipe for structured data exchange between them [17].
- SIP is based on HTTP whereas XMPP is XML based.
- In the case of SIP, if the IP addresses of the participants are known, a server is not required but XMPP requires a server to control the two participants [61].
- SIP uses UDP, TCP or TLS as transport protocols whereas XMPP only works with TCP and TLS [61].

Depending upon the requirements a system can be build using any or both of these protocols.

2.5 IP Multimedia Subsystem (IMS)

Mobile telephony's rapid growth has created a huge demand to include numerous services in mobile devices. Due to this the starting point for the work on IP Multimedia Subsystem (IMS) was that the services that were formerly available only on Internet are now needed to transform to the cellular platform.

Developed by the 3rd Generation Partnership Project (3GPP), the IP Multimedia Subsystem (IMS) is based on the SIP signaling protocol and IP. IMS is an IP-based network infrastructure to deliver speech, data and other multimedia services regardless of the access medium.

IMS provides three primary goals or solutions to three important issues that operators faces today, i.e. Quality of Service (QoS), charging and billing, and integration of services [54].

Modern networks are mostly based on packet switched protocols and try to provide the best possible service without any guarantee of QoS. In some cases, this results in a bad user experience. IMS resolves this issue by establishing sessions in a way that takes QoS into account in order to enable a good user experience [54].

IMS provides the operator with information about the type of service used by the user so that the operator can decide on an alternative charging scheme based on the type of services the customer has used. An example of this is providing time based charging schemes for multimedia sessions [54].

Finally, the most important reason that IMS exists is to provide an integrated service. Due to vendor specific equipment, operators cannot use or combine any third party tools or services. This gives the operators limited options that they can provide to their customers. IMS standardizes the interfaces providing the operators with multi-vendor services [54].

2.5.1 Architecture of IMS

IMS has a three layered architecture (figure 2.3) that comprises of application layer, session control layer and connectivity layer. The IMS architecture is designed to be very flexible and operators are free to combine or remove components that are not required by them [54]. This model gives IMS the ability to offer simplified operations, service consistency, ease of service bundling, streamlined provisioning, and common billing [82].

The application layer is the uppermost layer in the IMS architecture. It consists of application and content servers that enable various services to the users. A typical SIP application server would implement generic and reusable service enablers like presence and group list management as services [26].

The session control layer is the middle layer and the most important layer as it carries out the core activities of IMS and interacts with the control layer and connectivity layer [26].The CSCF (Call Session Control Function) is the primary node of this

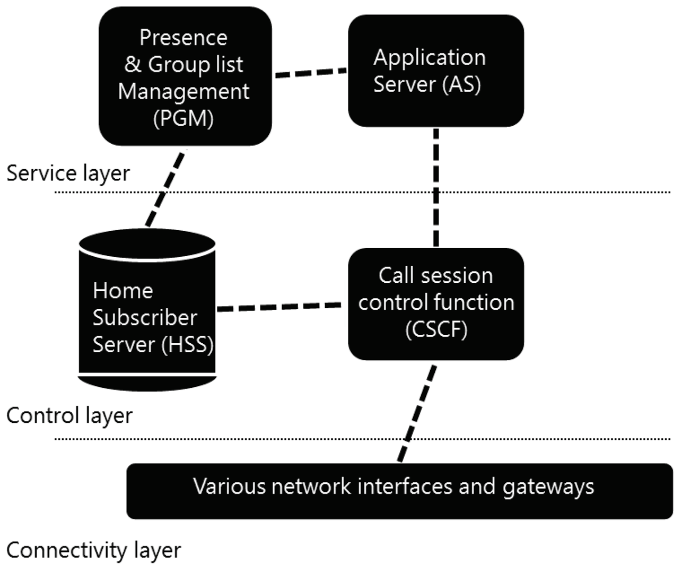
Presence & Group list Management (PGM)
Application Server (AS)
Service layer
Home Subscriber Server (HSS)
Call session control function (CSCF)
Control layer
Various network interfaces and gateways
Connectivity layer

Figure 2.4: Architecture of IMS

layer.

The connectivity layer or the Interworking and Media Layer is the lowest layer that provides access to the IMS by communicating with various vendor specific and external networks. It consists of media gateways and media servers for media processing [26] [82].

2.5.2 Core components of IMS

IMS includes a wide range of components. Briefly some of the core components are as follows:

- **Service enablers:** Service enablers are present in the application layer. They represent the basic and reusable base for creation and consumption of multimedia services. Service enablers comprise of two components: Presence and Group list management [26].

  1. **Presence:** Presence service enabler provides the users with information about the availability of and the types of communication services supported by other users within the same group. The users can set their preferences on what information can be viewed by whom [26].
  2. **Group list management** The group list management service enabler allows the users to create and manage groups and members associated with those groups. The user can grant or revoke permission to a group according to their preferences. Some application examples include personal buddy lists, black lists, public/private groups' access control lists etc. [26] [8].

- **CSCF (Call session Control Function):** CSCF is the core component present in the session control layer. It is responsible for managing and scaling session control in the IMS, making the network efficient. Based on different functionalities, CSCF is divided into three categories that are [54]:

  1. **S-CSCF (Serving-CSCF):** S-CSCF is the core CSCF. It is a SIP server and provides session control, registration and authentication [101].
  2. **P-CSCF (Proxy-CSCF):** P-CSCF acts as the access point to the IMS, both from a home network or a visited network. It forwards the SIP messages to the home S-CSCF [82].
  3. **I-CSCF (Interrogating-CSCF):** I-CSCF acts as the gateway to each individual IMS domain. It determines access to the S-CSCF and locates the user. It also prevents unauthorized users from gaining access to the S-CSCF and HSS [101].

- **Home Subscriber Server (HSS):** HSS is also present in the session control layer. It is a database that stores all the subscribers information which includes user identities, services, location etc. S-CSCF uses HSS to obtain user profiles [101].

- **Application Server (AS):** The AS is an application layer server that hosts and executes services and interacts directly with CSCF's and HSS. It is based on SIP and also uses the Diameter protocol [69] [101].

2.6 Summary

In this chapter we have provided the background to familiarize the reader with the scope of the thesis. We started with collaboration tools and focused mainly on the peer-to-peer architecture. We also covered concepts like ICE for NAT traversal with STUN and its extension TURN and introduced the idea of how ICE can traverse firewalls. We also described SIP and XMPP signalling protocols and introduced how these protocols facilitate a peer-to-peer session. Finally, the basics of IP multimedia subsystem (IMS) were discussed. The important thing to know about IMS is to understand the various components that are linked to form the IMS infrastructure.

In the next chapter we will discuss some more concepts related to the Web. This would provide a better foundation to understand how interactions happen in the application layer.

Chapter 3

Web based multimedia technologies

This chapter gives a general overview of the latest version of the core language of the World Wide Web, that is, the *Hypertext Markup Language Version 5* commonly known by its short form HTML5. Currently the status is still under development but portion of it is already supported in the latest versions of all major browsers [3].

In addition in this chapter we will also discuss WebRTC (and its APIs), which is a facilitator of real-time communication in the Web.

3.1 HTML5

Web pages are written in Hypertext Markup Language (HTML). HTML was created by Tim Berners-Lee in 1989. Originally, web browser served just as a reader to Web pages. But today web browsers have developed into a portal to different online media. Thus, HTML5 is not merely a mark-up language. It can primarily be seen as a family of technologies that includes JavaScript API's and CSS. Therefore, this chapter will also focus on all these major technologies, which together make HTML5 a powerful web technology tool [60] .

3.1.1 Evolution

HTML has gone through a series of changes in its evolution. Presently it is maintained by the 'World Wide Web Consortium' (W3C) that develops open standards for the World Wide Web after the IETF closed its 'HTML Working Group' in 1996.

In 1997 to make HTML4, the current HTML standard, more extensible and increase interoperability with other data formats, XHTML, the XML based equivalent was proposed for development, thereby halting the development of HTML4 further by W3C. The development of XHTML was completed in 2000 with release of XHTML version 1.0.

By 2004, W3C supported developing XML-based technologies by rejecting the renewal efforts of HTML4 by Mozilla and Opera. This resulted in the formation of a new community called "Web Hypertext Application Technology Working Group (WHATWG)" jointly by Apple, Mozilla and Opera for further development of HTML and related technologies (that we know today as HTML5) based on core principles of backward compatibility, detailed specification and interoperability. W3C showed interest in joining the development of HTML5 and in 2007 it started working with WHATWG [4].

In December 2012, W3C published the complete definition of HTML5 and Canvas 2D (the two dimensional context for the HTML 'canvas' element which provides objects, methods and properties to draw and manipulate graphics on a canvas drawing surface [95]) specification and also stated that the final standardisation work would be completed by 2014 after checking interoperability with different browser implementations and testing [5].

3.1.2 HTML5: Behind the scenes

HTML5's name is descriptive enough, stating that it is the 5th revision of the HTML standard. It is the successor to HTML4.01, DOM2 HTML and eXtensible HTML 1.1 (Second version) [93] [68].

HTML5 is the combination of various technologies and features that is going to revolutionize Web applications and ease the process of their development. Many features were already being used by developers without any standardization earlier. Also it is an approach to collect and standardise technologies that were already incorporated in web browser years ago [93].

As mentioned earlier, while designing HTML5, backward compatibility was a core principle to resolve compatibility issues with the existing features by providing fall-back content for old browsers. HTML5 is also an independent platform that works with any operating system supporting modern web browsers.

HTML5 is made up of three important technologies: the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript API's. The markup language provides the structure to the web page, whereas style sheet language CSS presents and styles the page and the JavaScript scripting language adds behaviour and interaction to the page.

The different types of HTML such as HTML 4.01 Strict, HTML 4.01 Transitional, XHTML 1.0 Strict, and many more are defined in their respective W3C specifications, but they also need to be defined in a machine-readable language specifying the legal structure, elements and attributes of a type of HTML. This definition is called a "Document Type Definition" or DTD for short, and its syntax is "Doctype".

HTML documents (Listing 3.1) are made up of elements called tags, which define the presentation of a web page. The <link> tag represents the relationship between a document and an external resource (mostly external CSS file). The <script> tag is used to define a client-side script, such as a piece of JavaScript.

Salient features of HTML5 as a markup language are as follows:

- HTML5 is a superset of HTML 4.01, which means that older pages will still work in the latest browsers.
- CSS and JavaScript are now the defacto standards that come with the HTML5 package. Therefore the "link" and "script" tags no longer need to be specified with a type attribute.
- HTML5 has a simple "doctype" that is supported by older browsers and all future versions will have the same underlying style as mentioned by W3C.
- UTF-8 is made the standard character set by simplifying the <meta> tag to include only the character encoding [68].

```
<!doctype html>
<html>
 <head>
  <title> My Sample Title </title>          // title
  <meta charset="utf-8" >                   // meta tag
  <link rel="stylesheet" href="lounge.css"> // link tag
  <script src="hello.js" ></script>          // script tag
 </head>
 <body>
  <h1> Welcome </h1>
  <p>
    <img src="image.png" alt="Drinks" >
  </p>
  <p>
     Hello World
     <a href="hello.html" > Hello </a>
  </p>
 </body>
</html>
```

Listing 3.1: The general structure of HTML5 document

3.1.3 HTML5 and Javascript

A script refers to programmable code that runs without pre-processing. JavaScript is the scripting language used in Web applications. It gets executed within a browser during page download. It is the most popular European Computer Manufacturers Association (ECMA) script developed by Netscape and was standardized by ECMA International [11][12]. It was originally implemented in the client side to interact with the user's actions, thereby allowing dynamic behavior. It was not until recently that developers realized the capabilities of JavaScript as a more general tool. Due to its rapid development and the sudden increase in interest, JavaScript has grown from a mere client side scripting language to a full-fledged programming language that can also be used for server side implementation. This has been made possible by additions like Node.js (a server-side solution for JavaScript), jQuery (a JavaScript library), JSON (JavaScript Object Notation) and MongoDB (a database based on JavaScript) [10].

A HTML document (Listing 3.2) is structured hierarchically like a tree, where each branch in the tree may be referred to as an element in the structure. For each element, the browser creates an object that represents it and places it in different hierarchical positions. This structure (Figure 3.1) is often called Document Object Model (DOM). Together with JavaScript, DOM allows the implementation of interactive web pages.

The process of an action occurring in the browser when a web page is loaded with or without the user's interpretation is usually known as an Event. The object that performs the task when the event occurs is called the event handler. Usually the event handling is performed by JavaScript which is assigned to the object or methods that execute as soon as an event is called [49].

When a page is loaded in the web browser, the HTML is parsed by the browser and an internal model of the DOM is created. JavaScript interacts with the DOM to access and add behaviours to the elements in the document in order to create a dynamic impression for the viewer in the browser.

The execution is performed top down. It is single threaded since JavaScript itself is also single threaded. This is why, when loading external JavaScript files, the parsing of the main HTML page is suspended. However, the CSS files can be download simultaneously.

```
<html>
   <head>
     <script src="sample.js"></script>
     <link rel="stylesheets" href="style.css"></link>
   </head>
   <body>
     <h1>Hello World</h1>
     <p>An example of HTML5<p>
   </body>
</html>
```

Listing 3.2: An simple example of a HTML document with CSS and JavaScript elements

3.1.4 HTML5 and Communication API

Traditionally, the Web used the request/response mode of communication as HTTP was considered to be the primary protocol for client/server interaction. HTTP, using the request/response mode, would simply load a static page from the server each time the client made a request.

In early 2005, Asynchronous JavaScript and XML (AJAX) was introduced to make

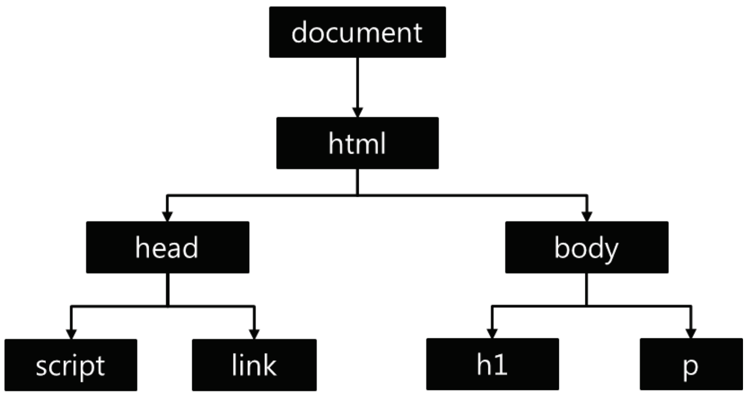

Figure 3.1: Representation of DOM hierarchial structure based on Listing 2.2

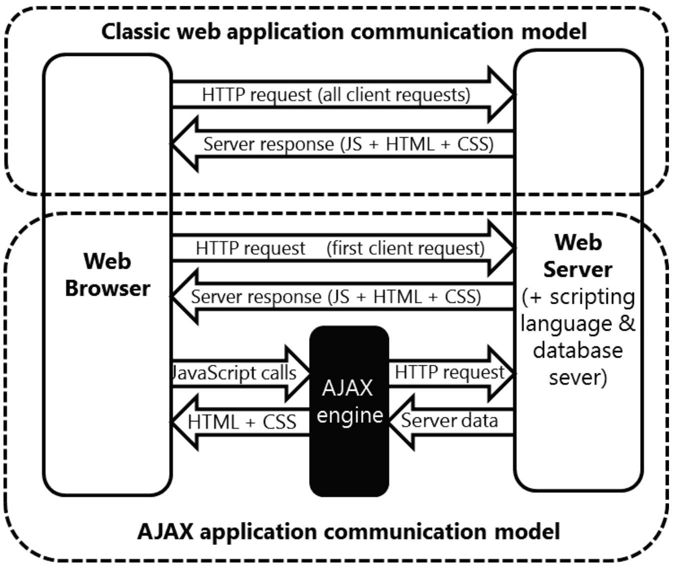
Classic web application communication model
HTTP request (all client requests)
Server response (JS + HTML + CSS)
Web Browser
HTTP request (first client request)
Server response (JS + HTML + CSS)
Web Server (+ scripting language & database sever)
JavaScript calls
HTML + CSS
AJAX engine
HTTP request
Server data
AJAX application communication model

Figure 3.2: AJAXmodel

the Web look dynamic. An AJAX based web application adds a new layer to the communication model as shown in Figure 3.2. The entire page is loaded only at the first request which downloads the AJAX engine apart from HTML, JavaScript and CSS code. This AJAX engine will then send data to and retrieve data from the server asynchronously in the background. Information is then displayed by the engine without reloading the entire page. Since the entire page is not reloaded, the interface looks more responsive. This is because only the necessary information is passed between the client and server.

There were many other technologies around at the time when AJAX was developed such as Long Polling, Flash, XHR multipart requests and HTML files [56]. All these carry the HTTP overhead that makes them unsuitable for low latency applications. For example, in a real-time scenario like multiplayer gaming, using such a communication mode would be quite ill-suited.

The latest development in evolution of client/server communication for the Web is WebSockets, a new message based communication API built on Transmission Control Protocol (TCP), is suited for overhead and latency issues associated with bi-directional client/server interaction. It was part of the HTML5 specification that was later separated to keep both HTML5 and Web Sockets standards more focused. It enables bidirectional communication and keeps an open connection between a web browser and the server [111]. Messages between the client and server can be sent at anytime and are handled asynchronously.

3.2 WebRTC

WebRTC is a technology that enables real time communication over the World Wide Web. It is free and open source. Basically, it enables a web browser to support a plug-in free real time communication.

WebRTC technology can be implemented by calling simple Javascript APIs within any browser application provided the browser is WebRTC enabled and access to real-time communication capabilities has been authorized by the user. WebRTC supports communication of audio, video and auxiliary data without any restriction by the browser. WebRTC allows for peer-to-peer communication between participants. It provides the freedom to use any signalling protocol to establish the peer-to-peer communication channel by leaving it to the developers choice [42].

3.2.1 Evolution/History

Google acquired On2 (a video codec company popular for its VPx series of codecs, latest being VP8) and Global IP Solutions (a company building a media framework for real-time audio and video communication) in early 2010 [108]. Google made these technologies open source and involved the standard bodies of the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) for standardizing them in May 2011 [62]. IETF is responsible for the architecture and protocol specification implemention of the WebRTC stack in the the browser, whereas W3C is responsible for the Javascript API specification that can be used to deploy WebRTC in various Web applications [42]. The first implementation of a WebRTC enabled browser was prototyped by Ericsson Labs [105]. It is currently supported by three browser vendors namely Google, Mozilla and Opera [19].

3.2.2 Architecture

As discussed earlier, the WebRTC technology comprises of two building blocks: the WebRTC stack in the browser and the WebRTC JavaScript API. In an inner architectural view (Figure 3.1), these look like the upper and lower layers. The upper layer is the browser JavaScript API that lets web application interact with the WebRTC technology and the lower layer is a set of protocols the enable real-time communication between browsers.

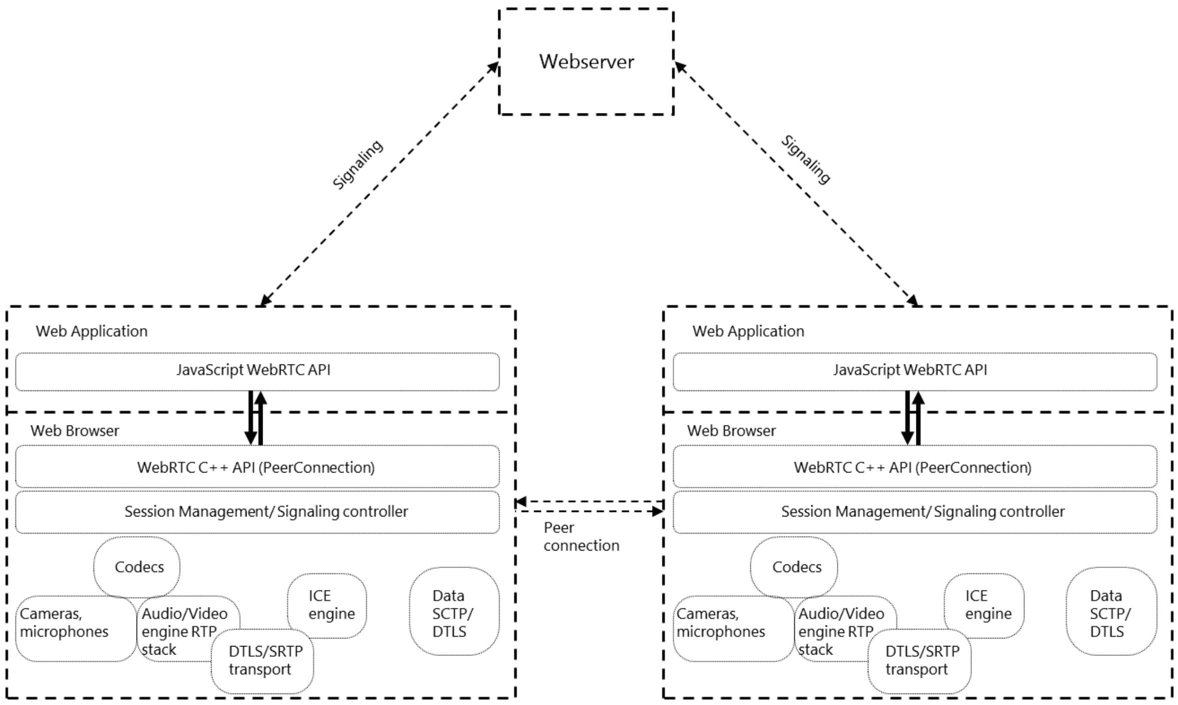

Figure 3.3: WebRTC architecture

The WebRTC browser stack is responsible for providing all the features of WebRTC like Network address translation (NAT) traversal, security, media and data transport. The WebRTC Javascript API acts as an intermediary link to control these features and to make use of them via a Web application.

The functionalities that WebRTC offers fall into three API: getUserMedia, PeerConnection and DataChannels. In short media streams (that is, the getUserMedia API) gives the browser access to the user's microphone and camera. Peer connection (that is, the PeerConnection API) is responsible for peer-to-peer connectivity over the Web. Data channels (that is, the DataChannel API) allows the browser to exchange auxiliary/application data in a peer-to-peer manner.

In an implementation architecture, a WebRTC session comprises of at least three parties. This includes the application provider (generally a Web Server) and two peers that run the application in their web browsers. Interoperability is a major issue in WebRTC due to differences in WebRTC implementations by browser vendors. In this case both peers are the using same browsers. Also a commnication session has already been established between the peers using certain signaling protocol like SIP (the WebRTC architecture does not restrict the use of any specific protocols for signaling). The peers negotiate their own available local parameters (i.e. media format, resolution and bandwidth limitations) by exchanging session descriptions [71] to create a connection between each other for media exchange.

3.2.3 Media Access and streaming

Real-time communication in a browser is possible by accessing the media streams (video, audio, or both) from the local device for instance from a local camera. Originally, the browsers relied on Flash, Silverlight or other plugins to capture audio and video from the device's hardware. With WebRTC along with HTML5, it is now possible to access the media streams from the local devices(camera, microphone) directly from the browser [65]. This is the core functionality of WebRTC.

"getUserMedia()" is the API (described in Listing 3.3) that allows Web applications to access media streams captured by the device's camera and microphone. It consists of zero or more synchronized media tracks, representing a local media resource such as a camera or microphone [103].

```
<video id="media" autoplay ></video>
<script>
var video = document.getElementById('media');
navigator.getUserMedia({video: true, audio: true }, success, error)
   ;  // JavaScript for accessing webcam and microphone input
   without a plug-in
   function success(stream) {
     video.src = window.URL.createObjectURL(stream);
   }
</script>
```

Listing 3.3: A simple example to embed Media Streams in a HTML document

3.2.4 Peer-to-peer connections

Another core functionality of WebRTC is the PeerConnection API. It provides peer-to-peer exchange of streaming data between two browsers. This is done after successfully establishing a session between the browsers with the help of signaling protocols via a server for instance using XMLHttpRequest, Web Sockets, SIP over WebSockets etc.

The PeerConnection acts as a communication channel between browsers. Media streams are added to the PeerConnection by the local peer for transmitting them to the remote peer. It is done by PeerConnection object which is created when the PeerConnection constructor is invoked in a JavaScript application.

3.2.5 Real-time transport in WebRTC

For transporting real-time media and real-time data, WebRTC uses two important protocols: Real-time Transport Protocol (RTP) and Stream Control Transmission Protocol (SCTP), respectively.

The real-time transport protocol (RTP) is an application layer, end-to-end, real-time media transfer protocol. In the context of WebRTC, it transfers real-time media, such as audio and video. It provides functionalities such as payload type identification, sequence numbering, timestamping and delivery monitoring. RTP usually runs on top of User Datagram Protocol (UDP) and can also be used with a framing layer on top of TCP. It comprises of two parts: the RTP data transfer protocol and the RTP control protocol (RTCP). The RTP Control Protocol (RTCP) is used for monitoring quality of service (QoS) and providing periodic feedback about the ongoing session [91]. Secure RTP (SRTP) is used to provide end-to-end confidentiality, message authentication and replay protection [103].

Auxiliary data can be transported directly between browsers using DataChannel along with SCTP [107][78].

3.2.6 DataChannel

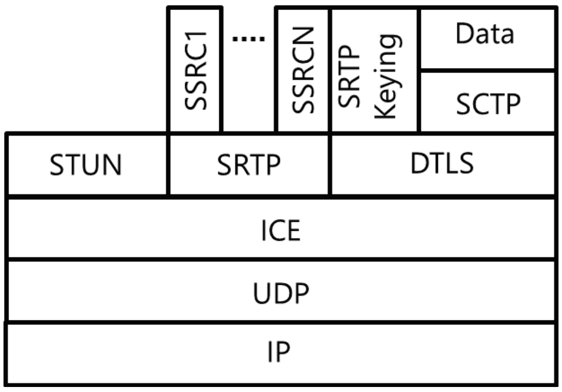

Figure 3.4: Protocol stack of WebRTC Datachannel

The WebRTC DataChannel is a bidirectional data channel between two peers. It is the third core functionality of WebRTC and is related to real time transfer of non-media data types (application data) with low latency, encryption and high throughput. Some real-world use cases include multi-player games, collaboration tools, decentralized networks, remote desktop applications etc.

There are some key features of DataChannel: to use a DataChannel, a peer session must be established using a PeerConnection API, multiple simultaneous DataChannels can be created by single PeerConnection API, DataChannels support both reliable and unreliable delivery methods and also incorporate built-in congestion control, the Datachannel API is similar to WebSockets and is independent of audio/video streams [110]. Additionally, it also supports binary types such as ArrayBuffers [115].

The transport of actual data is done by the underlying protocol stack that is made up of SCTP over Datagram Transport Layer Security (DTLS) over UDP (Figure 3.4). It also provides a NAT traversal solution.

The Stream Control Transmission Protocol (SCTP) is used as the generic transport protocol for these kinds of data types. SCTP is a packet-based and connection-oriented protocol providing transport functionalities to Internet applications. It is unicast, reliable and rate adaptive [89]. SCTP packets comprise of a common header along with one or more chunks. A chunk might contain either control information or user data [106]. Each SCTP connection supports multiple unidirectional streams, ordered and unordered delivery of user messages, reliable and partially-reliable transport of user messages.[48]. The SCTP over DTLS protocol [109] provides security functionalities such as confidentiality, source authentication, encryption etc.

3.3 Summary

In this chapter we provide a basic guide to understanding the various Web technologies related to WebRTC. We present the history behind and the correlation among these technologies, giving the reader a broader perspective on WebRTC.

Apart from WebRTC, another core-technology focus of this thesis is HTML5. We explained how a DOM (Document Object model) is formed after the HTML document is parsed by the browser and also described its involvement in enabling interaction among HTML, JavaScript and CSS.

Web based communication protocols, mainly AJAX and Websockets were discussed and compared. Websocket's advantage over traditional HTTP based communication in a real-time session is also highlighted.

Finally, we explained the WebRTC architecture and its three APIs. The architecture shows the basic model of any WebRTC based application. Thus, in our implementation described in the later Chapters of the thesis we will be following the same fundamental architecture. GetUserMedia, PeerConnection and DataChannel WebRTC APIs were introduced. In our implementation we will use these APIs to create a peer connection between browsers and exchange data over the peer-to-peer channel.

In the subsequent chapters, we will discuss our use case.

Chapter 4

Design and implementation

In this Chapter we present the methodology and design used to build a prototype peer-to-peer browser-based Web application with file transfer and chat features in the context of WebRTC. It is completely based on the WebRTC DataChannel API [50]. We have focused on the transfer of arbitrary data such as files, pictures, folders, documents etc. created by an application.

WebRTC provides Web applications with the ability to transfer real-time media and data files peer-to-peer without any browser plugins. WebRTC's DataChannel API enables the browser to send text or binary data over an active peer connection by opening/closing SCTP channels. At the time of writing this thesis, WebRTC standards were still evolving and this was making it more challenging to use these API's for implementation in a Web application. Different browsers have different requirements for the implementation, and the main issue is interoperability. Due to the fact that features are constantly changing and being updated, the implementation used in this thesis is highly experimental and based on the instructions and documentation provided between the periods of 1st March 2013 to 31st March 2013. The experimentation was carried out using the browsers Google Chrome Canary Version 26.0.1410.43 [30] and Mozilla Firefox Nightly Version 22.0a1 [35].

A case-study is also made later in the chapter. In this case-study we describe how IMS can be used as a signaling channel for establishing a WebRTC DataChannel between two collaborating endpoints.

4.1 Tools Used

Since our implementation is a Web based application and also due to the reason that WebRTC APIs are purely in JavaScript, that interacts directly with the browser, we use JavaScript Version 1.8.5 [14] as the primary language for implementing our prototype web application. HTML5 is used as the markup language and CSS3 as the style sheet language for presentation. Twitter-bootstrap Version 2.3.2 [18] and jQuery Version 1.9.0 [13] are the two frameworks we use for front-end graphics. No specific Integrated Development Environment (IDE) was used.

Site44 [38] was used as the HTTP server for hosting the Web application due to its easy integration with Dropbox [32]. Firebase [34] was used as the signaling server due to its scalability and free sandbox for testing purposes. As already mentioned, Google Chrome Canary Version 26.0.1410.43 [30] and Mozilla Firefox Nightly Version 22.0a1 [35] were used as the browsers to test the application.

4.2 Datachannel API

The DataChannel API enables access to SCTP channels via which arbitrary data is transferred. Here we describe the event handlers for the DataChannel JavaScript API that we used while implementing our prototype. This would help in understanding the later parts of this chapter. We used **channel.onopen** for opening the DataChannel. **channel.onmessage** is used for receiving messages. A basic structure is shown in listing 4.1.

```
var channel;
function openDataChannel() {
channel = peerConnection.createDataChannel('RTCDataChannel',
        {
            reliable: true/false
        }
    );
    channel.onmessage = function (event) {
        console.log(event.data);
    };
    channel.onopen = function (event) {
        channel.send('RTCDataChannel opened.');
    };

    channel.onclose = function (event) {
        console.log('RTCDataChannel closed.');
    };

    channel.onerror = function (event) {
        console.error(event);
    };
    peerConnection.ondatachannel = function () {
        console.log('peerConnection.ondatachannel event fired.');
    };
}
```

Listing 4.1: A basic structure of Datachannel JavaScript API

4.3 Architectural Design

Figure 4.1 shows the architecture of our prototype. It consists of two clients (A and B) each running in a WebRTC enabled Web-browser. Our web application (X) is running inside an HTTP server. A third-party server is used as a signaling server to establish, coordinate and terminate the connection between client A and client B. SDP negotiations between the clients through the signalling server are done over Websockets. A STUN/TURN server is used for NAT traversal. The Web application interacts with the browser via WebRTC JavaScript APIs and the SCTP channels are established between the browsers directly. The next section explains the architecture in greater detail.

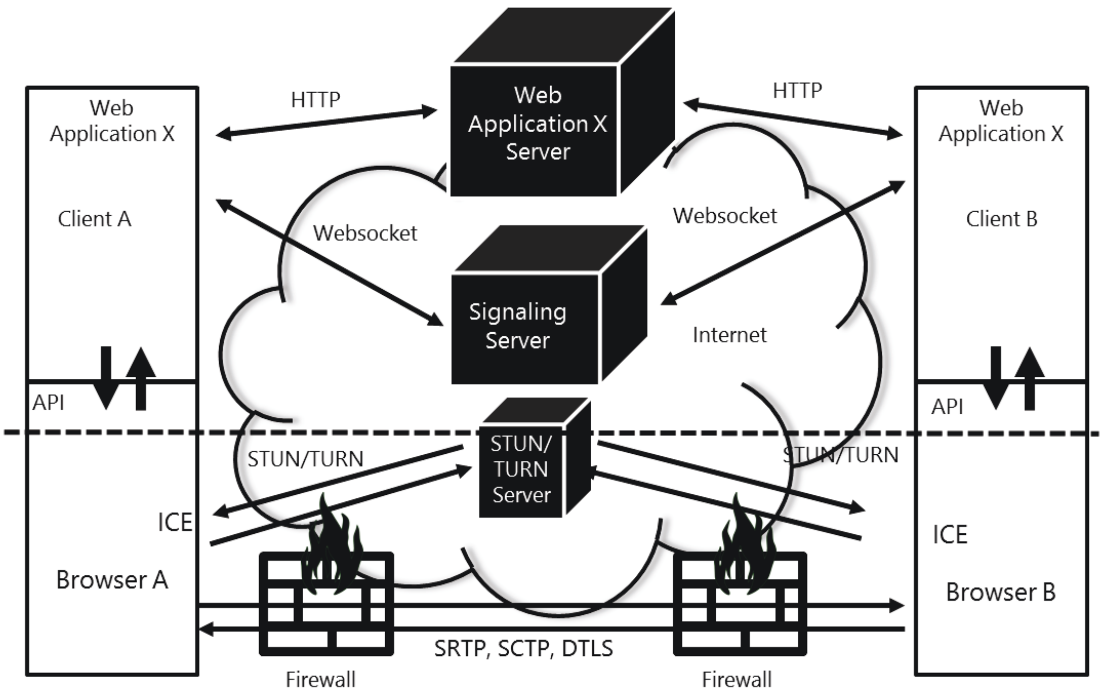

Figure 4.1: Architectural design

4.3.1 Implementation

The implementation of the prototype was solely client-side, as WebRTC APIs can be fully controlled by the clients. Third-party servers were used for signaling and for displaying presence.

We have implemented two prototypes for transferring arbitrary data: one for Chrome Canary and the other for Firefox Nightly. Because they have different specifications and requirements for implementing WebRTC, we have implemented two browser-specific prototypes. The differences are minor but significant (the differences will be discussed along the chapter).

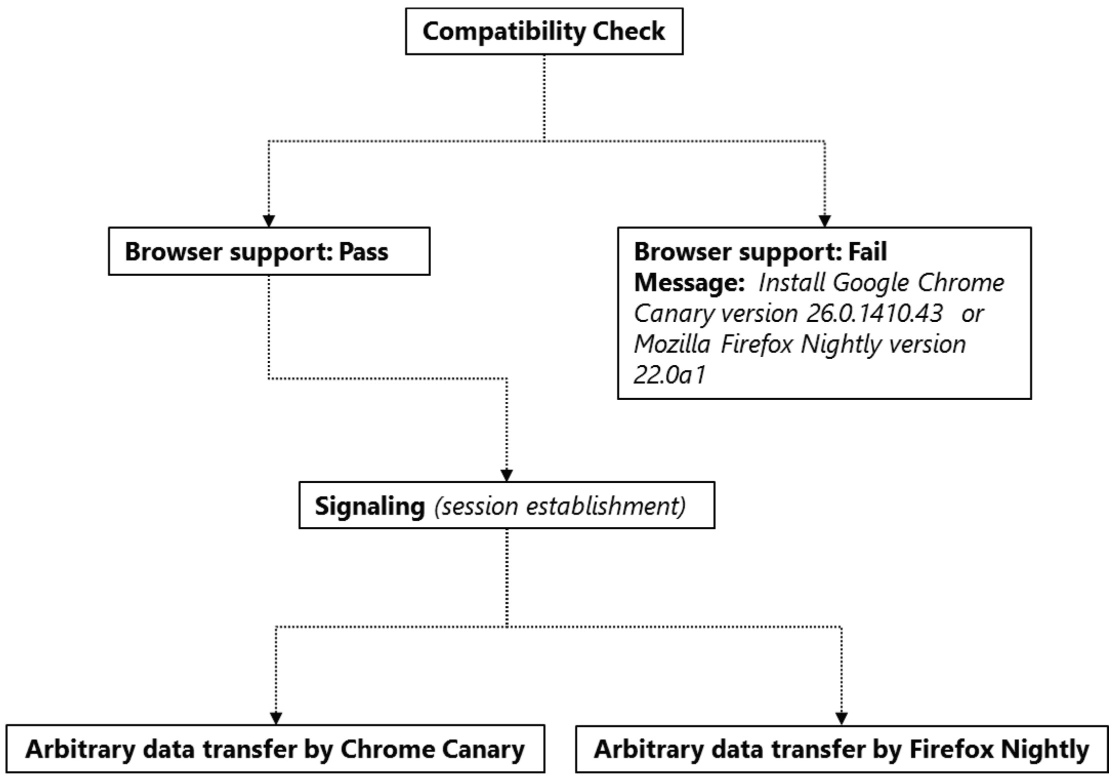
Compatibility Check
Browser support: Pass
Browser support: Fail
Message: Install Google Chrome Canary version 26.0.1410.43 or Mozilla Firefox Nightly version 22.0a1
Signaling (session establishment)
Arbitrary data transfer by Chrome Canary
Arbitrary data transfer by Firefox Nightly

Figure 4.2: Implementation flow model

Figure 4.2 shows how our implementation checks browser compatibility, and selects which data transfer method to use. Generally, the implementation is divided into three steps: the compatibility check, signaling implementation and the arbitrary data transfer implementation between the same vendor's browsers. It is important to note that only the data transfer is browser specific due to interoperability issues between two different browsers. In our prototype we have support for two clients. The offerer is the client that makes a request for a connection with the other client. The answerer is the client that responds to the request made by the offerer.

4.3.2 Compatibility Check

As discussed earlier at the moment of writing this thesis, WebRTC was still evolving and was not implemented on all available browsers. Some browsers supported only a few functionalities of WebRTC. We confined our prototype to a specific browser environment that needed to support all the necessary WebRTC APIs for the prototype to be successful. Therefore it was necessary to perform a compatibility check notifying users about the need to switch to the supported browsers (Google Chrome Canary version 26 or later and Mozilla Firefox nightly version 19 or later [62]) to test our prototype. Our prototype checks three parameters as pass of compatibility test, these being:

- Support for getUserMedia,
- Support for PeerConnection, and
- Support for DataChannel

4.3.3 Signaling Implementation

When a peer connects to the Internet it is normally behind NAT's and firewalls. To connect two peers it is necessary that they know where each of them can be reached. In our implementation peers represent the browsers. These browsers should also comply to the same standard of DataChannel, in order to be able to exchange data. After mutually agreeing on all these parameters and by knowing the location (IP address and port) of each other they can connect in a peer-to-peer fashion. These parameters are communicated by the peer using a signaling mechanism. This signaling mechanism establishes a session[1]. Signaling is responsible for establishing, coordinating and terminating the session between peers.

In general, two WebRTC applications need to use a signaling mechanism before being able to establish a peer-to-peer channel for data transfer. WebRTC does not specify any signaling mechanism or protocols, so the developer needs to select a mechanism by himself.

The operational flow of our signaling implementation is shown in Figure 4.3 and described in detail below:

- PeerConnection is the API used to set up peer-to-peer sessions. It establishes the P2P link. The offerer initializes the PeerConnection object as shown in step 2 of Figure 4.3. Listing 4.2 shows the code snippet for the initialization.

```
var PeerConnection = w.mozRTCPeerConnection || w.
    webkitRTCPeerConnection
var peerConnection = new PeerConnection(
    Ice servers, optional
);
```

Listing 4.2: PeerConnection initialization

- On a new incoming PeerConnection the browser sends messages to the STUN/-TURN server to obtain server reflexive and relayed candidates (see step 3 of Figure 4.3). Depending on whether it is a STUN or TURN server, the server either returns the server reflexive address of the browser (see step 4 of Figure 4.3) or both returns the server reflexive address (in a TURN Allocate response) and a relayed address on the TURN server that can be used by the browser to send and receive data. The browser fires the "onicecandidate" event for each STUN/TURN response (see step 5 of Figure 4.3). The browser itself

[1]Session is the conversation (exchange of information) that takes place between the two peers once the peers discovers each other.

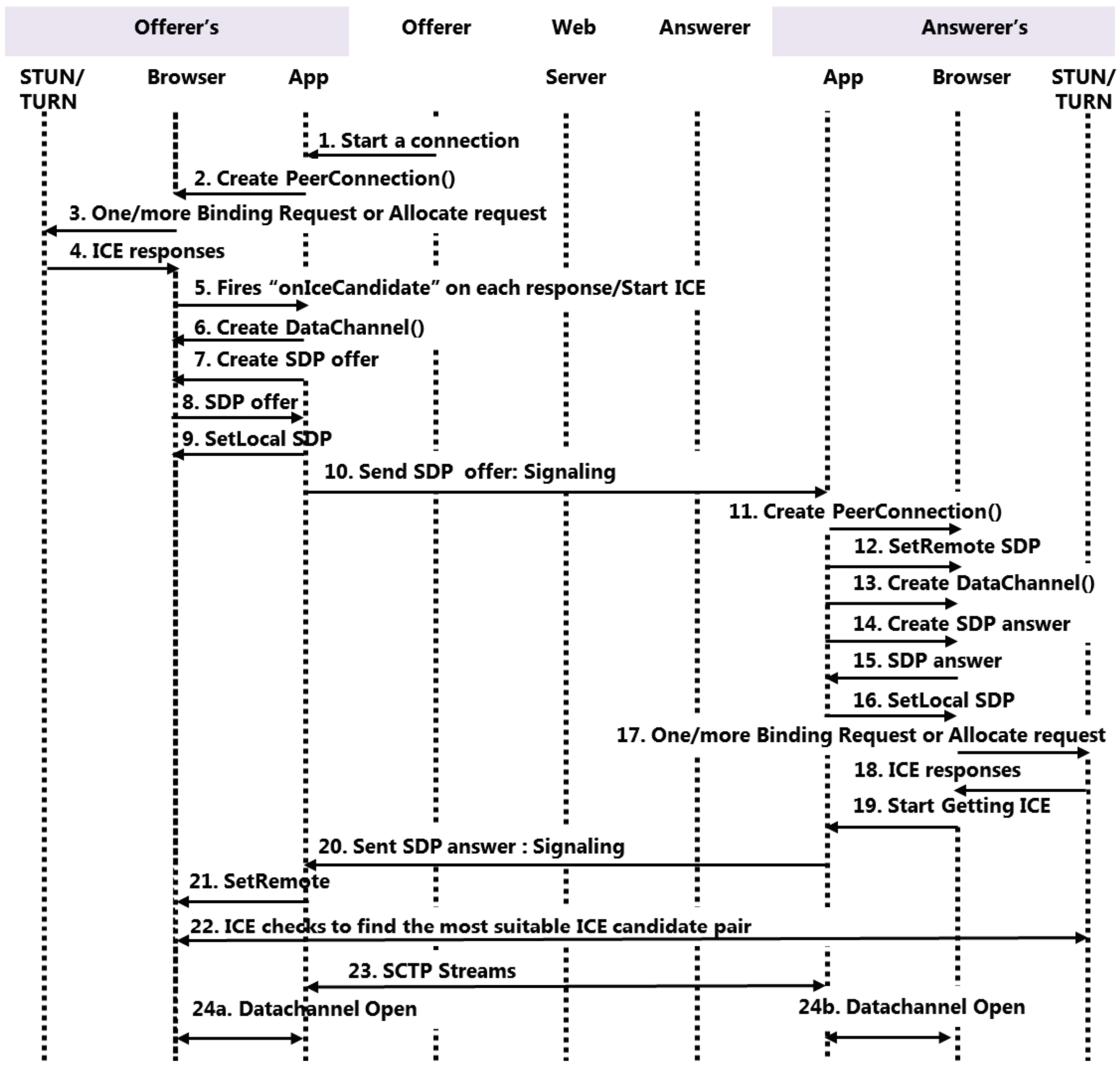

Figure 4.3: Flowchart of signaling

sets limitations for how many ICE candidates the STUN/TURN server should return.

- A PeerConnection object gathers the ICE candidates, creates SDP offer (see step 7 of Figure 4.3) and passes it over to the PeerConnection API (see step 8 of Figure 4.3) which sends it to the other client in the remote side (see step 10 of Figure 4.3). This is done over Websockets via the server.

- The answerer on the remote side gets the SDP offer (see step 10 of Figure 4.3), and gives it to the PeerConnection API as the remote description. The answerer then accepts the offer, generates its own SDP answer and passes it back to the PeerConnection API. The PeerConnection sends it back to the offerer. (see step 11-20 of Figure 4.3)

- The offerer gives the SDP answer to the peer connection as the remote SDP and ICE checks are done to find the most suitable candidates. (see step 21-22 of Figure 4.3)
- After a successful handshake, both the peers have all the information required to set up a peer-to-peer connection.
- Thus the SCTP ports get opened. Soon the "datachannel.onopen" event gets fired. In the "datachannel.onopen" event, the "onChannelOpened" method lets us know that the data ports are ready to transfer data/text files. On the other hand, the "onChannelMessage" callback is fired each time when SCTP ports receive a text message or data. (see step 23-24 of Figure 4.3)

4.3.4 Data Transfer Implementation

Since WebRTC DataChannel standardization was still ongoing at the time of writing this thesis, the DataChannel implementation of the Chrome and Firefox browsers were incompatible. The transfer of data was possible only among the same kind of browsers.

DataChannel Implementation : Before discussing browser specific file transfer implementations, it is important to note that to use the DataChannel in Firefox for exchanging arbitrary data we need to set up an audio/video stream even though we are not sending any media. This is a fake setup to open SCTP streams and has been identified as a bug. In contrast, in Chrome the DataChannel doesn't work if an audio/video session is already active [20].

During the implementation of our prototype, Chrome was using RTP instead of SCTP for data transfer [21].

Chrome Implementation

The operational flow of our data transfer implementation in Chrome Canary is shown in Figure 4.4. It is described in the following steps:

- Once the offerer uploads a file or types a message in the Chrome Canary browser the File API [2] reads the file as a data URL and then splits it into small chunks. The packets are split into sizes of 1000 characters. (see steps 1-4 of Figure 4.4)
- These split chunks are transferred via the SCTP channel in JSON strings over RTP data port in loops. Only one channel can send data at a time. Chrome Canary does not support multi-streaming yet. (see step 5 of Figure 4.4)
- DataChannel in Chrome supports unreliable transport only [19].

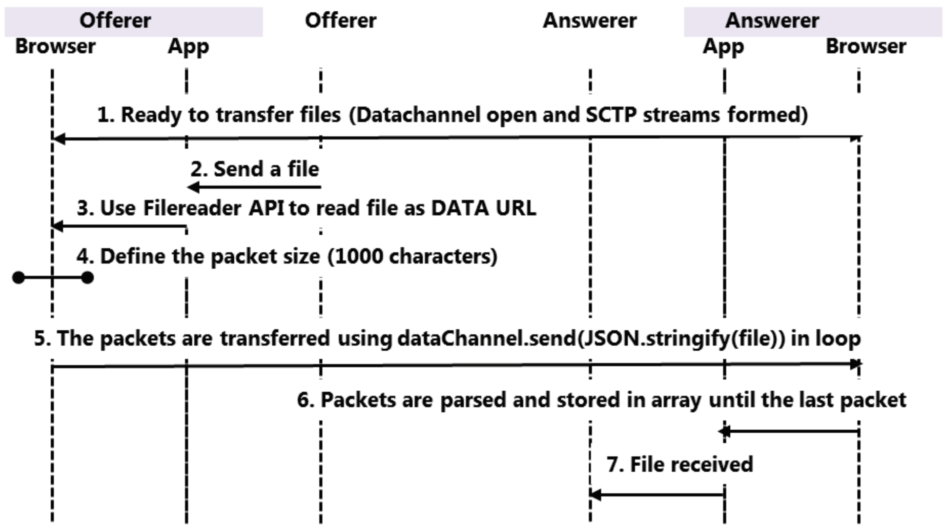
Offerer
Browser
App
Offerer
Answerer
Answerer
App
Browser
1. Ready to transfer files (Datachannel open and SCTP streams formed)
2. Send a file
3. Use Filereader API to read file as DATA URL
4. Define the packet size (1000 characters)
5. The packets are transferred using dataChannel.send(JSON.stringify(file)) in loop
6. Packets are parsed and stored in array until the last packet
7. File received

Figure 4.4: Flowchart of data transfer implementation in Chrome Canary

- When the chunks are received by the other peer, they are parsed and stored in an array until the last packet is received. Finally, the whole file is saved. (see steps 6-7 of Figure 4.4)
- Due to limitation in number of characters in a single packet and storage limitation large files cannot be handled and the Chrome browser is crashes.

Firefox Implementation

The operational flow of our data transfer implementation in Firefox Nightly is shown in Figure 4.5 and is described below:

- Once the offerer uploads a file in Firefox Nightly, the file reader API [2] is used to read the file into blobs. Firefox analyzes the message type and length to create packets accordingly. Then the larger blobs are broken down into smaller chunks. These chunks are transferred directly over 16 SCTP streams by calling "RTCDataChannel.send(file)" method in reliable mode. On Firefox, text and files can be sent simultaneously. (see steps 1-5 of Figure 4.5)
- The other peer waits until it receives all the chunks. Each chunk contains timestamps so that Firefox can understand how many chunks are remaining. As Firefox implementation is reliable, packet loss is handled by retransmission. When a Firfox receives the last chunk it calls the "dataChannel.onmessage" event and returns the whole file as a blob. Now the file reader API reads the blob as dataURL to save it to disk. (see steps 6-8 of Figure 4.5)

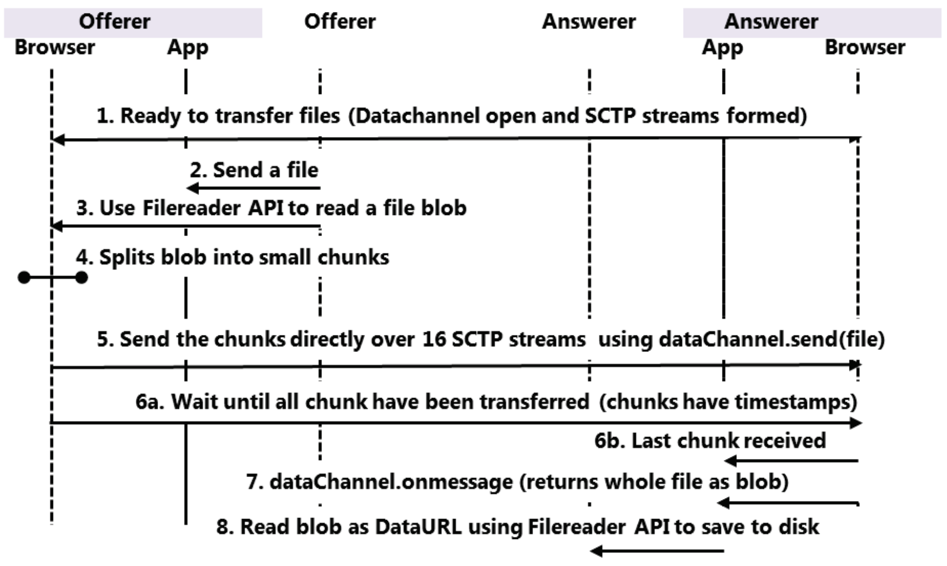
Offerer Browser
Offerer App
Offerer
Answerer
Answerer App
Answerer Browser
1. Ready to transfer files (Datachannel open and SCTP streams formed)
2. Send a file
3. Use Filereader API to read a file blob
4. Splits blob into small chunks
5. Send the chunks directly over 16 SCTP streams using dataChannel.send(file)
6a. Wait until all chunk have been transferred (chunks have timestamps)
6b. Last chunk received
7. dataChannel.onmessage (returns whole file as blob)
8. Read blob as DataURL using Filereader API to save to disk

Figure 4.5: Flowchart of data transfer implementation in Firefox Nightly

4.3.5 Analysis

After the implementation of the two prototypes i.e. in Chrome and in Firefox, we compared the session descriptions produced by both implementations. An extract of SDP generated in the consoles of Chrome and Firefox are shown in Appendix C and D, respectively. "a=sendrecv" can be seen in Appendix D that shows a fake audio/video call stream is created as mentioned above. Again in the Chrome SDP (Appendix C) we can see that the exchange of data is through RTP, compared to SCTP/DTLS in Firefox SDP (Appendix D). The default SCTP streams can be seen as '16' in Appendix D. From the SDP, we can see that Firefox is closer to the W3C WebRTC DataChannel specifications [50].

4.4 Case study

In this section, we will present three technologies (that is, WebRTC DataChannel, IMS and Microsoft Lync) that would be used to describe a case study.

The IP Multimedia Subsystem (IMS) is the next-generation core network for 3G mobile systems. IMS uses Session Initiation Protocol (SIP) as the session control protocol.

Many enterprises use Microsoft Lync services for collaboration and communication [37]. Lync (previously known as Office Communicator) is a unified enterprise communications platform [57] [74]. It provides access to all of Microsoft's Office tools along with audio, video, and Web conferencing, screen sharing, instant messaging and presence and enterprise voice. It is available on almost all platforms (desktop, web and mobile) and can easily be integrated with existing infrastructure with high reliability and powerful administration tools [58]. Developers can embed Lync features into their own applications. This makes Lync highly extensible and flexible as each feature can be implemented independently. As an example, the lyncpresence.orbitone.com uses the Lync Presence information directly on a website [40].

The Lync Server [100] (previously known as Office Communications Server) is a server that provides the infrastructure for real-time enterprise communications between various Lync oriented clients. It can be used to communicate within a single organization, between multiple organizations and with external individuals. It uses Session Initiation Protocol (SIP) for signaling along with the SIMPLE extensions to SIP for IM and presence [75].

The Microsoft Lync server also gives an option for interoperability and allows its information to be accessed by third-party software. However, for successful integration, the third party software needs to qualify for the Microsoft Unified Communications Open Interoperability Program (UCOPI) [59]. This is a qualification program that provides enterprise telephony services and infrastructure with access to the Microsoft Lync Server providing seamless setup, support and integration. Cisco Unified Communication Integration $^{(TM)}$ for Microsoft Lync is an example of software that qualifies for UCOPI and has the capability to directly connect to Microsoft Lync [31].

WebRTC has the capabilities to provide peer-to-peer data communication between users. It is a built-in feature in the browser and interacts with any web application using JavaScript APIs.

In this thesis, we propose a novel idea to connect the above mentioned technologies (that is, WebRTC DataChannel, IMS and Microsoft Lync) in order to show how arbitrary data can be transferred peer-to-peer between two clients in two different collaborative environments with different access technologies and different networks. This proposal consists of a case study, designing an architecture to show the interoperability of browser/based version of Microsoft Lync and a native WebRTC client (browser) to transfer peer to peer data with IMS used as a signaling channel to establish a WebRTC DataChannel between the two browsers.

Our case study attempts to answer the question of how arbitrary data can be transferred peer-to-peer between the two clients in two different collaborative environments with different access technologies and different networks. To demonstrate this we used IMS as the network infrastructure and browser/based version of Microsoft Lync and WebRTC-based browser as the respective clients in an enterprise environment.

In the following sections we describe the architecture and the case study.

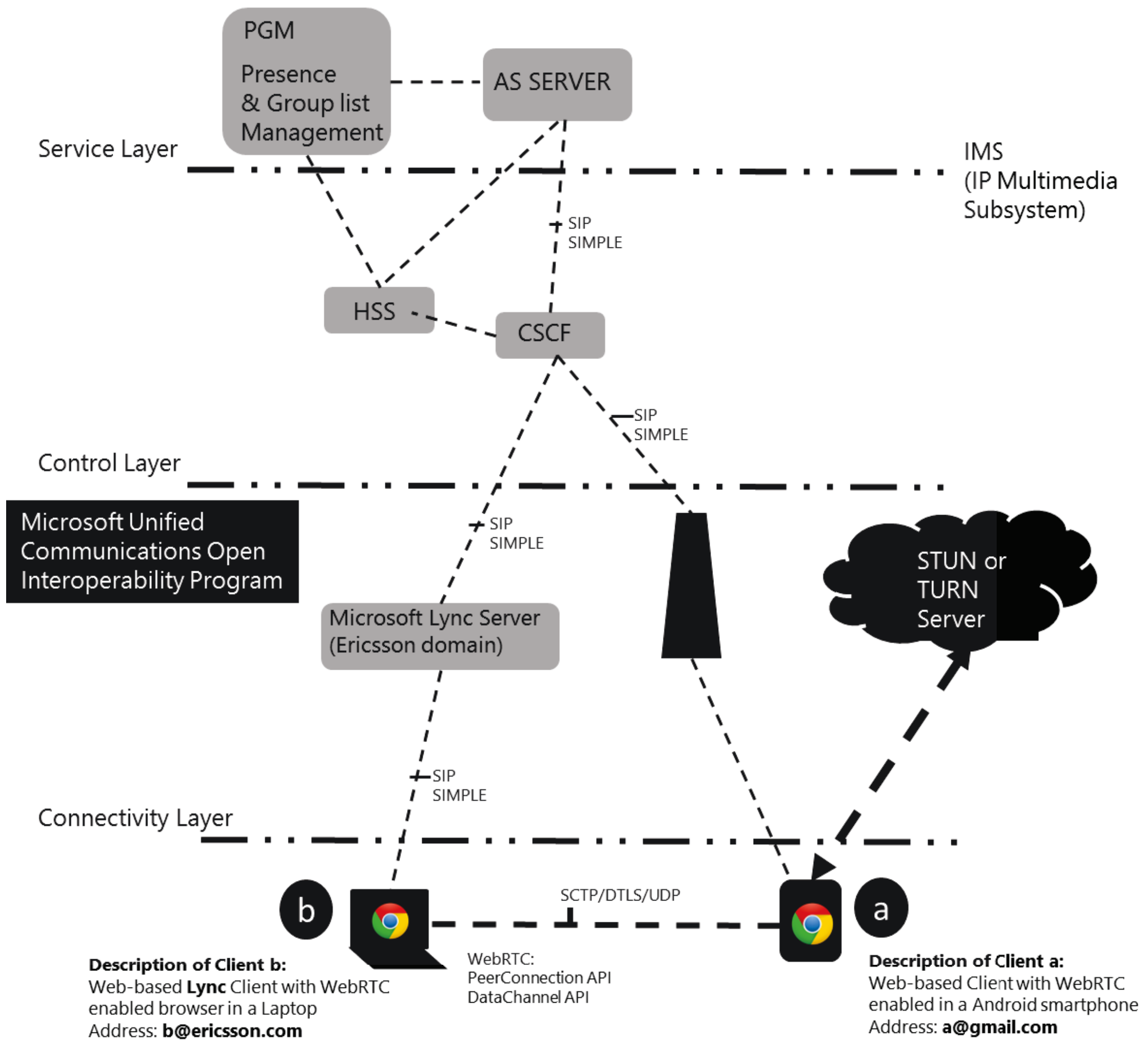
PGM
Presence & Group list Management
AS SERVER
Service Layer
IMS (IP Multimedia Subsystem)
SIP SIMPLE
HSS
CSCF
SIP SIMPLE
Control Layer
Microsoft Unified Communications Open Interoperability Program
SIP SIMPLE
STUN or TURN Server
Microsoft Lync Server (Ericsson domain)
SIP SIMPLE
Connectivity Layer
b
SCTP/DTLS/UDP
a
Description of Client b: Web-based Lync Client with WebRTC enabled browser in a Laptop Address: b@ericsson.com
WebRTC: PeerConnection API DataChannel API
Description of Client a: Web-based Client with WebRTC enabled in a Android smartphone Address: a@gmail.com

Figure 4.6: Architecture of our Case Study

4.4.1 Proposal

We have used Ericsson as our subject enterprise and browser-based version of Microsoft Lync as the collaboration tool in this case study. Figure 4.6 depicts the architectural design. There are four major components: the Lync Server (of Ericsson corporate domain), an IMS core network, Client 'a' (A smartphone with address a@gmail.com) and Client 'b' (a Lync client in a Laptop browser with address b@ericsson.com).

We consider in our case study that the devices are IMS compliant [2] and all the

[2]IMS-compliant refers to systems that are designed to support multiple AS within the same IMS infrastructure. This allows the IMS user to use many new services without deploying other infrastructure (authentication, authorization, charging, etc.) for the same device, making it simple and efficient. [27]

browsers support WebRTC (also considering that the smartphone browser will support WebRTC as predicted by [53]). We also assume the availability of a browser-based Lync application that supports WebRTC. The IMS Application server (AS) is qualified for Microsoft Unified Communications Open Interoperability Program (UCOPI) thereby getting access to the Lync server of Ericsson domain.

Our server is a part of the AS. Client 'a' registers itself to the IMS by exchanging credentials with the CSCF and HSS. In IMS, registration is an important process for security, location and services that a user is subscribed to.

Client 'a', once registered, gets access to the AS and its services. The Presence and Group Management (PGM) keeps track of the user's presence information with SIP SIMPLE protocol and manages its list.

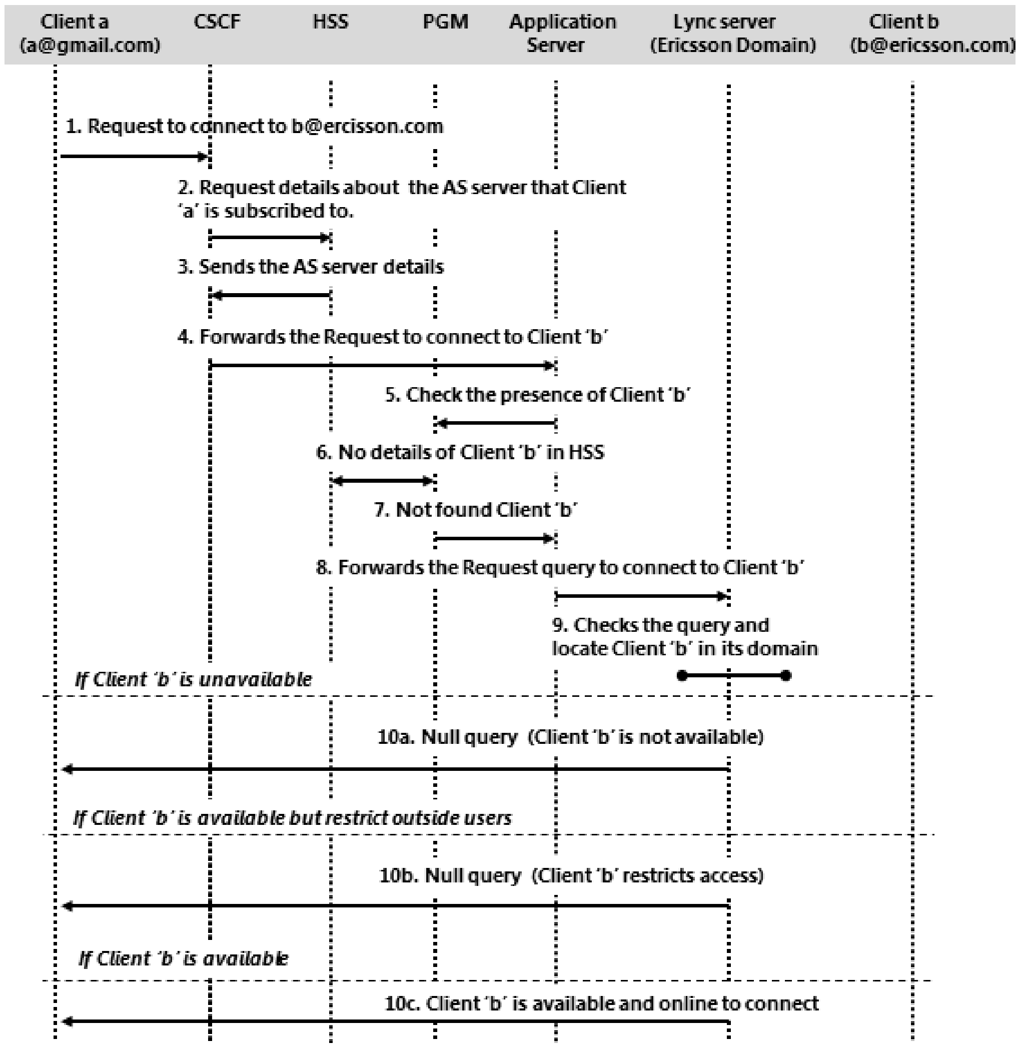
Client a (a@gmail.com)
CSCF
HSS
PGM
Application Server
Lync server (Ericsson Domain)
Client b (b@ericsson.com)
1. Request to connect to b@ercisson.com
2. Request details about the AS server that Client 'a' is subscribed to.
3. Sends the AS server details
4. Forwards the Request to connect to Client 'b'
5. Check the presence of Client 'b'
6. No details of Client 'b' in HSS
7. Not found Client 'b'
8. Forwards the Request query to connect to Client 'b'
9. Checks the query and locate Client 'b' in its domain
If Client 'b' is unavailable
10a. Null query (Client 'b' is not available)
If Client 'b' is available but restrict outside users
10b. Null query (Client 'b' restricts access)
If Client 'b' is available
10c. Client 'b' is available and online to connect

Figure 4.7: Flowchart of messages flow in the proposed case study

The operational flow of messages of our proposed solution is shown in Figure 4.7.

In Figure 4.6, we assume that client ’a’ wants to connect to client ’b’ for real-time communication. We now describe the operation of our proposed solution in the following steps :

- Client ’a’ sends a request to the AS with Client ’b’s’ address (b@ericsson.com). The request first reaches the CSCF that interacts with the HSS to find the corresponding service and forwards it to the corresponding AS[3] that is running the service. (see steps 1-4 of Figure 4.7)
- AS checks from PGM if it can detect the presence of client ’b’ and since client ’b’ is not registered in the HSS, the AS forwards the query to the Lync Server that is associated with the AS. The Lync server checks for the query. It finds that client ’b’ is in its domain. (see steps 5-9 of Figure 4.7)
- As a Lync client, client ’b’ has the flexibility to set its availability and access permissions. If Client’b’ has restricted outsiders from accessing it, then anyone outside the Ericsson domain’s Lync server is not allowed to send a request to Client ’b’ and a null query will be sent back. (see step 10a of Figure 4.7)
- Considering that Client ’b’ has allowed access to it but is not available when someone contacts it, again a null query is returned back. (see step 10b of Figure 4.7)
- If everything goes fine and client ’b’ is available, the Lync server forwards the request to client ’b’. If client ’b’ accepts the request, a success query is sent back to client ’a’ updating the presence of both the users to be available. Thus completing the searching procedure. (see step 10c of Figure 4.7)

Now for a peer-to-peer file transfer, both clients carry out a negotiation by transferring SDP’s. Since Lync server uses NAT traversal[4], it can easily get its client’s (Client ’b’) ICE candidate addresses. Client ’a’ uses the external TURN/STUN server for NAT traversal. Once the ICE candidates are known by both the client’s, the ICE connectivity check is done to find out the best possible path. After the path is determined (session gets established) the WebRTC DataChannel opens the SCTP streams for file exchange (peer-to-peer) between client’a’ and client’b’ directly.

4.5 Summary

In this chapter we presented our prototype and described the design and methodology used for our implementation. The prototype was implemented between 1st March and 31st March 2013. The prototype is a simple example of a collaboration

[3]In Figure 4.6 we have only one AS server, but in reality IMS can host several AS servers

[4]To traverse firewalls, Lync Server uses the Interactive Connectivity Establishment (ICE) to determine the most direct media path between two endpoints[86].

tool between two browsers with file-sharing and chat features. The main idea behind implementing the prototype was to understand the current development of WebRTC and to create a prototype to test the performance of WebRTC. Apart from that a case study was also presented. It was a simple proposal about integrating enterprise collaborative environments. The case study examined the future of enterprise collaboration with technologies like WebRTC and IMS.

The next chapter covers the performance evaluation of our prototype and compares it with some popular collaboration tools.

Chapter 5

Performance analysis

In this chapter, we make a comparative study of popular file transfer applications and compare them to our own prototypes. First we introduce and describe the test set-ups of all the applications used for this study. Then we evaluate the performance of the applications.

5.1 Scenario

For performance evaluation, we have used eight applications making use of file transfer. They are: Skype [39], Google Talk [36], Facebook [33], Bistri [29], Dropbox [32], Cisco WebEx [41] and our own WebRTC-based implementation on Google Chrome and Firefox Nightly. Except Skype and Google Talk, all the others are Web-based applications.

To measure the performance of the applications, we used several techniques so that we could correctly evaluate their performance and reach a proper result. The performance evaluation is separated into two categories based on the communication architecture of the applications, i.e. peer-to-peer and client-server architecture. This was because, the two categories: peer-to-peer and client-server, have different ways of transferring files. Unlike peer-to-peer, where the file flows continuously between the end users devices, the client-server architecture requires that we upload and download the file to the server. Therefore, for performance evaluation, in the client-server architecture we need to record two times: the upload time (upload time is the time required to transfer a file from the user's local machine to the server or the time required to transfer a file from client to server machine) and the download time (download time is the time required to transfer a file from the server to the client or user's local machine).

$$\text{Bitrate (in Mbps)} = \frac{\text{File size(in Mb)}}{\text{Time (in seconds)}} \tag{5.1}$$

We created five arbitrary text files of sizes 10 kB, 100 kB, 500 kB, 1000 kB, 5000 kB and 10000 kB. We transferred these files between two laptops (one acting as a sender and the other as a receiver) and noted the times taken for the transfers according to the communication architecture. As we already had the time and file size, we calculated the speed using Formula 5.1.

$$\text{Total speed (in Mbps)} = \text{Upload speed (in Mbps)} + \text{Download speed (in Mbps)} \tag{5.2}$$

For peer-to-peer applications we compared the speed of all the applications whereas for client-server applications we made three kinds of observations i.e. the upload speed, the download speed and the total speed (shown in Formula 5.2) based on three time measures (upload time, download time and total time).

We needed to consider the Internet bandwidth at the point of the experiment and calculated a score to compare all these applications on a common base. This was done because the client-server applications used variable Internet bandwidth both for uploading and downloading. Thus a score (shown in Formula 5.3) was generated from the speed and the Internet bandwidth to compare our client-server applications. (Note: In some client to server implementations the value exceeds 100 due to the fact that the speed of the Internet might have changed.)

$$\text{Score (in \%)} = \frac{\text{Speed (in Mbps)}}{\text{Minimum bandwidth available to the application (in Mbps)}} * 100 \tag{5.3}$$

During the experiment the following factors were considered:

- In all the experiments same LAN was shared and they were within the same Ericsson Internet connection.
- The experiment might have some small human errors with timing which can be ignored considering that it does not have any impact on the final result.
- The Internet bandwidth is measured through speedtest.net before the start of each experiment.
- All the other applications running in the machine were shut down to leverage the maximum bandwidth for the test.

Now, we will discuss the set-up of each individual test.

5.1.1 Experimental Set-ups

Evaluation set-up of Bistri

Bistri is a web-based video and file sharing application. It uses XMPP for chat and signaling [7]. It also has file transfer features which work like the transfer of email attachment. It uses a client-server based communication model so shared files need to be uploaded or downloaded. A file can be send/shared with multiple recipients. Bistri uses a file size limit of 10 MB [15].

The experimental set-up (shown in figure 5.2) for performance testing of the file transfer feature in Bistri included two laptops (more specifications about the laptops are available in figure 5.2) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. Files were transferred between the two laptops (from Windows 8 to Mac OS). The time required to upload/download the file to/from the server were measured for further analysis.

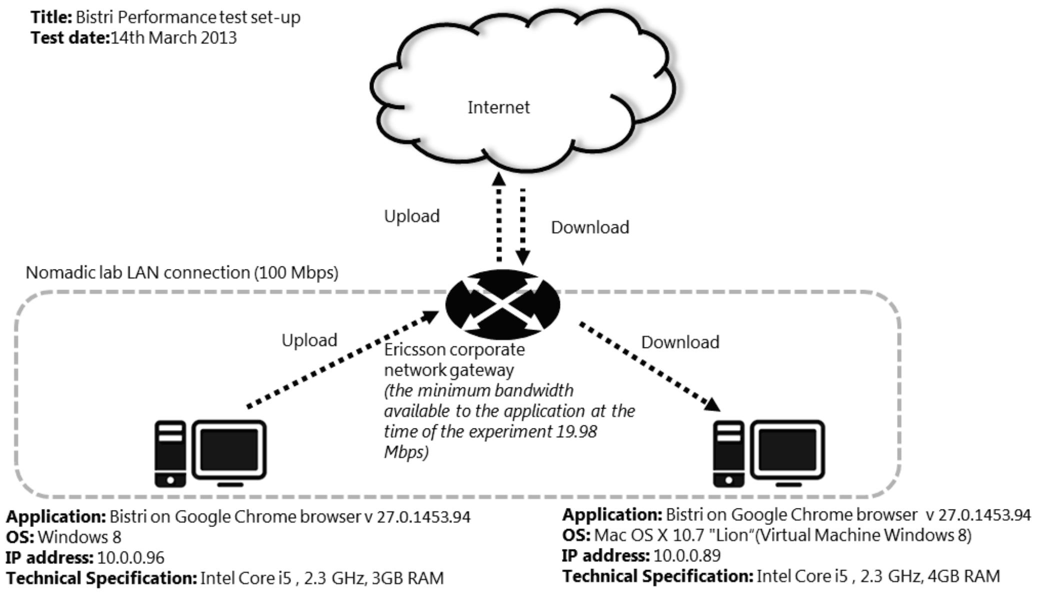
Title: Bistri Performance test set-up
Test date:14th March 2013
Internet
Upload
Download
Nomadic lab LAN connection (100 Mbps)
Upload
Download
Ericsson corporate network gateway
(the minimum bandwidth available to the application at the time of the experiment 19.98 Mbps)
Application: Bistri on Google Chrome browser v 27.0.1453.94
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Bistri on Google Chrome browser v 27.0.1453.94
OS: Mac OS X 10.7 "Lion"(Virtual Machine Windows 8)
IP address: 10.0.0.89
Technical Specification: Intel Core i5 , 2.3 GHz, 4GB RAM

Figure 5.1: Bistri Implementation

Evaluation set-up of Cisco WebEx

Cisco WebEx is a Web-based real-time conferencing system. It combines phone conferencing and video allowing all participants, wherever they are to see and hear the same content in real-time.

Cisco WebEx is based on client-server communication architecture [16]. More information about how it works is out of the scope of this thesis and can be found in reference [16]. File transfer in WebEx is usually done by initiating a session that includes the name of the file and its size. This information is sent to the other user.

The transfer usually occurs inside a secured channel, directly without a upload and download session. This gives the impression of a peer-to-peer session.

The experimental set-up (shown in Figure 5.4) includes two laptops (specifications in Figure 5.4) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. In the experimental setup, we first connected two users. Both users share a common dashboard. When a file is placed for transfer in the dashboard, the local path of the file is retrieved along with the file name and size and this information is placed in the common dashboard. When the other user downloads the file from the dashboard, the file is directly downloaded from the local path of the other machine. This creates the impression of peer-to-peer file sharing. The transfer time consist of only the download time from one machine to the other.

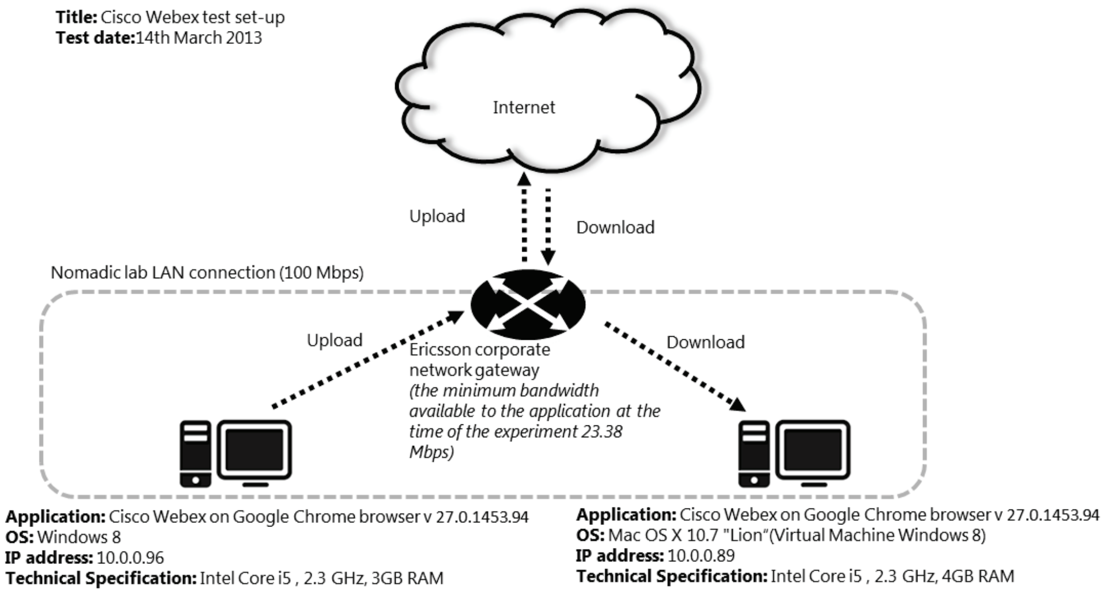
Title: Cisco Webex test set-up
Test date:14th March 2013
Internet
Upload
Download
Nomadic lab LAN connection (100 Mbps)
Upload
Download
Ericsson corporate network gateway
(the minimum bandwidth available to the application at the time of the experiment 23.38 Mbps)
Application: Cisco Webex on Google Chrome browser v 27.0.1453.94
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Cisco Webex on Google Chrome browser v 27.0.1453.94
OS: Mac OS X 10.7 "Lion"(Virtual Machine Windows 8)
IP address: 10.0.0.89
Technical Specification: Intel Core i5 , 2.3 GHz, 4GB RAM

Figure 5.2: Cisco WebEx Implementation

Evaluation set-up of Dropbox

Dropbox is a client-server cloud storage application that, apart from a Web client also has a desktop client. If a file is uploaded to the cloud the desktop client ensures that each machine has its own local copy. In our experiment we used the Web-client of Dropbox for uploading and downloading.

The experimental set-up (shown in Figure 5.5) included two laptops (specifications in Figure 5.5) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. Files were transferred between the two laptops (from Windows 8 to Mac OS). The transfer time required for uploading and downloading to/from the server were measured for analysis.

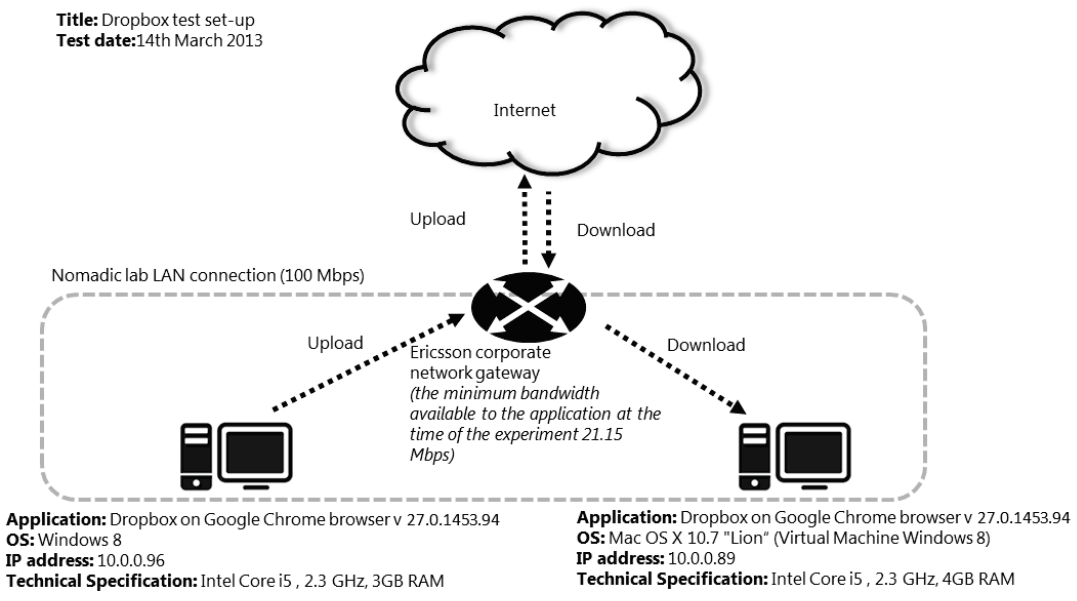
Title: Dropbox test set-up
Test date:14th March 2013
Internet
Upload
Download
Nomadic lab LAN connection (100 Mbps)
Upload
Ericsson corporate network gateway *(the minimum bandwidth available to the application at the time of the experiment 21.15 Mbps)*
Download
Application: Dropbox on Google Chrome browser v 27.0.1453.94
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Dropbox on Google Chrome browser v 27.0.1453.94
OS: Mac OS X 10.7 "Lion" (Virtual Machine Windows 8)
IP address: 10.0.0.89
Technical Specification: Intel Core i5 , 2.3 GHz, 4GB RAM

Figure 5.3: Dropbox Implementation

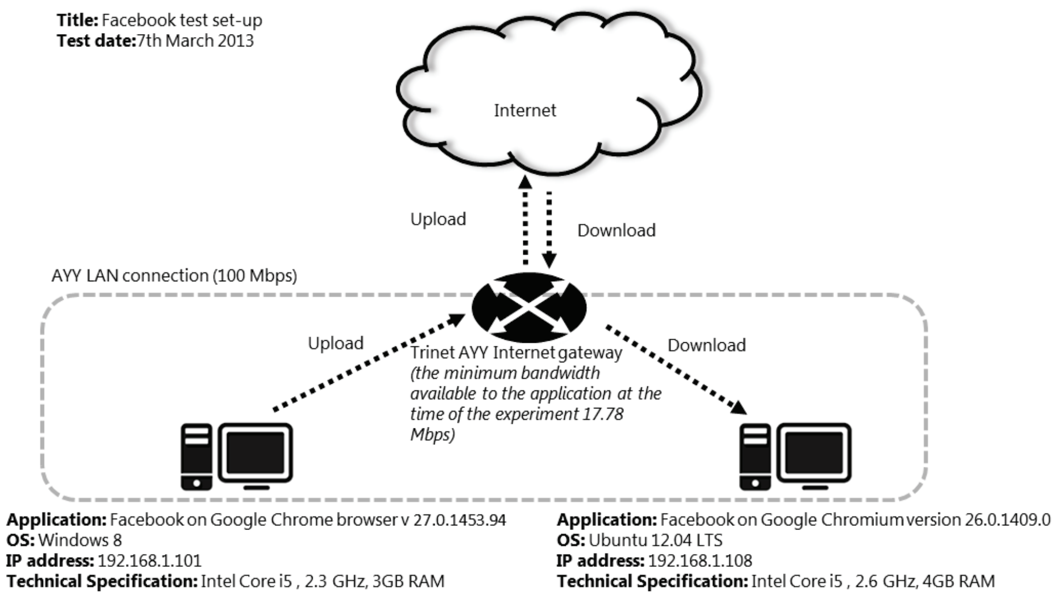
Title: Facebook test set-up
Test date:7th March 2013
Internet
Upload
Download
AYY LAN connection (100 Mbps)
Upload
Trinet AYY Internet gateway *(the minimum bandwidth available to the application at the time of the experiment 17.78 Mbps)*
Download
Application: Facebook on Google Chrome browser v 27.0.1453.94
OS: Windows 8
IP address: 192.168.1.101
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Facebook on Google Chromium version 26.0.1409.0
OS: Ubuntu 12.04 LTS
IP address: 192.168.1.108
Technical Specification: Intel Core i5 , 2.6 GHz, 4GB RAM

Figure 5.4: Facebook Implementation

Evaluation set-up of Facebook

Facebook is the most popular social networking site in the world [45]. Facebook has a chat feature that comes with file transfer abilities in the form of attachments. One cannot transfer files larger than 25 MB. The file transfer is client- server based.

The experimental set-up (shown in Figure 5.6) included two laptops (specifications in Figure 5.6) that were connected to the LAN which had access to Aalto University's

Internet gateway for Internet connectivity. Files were transferred between the two Laptops (from Windows 8 to Ubuntu OS), the transfer time required for uploading and downloading to/from the server were measured for analysis.

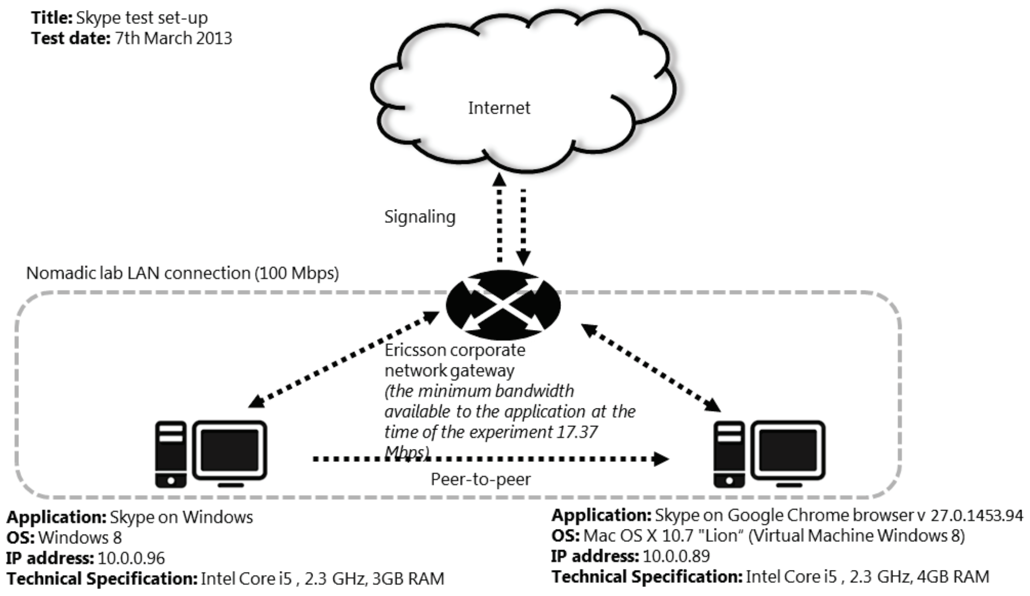
Title: Skype test set-up
Test date: 7th March 2013
Internet
Signaling
Nomadic lab LAN connection (100 Mbps)
Ericsson corporate network gateway
(the minimum bandwidth available to the application at the time of the experiment 17.37 Mbps)
Peer-to-peer
Application: Skype on Windows
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Skype on Google Chrome browser v 27.0.1453.94
OS: Mac OS X 10.7 "Lion" (Virtual Machine Windows 8)
IP address: 10.0.0.89
Technical Specification: Intel Core i5 , 2.3 GHz, 4GB RAM

Figure 5.5: Skype Implementation

Evaluation set-up of Skype

Skype is A popular video and voice application. It also includes file transfer and chat features. Skype is a hybrid communication architecture that uses both client-server and peer-to-peer architecture. The client-server architecture is used to keep information about the users for presence display and for signaling. File transfer takes place using peer-to-peer and the media traffic flowed directly between them over UDP[47].

The experimental set-up (shown in Figure 5.7) included two laptops (specifications in Figure 5.7) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. Files were transferred between the two Laptops (from Windows 8 to Ubuntu OS). The transfer time for each file size was measured for further analysis.

Evaluation setup of Google Talk

Google Talk is a client-server chat application that uses an open protocol, eXtensible Messaging and Presence Protocol (XMPP) and can be accessed by any client that support XMPP. Later a file transfer feature was added, which was pure peer-to-peer with no restriction on the file type or size. [22]

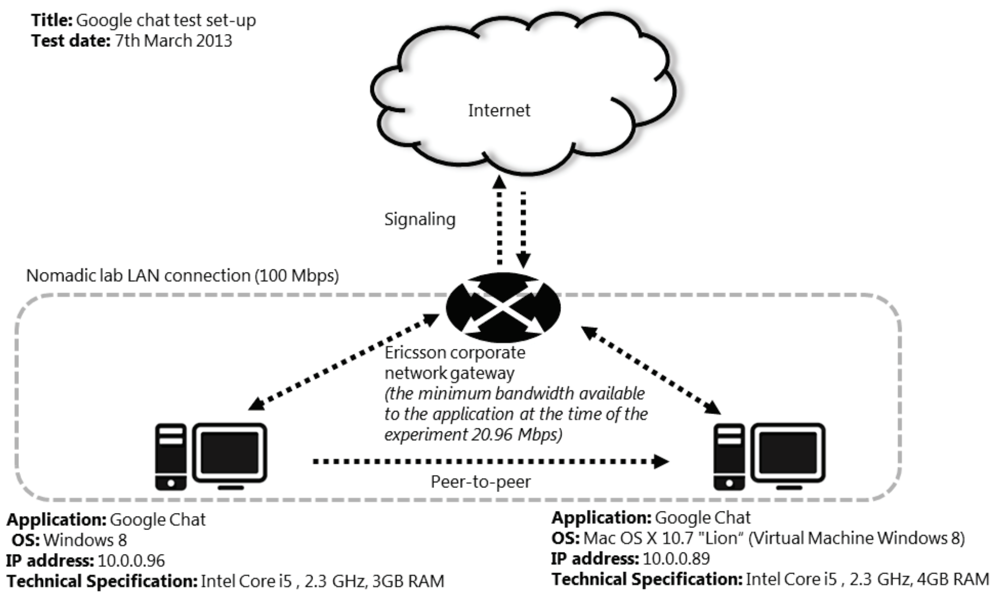
Title: Google chat test set-up
Test date: 7th March 2013
Internet
Signaling
Nomadic lab LAN connection (100 Mbps)
Ericsson corporate network gateway *(the minimum bandwidth available to the application at the time of the experiment 20.96 Mbps)*
Peer-to-peer
Application: Google Chat
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Google Chat
OS: Mac OS X 10.7 "Lion" (Virtual Machine Windows 8)
IP address: 10.0.0.89
Technical Specification: Intel Core i5 , 2.3 GHz, 4GB RAM

Figure 5.6: Google Talk Implementation

The experimental set-up (shown in Figure 5.8) included two laptops (specifications in figure 5.8) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. Files were transferred between the two Laptops (from Windows 8 to Mac OS). The file transfer time for each file size was measured for further analysis.

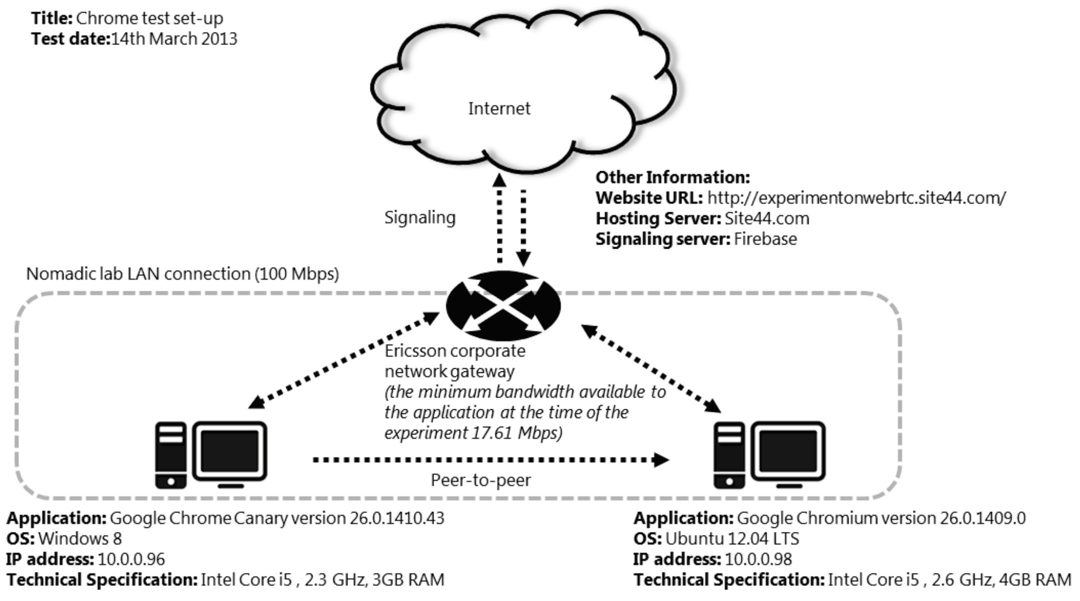
Title: Chrome test set-up
Test date: 14th March 2013
Internet
Other Information:
Website URL: http://experimentonwebrtc.site44.com/
Hosting Server: Site44.com
Signaling server: Firebase
Signaling
Nomadic lab LAN connection (100 Mbps)
Ericsson corporate network gateway *(the minimum bandwidth available to the application at the time of the experiment 17.61 Mbps)*
Peer-to-peer
Application: Google Chrome Canary version 26.0.1410.43
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Google Chromium version 26.0.1409.0
OS: Ubuntu 12.04 LTS
IP address: 10.0.0.98
Technical Specification: Intel Core i5 , 2.6 GHz, 4GB RAM

Figure 5.7: Chrome Implementation

Evaluation setup of Chrome

For this experiment, we used Google Chrome Canary (version 25.0.1410.43) to run our Web application. The experimental set-up (shown in Figure 5.8) included two laptops (specifications in Figure 5.8) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. This is a peer-to-peer communication architecture. Files were transferred between the two Laptops (from Windows 8 to Mac OS). The transfer time for each file size was measured for further analysis.

While transferring the file it was noticed that Chrome was not able to transfer large files (>1500kB) and that it crashes on every attempt. Thus to check the file size limit of Chrome, we created another three files of sizes 1500kB, 1700 kB and 1800 kB.

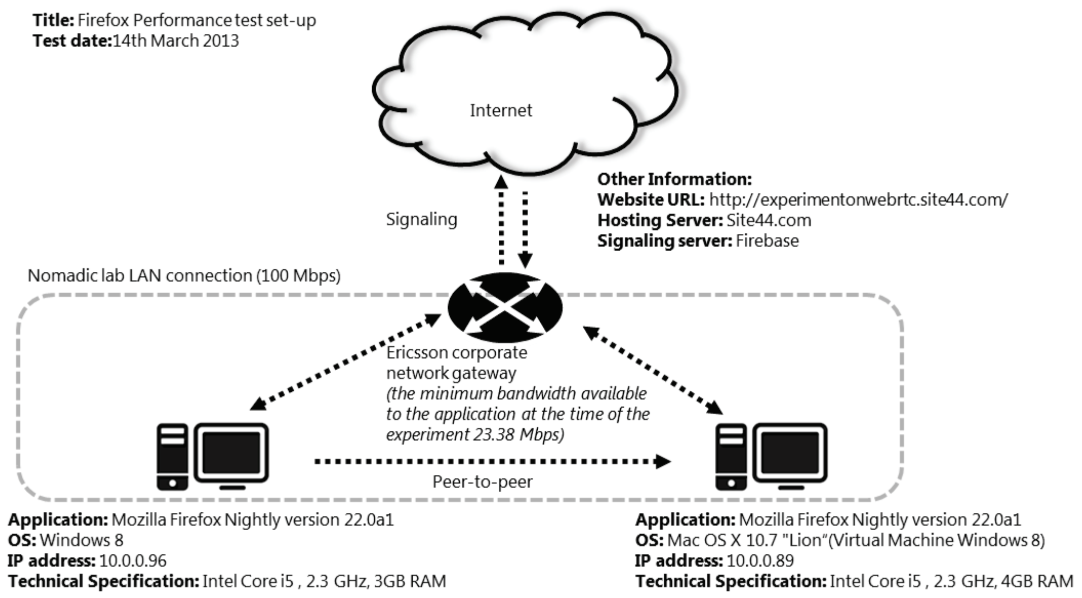
Title: Firefox Performance test set-up
Test date:14th March 2013
Internet
Signaling
Other Information:
Website URL: http://experimentonwebrtc.site44.com/
Hosting Server: Site44.com
Signaling server: Firebase
Nomadic lab LAN connection (100 Mbps)
Ericsson corporate network gateway
(the minimum bandwidth available to the application at the time of the experiment 23.38 Mbps)
Peer-to-peer
Application: Mozilla Firefox Nightly version 22.0a1
OS: Windows 8
IP address: 10.0.0.96
Technical Specification: Intel Core i5 , 2.3 GHz, 3GB RAM
Application: Mozilla Firefox Nightly version 22.0a1
OS: Mac OS X 10.7 "Lion"(Virtual Machine Windows 8)
IP address: 10.0.0.89
Technical Specification: Intel Core i5 , 2.3 GHz, 4GB RAM

Figure 5.8: Firefox Implementation

Evaluation set-up of Firefox

The Firefox experiment used is a peer-to-peer communication architecture. For this experiment, we used Mozilla Firefox Nightly (version 22.0a1) to run our Web application. The experimental setup (shown in Figure 5.9) included two laptops (specifications in Figure 5.9) that were connected to the LAN which had access to Ericsson's Corporate network gateway for Internet connectivity. Files were transferred between the two Laptops (from Windows 8 to Mac OS). The file transfer time for each file size was measured for further analysis.

5.2 Observations made

5.2.1 Evaluation of performance for peer-to-peer model

We choose four peer-to-peer applications for the test cases i.e. Skype, Google Talk, and our WebRTC Chrome and Firefox implementations. As mentioned earlier, in the performance evaluation of the peer-to-peer communication architecture we directly compare the speed of file transfer. Table A.1 (in Appendix A) shows the data after calculation (Formula 5.1). Note that Chrome was unable to transfer files above 1500kB and so the corresponding fields after 1000kB (Table A.1) are left blank.

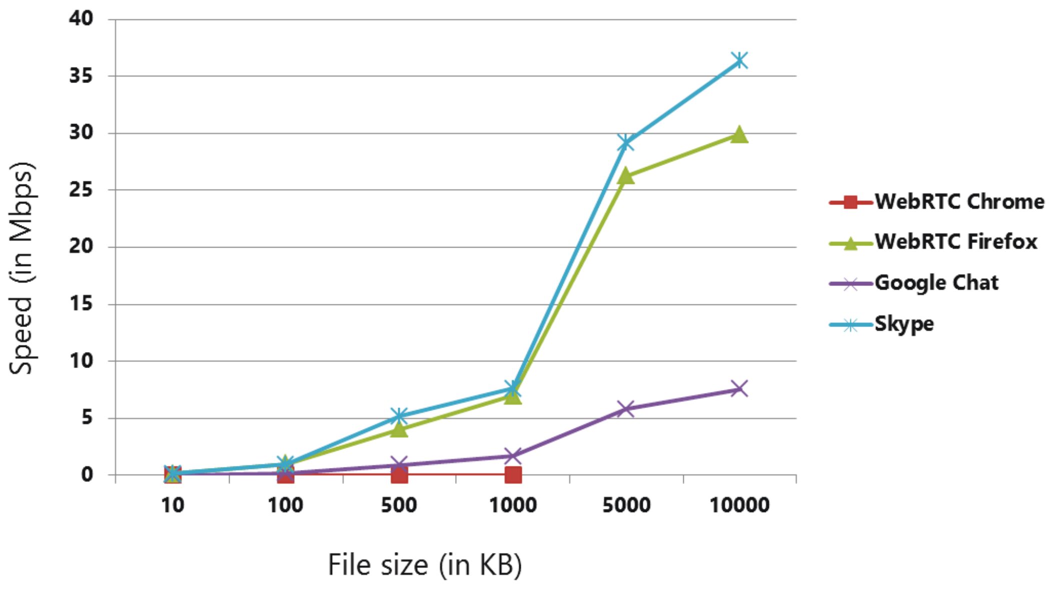

Figure 5.9: Comparison of speed of various peer-to-peer applications

Figure 5.13, shows a graph comparing peer-to-peer applications in terms of file transfer efficiency. The result show : Skype (fastest) > WebRTC Firefox > Google Chat > WebRTC Chrome (slowest).

We observe that Skype is the fastest and our WebRTC Chrome implementation is the slowest. As WebRTC is still developing, we believe that its performance will improve. On the other hand, our WebRTC Firefox implementation is the second fastest file-transfer application after Skype. Google Chat showed steady growth as the file size increase. It is interesting to observe that the difference of file transfer speeds between the four applications varies as the file size increases, this might be due to the reason that applications have different algorithms and protocols for transferring large files. However, Skype performance changes depending upon how the communication between the peers are done (i.e. may be when peers are behind the NATs or indirections via third party nodes are necessary).

5.2.2 Evaluation of performance for client-server model

s We have chosen four client-server applications for the test cases: i.e. Facebook, Dropbox, Bistri and Cisco WebEx. As mentioned earlier, in the performance evaluation of client-server applications, we compare the performance by calculating a score using Formula 5.3. Therefore, for performance evaluation in the client-server architecture we made three observations i.e. one each for upload, download and total score. We analysed each set of observations to understand the performance. For Cisco WebEx no upload time or download time is presented. This is because although it is claimed to use a client-server architecture, it showed some behaviour of direct transfer once a session was established. Thus, we included it only in the total comparison (Figure 5.12) of client-server performance. Below are the results of all the three kinds of analyses:

Comparison of client-server upload performance

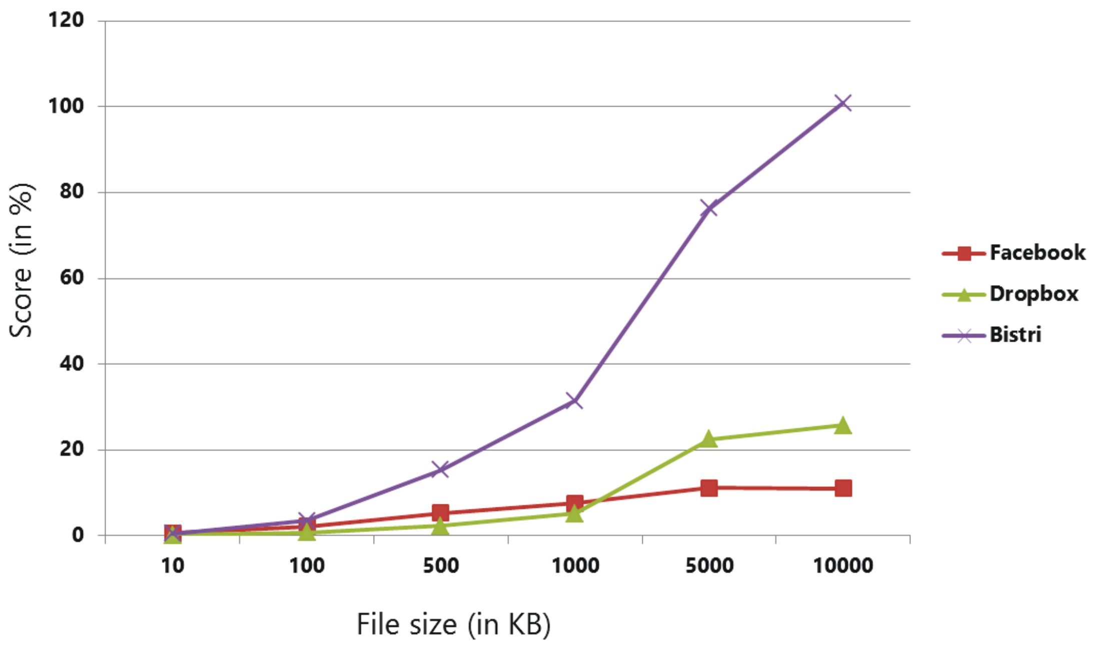

Figure 5.10: Comparison of client-server upload performance

Figure 5.11 shows a graph comparing the client-server applications' upload performance in terms of efficiency in file transfer. The results are: Bistri(fastest) > Dropbox > Facebook(slowest).

Comparison of client-server download performance

Figure 5.12 shows a graph of client-server applications' download performance comparision in terms of efficiency in file transfer. The results are: Dropbox (fastest)>

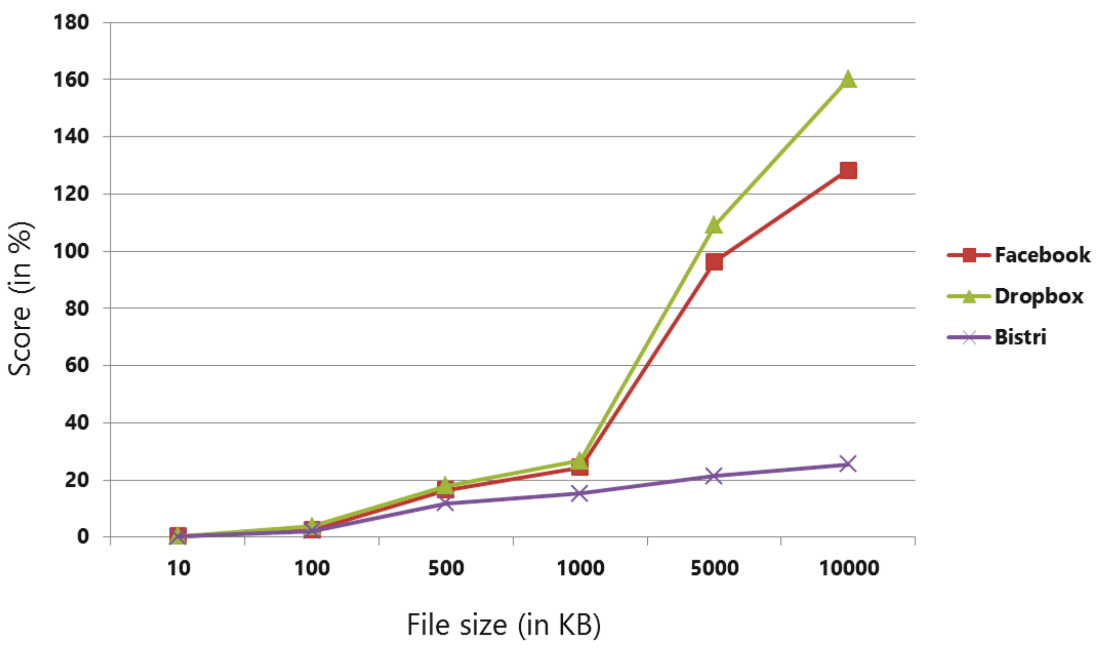

Figure 5.11: Comparison of client-server download performance

Facebook > Bistri(slowest). The performance of Bistri seems to improve slightly as file size increases while this is found to be true for both Dropbox and Facebook.

Comparison of client-server total performance

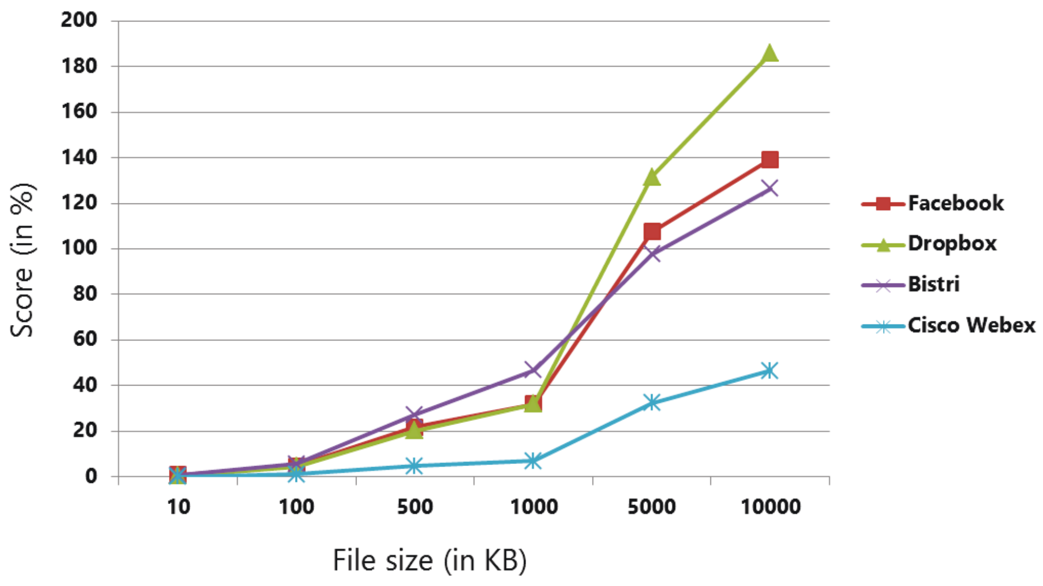

Figure 5.12: Comparison of client-server total performance

Figure 5.13 shows a comparison of client-server applications' total performance, in terms of efficiency in file transfer. The results are: Dropbox > Facebook > Cisco WebEx > Bistri. We found that Dropbox is the fastest among the four compared client-server application. Its performance follows a similar pattern as Facebook and Bistri as the file size increases. Cisco WebEx is the slowest of the four. The reason might be that it continuously transfers files with a server in between.

5.3 Analysis of results

We compared (comparision data are in Appendix B) Google talk (from the above results, one of the slowest peer-to-peer applications) with Dropbox (from the above results, the fastest client-server application) to evaluate our WebRTC Firefox implementation. From this comparision we observed that after a while Google Talk shows a steady growth in speed while the speed of file transfer in Dropbox increases drastically (shown in Figure B.1). According to theory peer-to-peer architecture should be faster than the client-server architecture [63]. This is case specific and may not always be true.

Due to the on-going development process of WebRTC which does not have TURN support, our protoype was limited within the LAN and thus could not be tested over the Internet. But the purpose of measuring the client-server and the peer-to-peer model on different scales was to compare them and also to understand the mechanisms behind various popular file transfer tools.

For the WebRTC Firefox implementation, the default number of SCTP streams is 16. Recent developments claim that this can be increased up to 100. With such a development we can assume that WebRTC's performance would be much faster in the future with low-latency. The reason behind the slow transfer of the WebRTC Chrome Implementation is that WebRTC has not yet implemented SCTP streams in Chrome and can make use of only one RTP stream at a time. Also, since it cannot transfer blobs and the file transfer size is limited, larger files must be divided into several smaller packets. This introduces a high transmission overhead resulting in packet loss, congestion and low performance. Furthermore, it is important to note that the test was done in a limited environment thus we have not considered or compared the Quality of Service (QoS) but evaluated the general performance.

5.4 Summary

In this chapter we presented the scenario and various experimental set-ups for our performance evaluation test. We used eight file transfer applications including two of our own prototypes. The evaluation comprised of both peer-to-peer and the client-server communication architectures. The evaluations discussed some critical performance issues in the WebRTC Chrome implementation due to RTP stream for data transfer.

Chapter 6

Conclusion

WebRTC aims to allow browsers to natively support interactive peer-to-peer real-time communications. In particular, WebRTC specifies a set of protocols that support transportation of media data, such as audio/video, and non-media data along the most direct possible path between two WebRTC clients. IETF and W3C are responsible for standardization of WebRTC.

In this thesis we investigated the WebRTC DataChannel during its on-going development. We tried to understand the behaviour of the WebRTC DataChannel from the point of view of developers, enterprises, Web services and service providers. This also included a wide range of studies on related concepts such as IP multimedia subsystem, signaling protocols, NAT traversal, collaboration tools etc.

A DataChannel is made up of SCTP over DTLS over UDP with native NAT traversal mechanisms. DTLS is responsible for providing security, confidentiality, and source authentication. It allows many possible uses with a wide range of possibilities such as P2P Web conferencing, multiplayer gaming, content delivery network etc.

We designed and implemented a prototype of a collaboration tool to understand the WebRTC features, especially WebRTC's DataChannel API and also how Chrome and Firefox browsers have implemented the WebRTC protocol stack. The design required a clear understanding of the HTML5, WebSockets and WebRTC JavaScript APIs. We then evaluated our prototype to analyse the datachannel performance. From a developer's perspective, WebRTC adds easy implementation procedures using simple JavaScripts API's. Also, the freedom of choosing signaling procedures makes it flexible.

A case study was also done in this thesis for the integration of enterprise collaboration tools with WebRTC applications for universal collaboration. The design included an IMS as the infrastructure to support communication between a 3G-IMS based smartphone with a WebRTC enabled browser (considering that the smartphone browser will support WebRTC as predicted by [53]) and Microsoft Lync applications. An approach was proposed to integrate Lync (an enterprise collaboration tool with its own enterprise Lync server and client) and a smartphone based Web-application(a non-enterprise application) with IMS for signaling and presence display and using the WebRTC DataChannel for data exchange. Lync and IMS are based on SIP. The Microsoft Unified Communications Open Interoperability Program eases the process of signaling and presence display.

Finally, the performance of the WebRTC DataChannel was analysed along with various other file transfer tools. Our WebRTC Firefox prototype showed good performance considering that WebRTC implementation in Firefox was still under development at the time of writing this thesis, whereas, WebRTC Chrome prototype had various issues making it still a very under developed and thus unsuitable for file transfer.

The result from this thesis does take a step towards understanding WebRTC and its key features. Our case study demonstrated that it is feasible to integrate DataChannel to existing enterprise collaboration tools. IMS signaling can be used to facilitate setting up a DataChannel between users in different collaborative environments. We showed that DataChannel implementation in Firefox is good enough compared to existing tools, whereas Chrome is still immature. Nevertheless, it is not enough to draw any conclusion. The scope of this thesis is a very fundamental and hopefully it will become a starting point in understanding the development of the WebRTC DataChannel.

Future work

The main challenges that WebRTC faces today include interoperability issues between various browsers. This is due to the codec battle between VP8 and H.264. In this battle, browser vendors like Google and Mozilla, on the one hand, support VP8 due to it being royalty-free (without any license fees), whereas Apple, Microsoft, and Cisco on the other hand, favor H.264 since many current devices have H.264 optimized hardware. The on-going standardisation process, Microsoft's own proposal and Apple's unresponsive behaviour towards WebRTC are some other major barriers in making WebRTC the de-facto standard for real-time communication on the Web. [81]

The on-going developments have not stopped the early adopters and developers from building innovative applications using WebRTC's technology. It can be expected that once the standardisation process is complete WebRTC will primarily be adopted by existing Web companies as an enhancement to their products. Some new companies are already exploiting WebRTC features that include PeerCDN (a P2P-based content delivery network that crowd sources the delivery of data on a website using WebRTC), Bistri (a voice/video conferencing tool that pulls all the user's contacts and their presence status and display it on their website using WebRTC) and Plivo (a cloud platform now supporting WebRTC based telephony applications) [80]

WebRTC has a bright future considering the technological width of the companies involved, spanning both traditional telecoms, enterprise communication companies and Web based companies. According to Disruptive Analysis [53], by the end of

2016 there will be over 3 billion WebRTC-enabled devices although not all 'enabled' devices will be actively used for WebRTC. Because of auto-updating browsers such as Chrome and Firefox, this transition will be led by PCs followed by smartphones and tablets.

Bibliography

[1] Components of a SIP-Enabled Network. [online] Available at: `http://goo.gl/rS9nx`. Accessed May 22, 2013.

[2] FileReader API. [online] Available at: `https://developer.mozilla.org/en-US/docs/Web/API/FileReader`. Accessed May 29, 2013.

[3] HTML5 browser test. [online] Available at: `http://html5test.com/results/desktop.html`. Accessed May 24, 2013.

[4] HTML5 History. [online] Available at: `http://www.w3.org/TR/2011/WD-html5-20110525/introduction.html#history-1`. Accessed May 24, 2013.

[5] HTML5 W3C news. [online] Available at: `http://www.w3.org/News/2012#entry-9667`. Accessed May 24, 2013.

[6] Internet Engineering Task Force. http://www.ietf.org/about/.

[7] Interview: Bistri on WebRTC for Video Chat Services. [online] Available at: `http://www.w3.org/QA/2013/02/interview_bistri_on_webrtc_for.html`. Accessed March 22, 2013.

[8] Introduction to IP Multimedia Subsystem (IMS), Part 2: Building an IMS infrastructure. [online] Available at: `http://www.ibm.com/developerworks/webservices/library/ws-soa-ipmultisub2/#grouplist`. Accessed May 22, 2013.

[9] Introduction to Network Address Translation (NAT) and NAT Traversal. [online] Available at: `http://www.pjsip.org/pjnath/docs/html/group__nat__intro.htm`. Accessed May 22, 2013.

[10] Is javascript the future of programming? [online] Available at: `http://mashable.com/2012/11/12/javascript/`. Accessed May 24, 2013.

[11] JavaScript. [online] Available at: `http://en.wikipedia.org/wiki/JavaScript`. Accessed May 24, 2013.

[12] JAVASCRIPT WEB APIS. [online] Available at: `http://www.w3.org/standards/webdesign/script.html`. Accessed May 24, 2013.

[13] jQuery. [online] Available at: http://jquery.com/. Accessed November 07, 2013.

[14] New in JavaScript 1.8.5. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/1.8.5. Accessed November 07, 2013.

[15] New release : file sharing on Bistri. [online] Available at: http://blog.bistri.com/page/5. Accessed June 10, 2013.

[16] Overview of IM Logging and Archiving. [online] Available at: http://www.webex.com/webexconnect/orgadmin/help/index.htm?toc.htm?cs_IM_Archiving.htm. Accessed June 2, 2013.

[17] SIP, RTP, and XMPP in the Emerging Real-Time Internet. [online] Available at: http://www.carahsoft.com/resources/Jabber/Jabber_Inc_SIP_RTP_XMPP_White_Paper.pdf. Accessed May 22, 2013.

[18] Twitter Bootstrap. [online] Available at: http://getbootstrap.com/2.3.2/. Accessed November 07, 2013.

[19] WebRTC. [online] Available at: http://www.webrtc.org/s. Accessed May 11, 2013.

[20] WebRTC for Firefox. [online] Available at: http://www.webrtc.org/firefox. Accessed March 12, 2013.

[21] WebRTC Google group - Issue 1315: Data channels should be ignored by receiver if the data channel constraint has not been set. Accessed March 03, 2013.

[22] What's New on Google Talk? [online] Available at: http://www.google.com/talk/whatsnew_more.html. Accessed June 3, 2013.

[23] World Wide Web Consortium. http://www.w3.org/Consortium/.

[24] XMPP Technologies: Jingle. [online] Available at: http://xmpp.org/about-xmpp/technology-overview/jingle/. Accessed May 22, 2013.

[25] XMPP Technologies Overview. [online] Available at: http://xmpp.org/about-xmpp/technology-overview/. Accessed May 22, 2013.

[26] White Paper IMS -IP Multimedia Subsystem. [online] Available at: http://www.citmo.net/library/Ericsson%20IMS.pdf, 2004.

[27] Introduction to IMS - White Paper, 2007. Accessed June 15, 2013.

[28] NAT Traversal White Paper, 2013. Accessed May 11, 2013.

[29] Bistri. [online] Available at: https://bistri.com/, 2013.

[30] Chrome Canary version 26.0.1410.43. [online] Available at: https://www.google.com/intl/en/chrome/browser/canary.html, 2013.

[31] Cisco UC Integration(TM) for Microsoft Lync, 2013. Accessed June 2, 2013.

[32] Dropbox. [online] Available at: https://www.dropbox.com/, 2013.

[33] Facebook. [online] Available at: https://www.facebook.com/, 2013.

[34] Firebase. [online] Available at: http://www.firebase.com/, 2013.

[35] Firefox Nightly version 22.0a1, 2013.

[36] Google Talk Beta. [online] Available at: https://support.google.com/talk/, 2013.

[37] Lync customer stories, 2013.

[38] Site44. [online] Available at: http://www.site44.com/, 2013.

[39] Skype version 6.1.0.129. [online] Available at: http://www.skype.com/en/, 2013.

[40] Web-based Lync client by Orbitone. [online] Available at: http://lyncpresence.orbitone.com/, 2013. Accessed June 12, 2013.

[41] Webex. [online] Available at: http://www.webex.com/, 2013.

[42] Alvestrand, H. Overview: Real Time Protocols for Brower-based Applications draft-ietf-rtcweb-overview-06, Feb. 2013.

[43] Androutsellis-Theotokis, S., and Spinellis, D. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv. 36*, 4 (Dec. 2004), 335–371.

[44] ap Eriksson, G., and Håkansson, S. WebRTC: enhancing the web with real-time communication capabilities. [online] Available at: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2012/er-webrtc-html5.pdf. Accessed May 11, 2013.

[45] Banks, E. Facebook is most popular social network for all ages; linkedin is second. [online] Available at: http://mashable.com/2011/11/04/facebook-most-popular-forrester/. Accessed June 20, 2013.

[46] Barkai, D. *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net.* Intel Press, 2001.

[47] Baset, S. A., and Schulzrinne, H. G. An analysis of the skype peer-to-peer internet telephony protocol. Tech. rep., 2004.

[48] Becke, M., Rathgeb, E. P., Werner, S., Rungeler, I., Tuxen, M., and Stewart, R. Data Channel Considerations for RTCWeb. *IEEE Communications Magazine* (2013), 34–41.

[49] Bergin, J. Event programming fundamentals. [online] Available at: `http://csis.pace.edu/~bergin/Java/eventfundamentals.html`. Accessed May 25, 2013.

[50] Bergkvist, A., Burnett, D. C., Jennings, C., and Narayanan, A. Tech. rep.

[51] Berners-Lee, T., and Fischetti, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Paw Prints, 2008.

[52] Broman, L. IMS platform prototype. Master's thesis, Luleå University of Technology.

[53] Bubley, D. "webrtc market status and forecasts- the hype is justified: it will change telecoms". *Disruptive analysis* (2012).

[54] Camarillo, G., and García-Martín, M. *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*. Wiley, 2009.

[55] Camarillo, G., and IAB. Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. RFC 5694 (Informational), Nov. 2009.

[56] Carter, M. HTTP Streaming and Internet Explorer. [online] Available at: `http://cometdaily.com/2007/10/25/http-streaming-and-internet-explorer/`. Accessed May 25, 2013.

[57] Corporation, M. About Microsoft Lync. [online] Available at: `http://mslync.blogspot.com/`. Accessed June 12, 2013.

[58] Corporation, M. UCC API: What is it and how does one write an application using it? [online] Available at: `http://blogs.technet.com/b/lync/archive/2008/05/13/ucc-api-what-is-it-and-how-does-one-write-an-application-using-it.aspx`. Accessed June 12, 2013.

[59] Corporation, M. Unified Communications Open Interoperability Program. [online] Available at: `http://technet.microsoft.com/en-us/lync/gg236602.aspx`. Accessed June 12, 2013.

[60] Crawford, S. How html5 works. [online] Available at: `http://computer.howstuffworks.com/html-five1.htm`. Accessed October 1, 2013.

[61] Daniel, A. Difference Between SIP and XMPP (Jabber). [online] Available at: `http://www.differencebetween.com/difference-between-sip-and-xmpp-jabber//`. Accessed May 22, 2013.

[62] Dutton, S. Getting started with WebRTC. [online] Available at: `http://www.html5rocks.com/en/tutorials/webrtc/basics/`. Accessed March 15, 2013.

[63] Educause. Things you should know about p2p. [online] Available at: `http://net.educause.edu/ir/library/pdf/est0901.pdf`, 2009. Accessed June 14, 2013.

[64] Egevang, K., and Francis, P. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[65] Eric Bidelman. Capturing Audio and Video in HTML5. [online] Available at: `http://www.html5rocks.com/en/tutorials/getusermedia/intro/#toc-round3`. Accessed May 25, 2013.

[66] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266, 6585.

[67] Ford, B., Srisuresh, P., and Kegel, D. Peer-to-peer communication across network address translators. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2005), ATEC '05, USENIX Association, pp. 13–13.

[68] Freeman, E., and Robson, E. *Head First HTML5 Programming: Building Web Apps with JavaScript*. A brain-friendly guide. O'Reilly Media, Incorporated, 2011.

[69] Garcia-Martin, M., Belinchon, M., Pallares-Lopez, M., Canales-Valenzuela, C., and Tammi, K. Diameter Session Initiation Protocol (SIP) Application. RFC 4740 (Proposed Standard), Nov. 2006.

[70] Gong, L. Jxta: a network programming environment. *Internet Computing, IEEE 5*, 3 (2001), 88–95.

[71] Handley, M., Jacobson, V., and Perkins, C. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[72] Hickson, I. HTML5: A vocabulary and associated APIs for HTML and XHTML. Tech. rep., W3C, May 2011. http://www.w3.org/TR/2011/WD-html5-20110525/.

[73] Inc., M. Browser. [online] Available at: `http://mashable.com/category/browser/`. Accessed May 11, 2013.

[74] Inc., W. F. Microsoft Lync. [online] Available at: http://en.wikipedia.org/wiki/Microsoft_Lync. Accessed June 3, 2013.

[75] Inc., W. F. Microsoft Lync Server. [online] Available at: http://en.wikipedia.org/wiki/Microsoft_Lync_Server. Accessed June 12, 2013.

[76] Inc., W. F. World Wide Web. [online] Available at: http://en.wikipedia.org/wiki/World_Wide_Web. Accessed May 3, 2013.

[77] Ingo, H. Session Initiation Protocol (SIP) and other Voice over IP (VoIP) protocols and applications. [online] Available at: http://goo.gl/8s8ni. Accessed May 22, 2013.

[78] Jennings, C., Hardie, T., and Westerlund, M. Real-Time Communications for the Web. *IEEE Communications Magazine* (2013), 20–26.

[79] Jones, P. M. BREAD, not CRUD. [online] Available at: http://paul-m-jones.com/archives/291. Accessed September 21, 2013.

[80] Keating, T. "2013 - The Year of WebRTC".

[81] Keating, T. WebRTC Challenges.

[82] Kinder, N. IMS Overview and the Unified Carrier Network. [online] Available at: http://www.iec.org/newsletter/sept06_2/analyst_corner.pdf. Accessed May 22, 2013.

[83] Lewandowski, S. M. Frameworks for component-based client/server computing. *ACM Comput. Surv. 30*, 1 (Mar. 1998), 3–27.

[84] Mahy, R., Matthews, P., and Rosenberg, J. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), Apr. 2010.

[85] MakeUseOf. 5 Free Browser-Based P2P File Sharing Sites With No Size Limits. [online] Available at: http://www.makeuseof.com/tag/5-free-browserbased-p2p-file-sharing-sites-filesize-limits/. Accessed September 21, 2013.

[86] Maximo, R. Microsoft Lync Server 2010: Lync Edge Server Security. [online] Available at: http://technet.microsoft.com/en-us/magazine/hh272676.aspx. Accessed December 08, 2013.

[87] McAfee, A. *Enterprise 2.0: New Collaborative Tools For Your Organization's Toughest Challenges.* Harvard Business School Publishing India Pvt. Limited, 2009.

[88] Mcafee, A. P. "Enterprise 2.0: The Dawn of Emergent Collaboration" . Reprint 47306. *MIT Sloan Management Review 47*, 3 (2006), 21–28.

[89] Ong, L., and Yoakum, J. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May 2002.

[90] Oppliger, R. Internet security: firewalls and beyond. *Commun. ACM 40*, 5 (May 1997), 92–102.

[91] Perkins, C., Westerlund, M., and Ott, J. Web Real-Time Communication (WebRTC): Media Transport and Use of RTP draft-ietf-rtcweb-rtp-usage-06, Feb. 2013.

[92] Perreault, S. NAT and Firewall Traversal with STUN / TURN / ICE. [online] Available at: `http://www.viagenie.ca/publications/2008-09-24-astricon-stun-turn-ice.pdf`. Accessed May 22, 2013.

[93] Pilgrim, M. *HTML5: Up and Running.* O'Reilly Media, 2010.

[94] Raggett, D., Hors, A. L., and Jacobs, I. HTML 4.01 Specification. Tech. rep., W3C, 1999. http://www.w3.org/TR/2004/REC-xml-20040204.

[95] Raggett, D., Hors, A. L., and Jacobs, I. HTML Canvas 2D Context, Level 2. Tech. rep., W3C, May 2013. http://www.w3.org/TR/2dcontext2/.

[96] Rosenberg, J. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), Apr. 2010. Updated by RFC 6336.

[97] Rosenberg, J., Mahy, R., Matthews, P., and Wing, D. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), Oct. 2008.

[98] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878.

[99] Rouse, M. Groupware. [online] Available at: `http://searchdomino.techtarget.com/definition/groupware`. Accessed May 21, 2013.

[100] Rouse, M. What is Microsoft Lync Server ? [online] Available at: `http://whatis.techtarget.com/definition/Microsoft-Lync-Server`. Accessed June 12, 2013.

[101] Russell, T. *The IP Multimedia Subsystem (IMS): Session Control and Other Network Operations*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 2008.

[102] Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921 (Proposed Standard), Oct. 2004. Obsoleted by RFC 6121.

[103] SALVATORE LORETO. WebRTC:Real-Time Communication between Browsers. [online] Available at: `http://www.sloreto.com/slides/Aalto022013WebRTC/slides.html#1`. Accessed May 25, 2013.

[104] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. A measurement study of peer-to-peer file sharing systems.

[105] STEFAN ALUND. Bowser - First WebRTC-enabled mobile browser. [online] Available at: `https://labs.ericsson.com/blog/bowser-the-world-s-first-webrtc-enabled-mobile-browser`. Accessed May 25, 2013.

[106] STEWART, R. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), Sept. 2007. Updated by RFCs 6096, 6335.

[107] STEWART, R., XIE, Q., MORNEAULT, K., SHARP, C., SCHWARZBAUER, H., TAYLOR, T., RYTINA, I., KALLA, M., ZHANG, L., AND PAXSON, V. Stream Control Transmission Protocol. RFC 2960 (Proposed Standard), Oct. 2000. Obsoleted by RFC 4960, updated by RFC 3309.

[108] TSAHI LEVENT-LEVI. What is WebRTC? [online] Available at: `http://bloggeek.me/webrtc/`. Accessed May 25, 2013.

[109] TUEXEN, M., SEGGELMANN, R., AND RESCORLA, E. Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP). RFC 6083 (Proposed Standard), Jan. 2011.

[110] UBERTI, J. WebRTC Google I/O 2012 . [online] Available at: `http://www.youtube.com/watch?v=E8C8ouiXHHk`. Accessed May 25, 2013.

[111] UBL, M., AND KITAMURA, E. Introducing WebSockets: Bringing Sockets to the Web. [online] Available at: `http://www.html5rocks.com/en/tutorials/websockets/basics/`. Accessed May 25, 2013.

[112] WILLIAMS, S. M. *The Impact of Collaborative, Scaffolded Learning in K-12 Schools: A Meta-Analysis.* Cisco Systems, Inc., 2009.

[113] XU, J., ZHANG, J., HARVEY, T., AND YOUNG, J. A Survey of Asynchronous Collaboration Tools. *Information Technology Journal 7* (2008), 167–171.

[114] YANG, M., AND YANG, Y. An efficient hybrid peer-to-peer system for distributed data sharing. *IEEE Transactions on Computers 59*, 9 (2010), 1158–1171.

[115] ZHANG, E. OMG DataChannels and how to use them. [online] Available at: `http://www.slideshare.net/zhange1/why-webrtc-datachannel-excites-me`. Accessed May 25, 2013.

[116] ZHANG, L. A retrospective view of network address translation. *Network, IEEE 22*, 5 (2008), 8–12.

Appendix A

Tables of Performance evaluation

Size of file (in kB)	WebRTC Chrome (in Mbps)	WebRTC Firefox (in Mbps)	Google Chat (in Mbps)	Skype (in Mbps)
10	0.0122	0.1436	0.0172	0.1333
100	0.0131	0.9709	0.1826	0.9639
500	0.0117	4.0692	0.9011	5.1948
1000	0.0117	6.9626	1.6532	7.619
5000		26.2985	5.787	29.1971
10000		29.9065	7.5422	36.3636

Table A.1: Data of peer-to-peer applications

Size of file (in kB)	Facebook (in %)	Dropbox (in %)	Bistri (in %)	Cisco Webex (in %)
10	0.6607	0.0727	0.4473	
100	2.1015	0.7123	3.4547	
500	5.3197	2.3195	15.341	
1000	7.538	5.1072	31.4286	
5000	11.1842	22.4473	76.2958	
10000	10.9696	25.7379	100.9073	

Table A.2: For Client-server upload

Size of file (in kB)	Facebook (in %)	Dropbox (in %)	Bistri (in %)	Cisco Webex (in %)
10	0.2842	0.3257	0.1492	0.112
100	2.5846	3.745	2.19	1.0345
500	16.47	17.8926	11.8643	4.7826
1000	24.3267	26.7693	15.3266	6.9247
5000	96.4209	109.1317	21.3233	32.4166
10000	128.355	160.0044	25.5641	46.5908

Table A.3: For Client-server download

Size of file (in kB)	Facebook (in %)	Dropbox (in %)	Bistri (in %)	Cisco Webex (in %)
10	0.9448	0.3985	0.5966	0.112
100	4.6861	4.4574	5.6448	1.0345
500	21.7898	20.2121	27.2053	4.7826
1000	31.8647	31.8766	46.7552	6.9247
5000	107.6053	131.5791	97.6191	32.4166
10000	139.3248	185.7425	126.4715	46.5908

Table A.4: For Client-server Total

Appendix B

Comparision data of p2p and client server

Size of file (in kB)	Google Talk (in Mbps)	Dropbox (in Mbps)
10	0.0029	0.0018
100	0.0266	0.0158
500	0.1129	0.067
1000	0.2063	0.1203
5000	0.4269	0.4635
10000	0.4043	0.6103

Table B.1: Critical analysis of peer-to-peer based Google Talk and client server based Dropbox applications

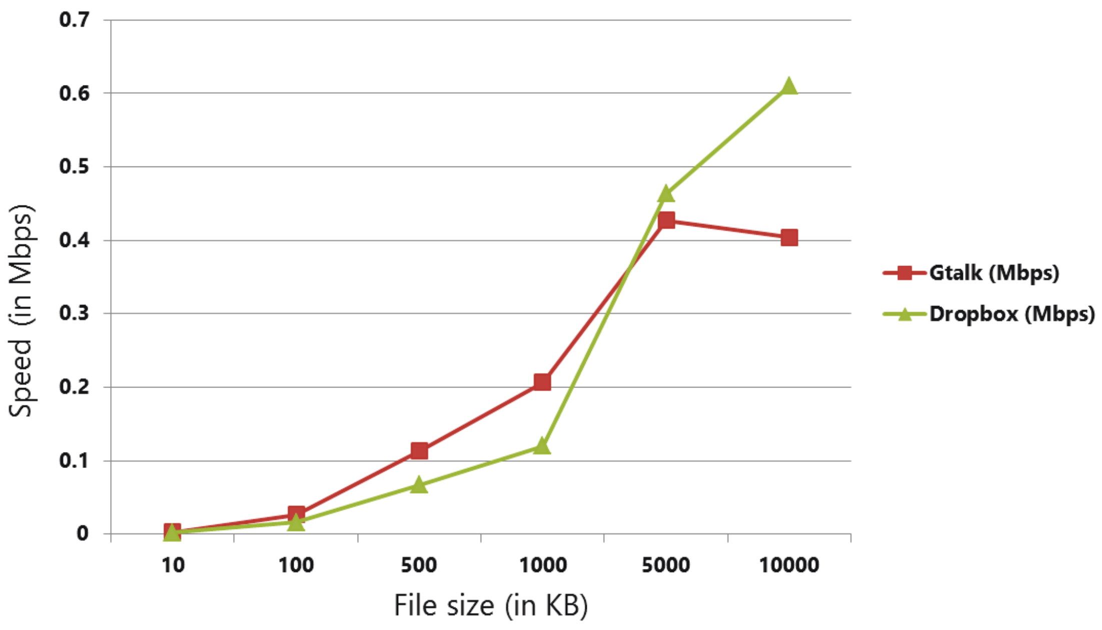

Figure B.1: Critical analysis of peer-to-peer based Google Talk and client server based Dropbox applications

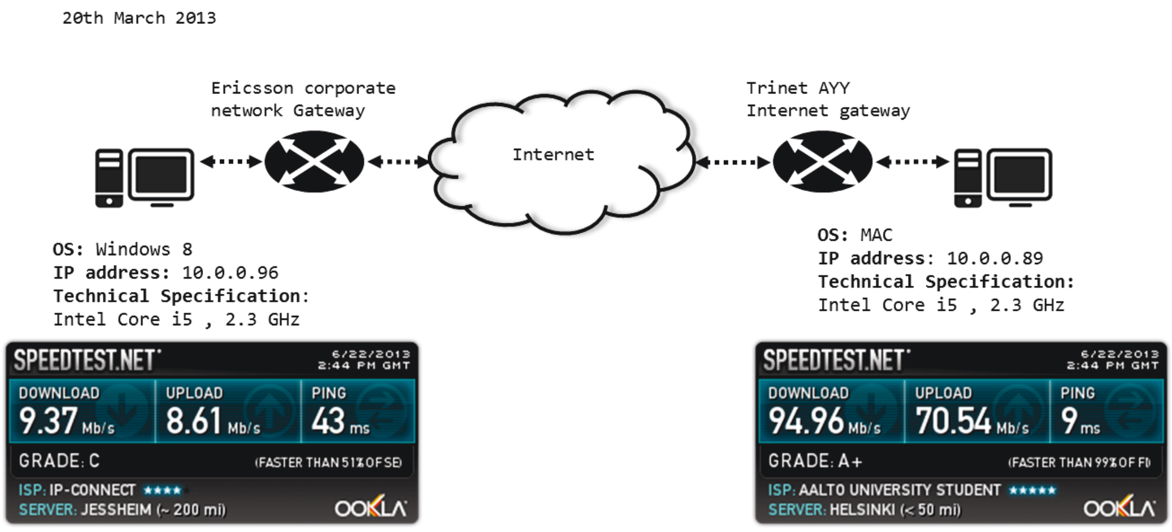

Figure B.2: Network Topology

Appendix C

Chrome browser SDP

```
Created RTCPeerConnnection with config:
  "{"iceServers":[{"url":"stun:stun.l.google.com:19302"}]}".
--------offer
v=0
o=- 4164328662 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio video data
a=msid-semantic: WMS
m=audio 1 RTP/SAVPF 103 104 111 0 8 107 106 105 13 126
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:TQSVVH49cRHbuhRx
a=ice-pwd:GmCi0WmSRSYkCHD8LUipWY5F
a=ice-options:google-ice
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=recvonly
a=mid:audio
a=rtcp-mux
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:fuNJ7AbW0x44QHaNzXPfBDV389lRnkIf84ElYI40
a=rtpmap:103 ISAC/16000
a=rtpmap:104 ISAC/32000
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:107 CN/48000
a=rtpmap:106 CN/32000
a=rtpmap:105 CN/16000
a=rtpmap:13 CN/8000
```

```
a=rtpmap:126 telephone-event/8000
a=maxptime:60
m=video 1 RTP/SAVPF 100 116 117
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:TQSVVH49cRHbuhRx
a=ice-pwd:GmCi0WmSRSYkCHD8LUipWY5F
a=ice-options:google-ice
a=extmap:2 urn:ietf:params:rtp-hdrext:toffset
a=recvonly
a=mid:video
a=rtcp-mux
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:fuNJ7AbW0x44QHaNzXPfBDV389lRnkIf84ElYI40
a=rtpmap:100 VP8/90000
a=rtpmap:116 red/90000
a=rtpmap:117 ulpfec/90000
m=application 1 RTP/SAVPF} 101
c=IN IP4 0.0.0.0
a=rtcp:1 IN IP4 0.0.0.0
a=ice-ufrag:TQSVVH49cRHbuhRx
a=ice-pwd:GmCi0WmSRSYkCHD8LUipWY5F
a=ice-options:google-ice
a=sendrecv
a=mid:data
b=AS:30
a=rtcp-mux
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:fuNJ7AbW0x44QHaNzXPfBDV389lRnkIf84ElYI40
a=rtpmap:101 google-data/90000
a=ssrc:3669356944 cname:XlaXnuhatB97wuli
a=ssrc:3669356944 msid:RTCDataChannel RTCDataChannel
a=ssrc:3669356944 mslabel:RTCDataChannel
a=ssrc:3669356944 label:RTCDataChannel
```

Appendix D

Firefox browser SDP

[14:00:26.386] ———offer sdp provided by offerer

```
[14:00:26.288] Created RTCPeerConnnection with config:
  "{"iceServers":[{"url":"stun:23.21.150.121"}]}".
[14:00:26.386] v=0
o=Mozilla-SIPUA 8317 0 IN IP4 0.0.0.0
s=SIP Call
t=0 0
a=ice-ufrag:e026ce37
a=ice-pwd:13e04c2a13d50b57cac358296a9bb1aa
a=fingerprint:sha-256 07:2D:B7:3A:1A:D4:70:67:75:3D:A4:35:3D:6C:4A:64
      :8D:05:07:7A:C8:F3:C9:62:21:74:88:40:42:51:15:E4
m=audio 54781 RTP/SAVPF 109 0 8 101
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:2rCp57flgAlhBx5i7lvgeh5wko87zkykgi3ivls7
c=IN IP4 193.234.218.122
a=rtpmap:109 opus/48000/2
a=ptime:20
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
a=candidate:0 1 UDP 2113667327 10.0.0.98 53034
  typ host
a=candidate:1 1 UDP 1694274815 193.234.218.122
  54781 typ srflx raddr 10.0.0.98 rport 53034
a=candidate:0 2 UDP 2113667326 10.0.0.98 57083
  typ host
a=candidate:1 2 UDP 1694274814 193.234.218.122
  63786 typ srflx raddr 10.0.0.98 rport 57083
```

```
m=video 65065 RTP/SAVPF 120
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:2rCp57flgAlhBx5i7lvgeh5wko87zkykgi3ivls7
c=IN IP4 193.234.218.122
a=rtpmap:120 VP8/90000
a=recvonly
a=candidate:0 1 UDP 2113667327 10.0.0.98 50785
  typ host
a=candidate:1 1 UDP 1694274815 193.234.218.122
  65065 typ srflx raddr 10.0.0.98 rport 50785
a=candidate:0 2 UDP 2113667326 10.0.0.98 60117
  typ host
a=candidate:1 2 UDP 1694274814 193.234.218.122
  58295 typ srflx raddr 10.0.0.98 rport 60117
m=application 57330 SCTP/DTLS 5000
a=crypto:1 AES_CM_128_HMAC_SHA1_80
  inline:2rCp57flgAlhBx5i7lvgeh5wko87zkykgi3ivls7
c=IN IP4 193.234.218.122
a=fmtp:5000 protocol=webrtc-datachannel;streams=16
a=sendrecv
a=candidate:0 1 UDP 2113667327 10.0.0.98 51694 typ host
a=candidate:1 1 UDP 1694274815 193.234.218.122
  57330 typ srflx raddr 10.0.0.98 rport 51694
a=candidate:0 2 UDP 2113667326 10.0.0.98 60395 typ host
a=candidate:1 2 UDP 1694274814 193.234.218.122
  50035 typ srflx raddr 10.0.0.98 rport 60395
```