TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SHAHZAD HUSSAIN

WEB BASED NETWORK SPEECH RECOGNITION

Master of Science Thesis

Examiners: Prof. Irek Defee, Prof. Jarmo Harju
Examiners and Topic approved in the
Faculty Council Meeting on
07.11.2012

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
SHAHZAD HUSSAIN: Web based Network Speech Recognition
Master of Science Thesis, 38 pages
January 2013
Major: Communication Eningeering
Examiner: Prof. Irek Defee, Prof. Jarmo Harju
Keywords: Network Speech Recognition engine, CmuSphinx, Jetty, HTTP Servlets, WebSockets, HTML5, doGet, doPost, Java API, ASR

In modern technological world there is a continuous flow of improvements, as it has been with computers, tablets and smart phones. There is a need to explore alternatives for effectively using those new devices and systems. Effective usage of those devices may include: virtual keyboards rather than physical ones, the use of a touch screen instead of a mouse, using location based services to find events that are happening around us instead of accessing a Website in order to get the information, talking to a smart phone or tablet in order to call, text or do tasks, thus replacing the physical interaction via a touchscreen or keyboard. The main contribution of this thesis is to design and implement a Web-based Network Speech Recognition system using Open Source components and new emerging technologies. This system can take audio queries from a Web browser, feeds them into the Speech engine and returns the result back to the Web browser client. Web-based Network Speech Recognition systems already have been built by Google, Nuance and many other companiess. Implementation however differs in various ways, such as the use of WebSockets in real time or the use of HTTP Request / Response method. The system developed in the theis is entirely composed of open source elements: the speech recognition engine that serves the speech recognition requests, and a Web Server to receive the audio stream from the Web browser clients. The designed system efficiency is high and it can serve multiple clients and it provides good processing power making it able to manage heavy load operations with reasonable effort.

PREFACE

I would like to thank my parents for giving me high moral values through out my life; this has helped me in my every life endeavour and has taught me the wonders of life.

I would like to express my deep gratitude to Prof. Irek Defee, and coordinator Elina Orava for supporting me during my academic career. I would also like to thank Prof. Jarmo Harju for assessing my thesis and providing comments.

I am also very thankful to Ericsson and part of the Ericsson Team such as Minna Hallikainen (section Manager) Marko Seikola (section Manager), Jouni Mäenpää (Nomadic Lab section Manager) and Jaime Jiménez (thesis supervisor) for providing me resources, information and help whenever I needed it.

Shahzad Hussain
January 2013

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TERMS AND DEFINITIONS

HTML	HTML (Hyper Text Markup Language) is the markup language that displays Web contents in the Web browser.
WebRTC	WebRTC (Web Real Time Communication) is the new emerging standard for the real time communication between Web browsers.
Webrtc.org	It's a free Open Source project supported by Google, Opera and Mozilla. It enables rich media real time communication application development for the browser using HTML5 and JavaScript.
BLOB	BLOb (Binary Large Object) is an object containing binary data. BLObs are usually images, audio and video obejcts.
HTML5	HTML5 (Hyper Text Markup Language 5) is the newest standard and the fifth revision of HTML. HTML5 has brought the possibility of using rich media content without using third party plugins.
WebSocket	WebSocket is the HTML5 feature for creating a socket connection between Web browsers or Web browser to server. It gives the possibility to have a real time communication.
Servlet/Web Application	A Servlet or Web Application is a Java class deployed by the Webserver in order to extend its capabilities.
Binary WebSocket	A binary WebSocket enables user to send binary data to the remotely connected host.
Jetty	Jetty is an HTTP Webserver, HTTP client and a Servlet container provided by Jetty.
ASR	ASR (Automatic Speech Recognition) enables user to input Speech to be recognized.
W3C	W3C (World Wide Web Consortium) is an international community for developing open Web standards.
HTTP Request	HTTP Request is the request made by Web browser to the Webserver for certain content or information. It may include a query to be processed.
HTTP Response	HTTP Response is the response replied by the Webserver with the processed HTTP Request result.
HTTP Body	HTTP Body is the body of HTTP Request header. It usually includes data such as text, image or audio etc.
Live Web Audio	It's the HTML5 feature for capturing audio from microphone in a Web browser.
SphinxServer	SphinxServer is a Java class responsible for instantiating Jetty Webserver instance. It listens for incoming WebSocket con-

	nections. It registers all its connected WebSocket clients.
AudioCollector	It’s a Java Servlet responsible for collecting audio from the WebSocket clients. It also transcodes audio to the required format.
SphinxRecognizer	SphinxRecognizer is a Java class responsible for setting up Recognition engine and its sub components.
Recognition engine	Recognition engine is the main core of recognizing Speech. It processes audio request and provides recognized result.
Recognizer	Recognizer is a Java class provided by Recognition engine for executing Speech Recognition.

1. INTRODUCTION

In the modern technological world, information processing tasks are shifting from desktop computers and laptops to small handheld devices such as smart phones, tablets and other small gadgets for retrieving information on the go. Tremendous amount of energy has been placed on bringing the world closer by creating smart applications that provide information just by pressing few buttons or tapping the touch screen.

Speech recognition and Text-to-Speech technology is currently being used in many systems such as handheld devices, infotainment system in cars, etc [1]. Their purpose is to facilitate interacting with these systems. For example: A person is lost while driving somewhere in a remote town in Finland, not knowing how to return to the capital Helsinki. This person can tell the infotainment system to find a route for him by commanding it: “Navigate to Helsinki”. The infotainment system will give the detailed route via voice instructions produced by speech synthesis i.e. Text-to-Speech. In the same way when someone wants to use phone to call, text, or find a route while driving, that person can talk to his mobile phone and the system will give direction to the destination.

Speech recognition technologies use a speech recognition engines. These engines are usually fed with a grammar, which is a set of words/sentence created with certain rules. The engine recognizes speech by matching the input to the grammar.

There are many companies like Nuance or Voxeolabs which provide speech recognition engines. There is also an open source version available called CmuSphinx [2]. These engines need more processing power and posses a very limited grammar set of few thousand words. Moreover, such engines are not capable of recognising a large percentage of the input due to the grammar limitation.

In this thesis, a Network Speech Recognition engine with efficient throughput is provided. Speech recognition computation is done remotely, thus saving processing power on handheld devices. Recorded speech is sent to the remote server via the Internet.

The principal Web technology usually used in network speech recognition is the HTTP Request / Response [3] mechanism where the audio is usually sent in chunks in HTTP Request message body. The result is received in the body of HTTP Response.

Preliminary research has been done trying two different approaches to the problem: HTTP Request / Response and WebSockets. The HTTP Request/Response uses HTTP Request to post audio stream to the network speech recognition server in order to process the audio stream. The other alternative uses WebSockets [4] to stream the captured audio data from the microphone to the network speech recognition server, where the input speech is processed and the results are returned back to the client.

The aim of this thesis is to build a more efficient and real time Network Speech Recognition system using WebSockets. The implementation of this thesis focuses only on network speech recognition using WebSockets, however comparison and test results

using HTTP Request / Response as well as WebSockets are mentioned in the later chapter. Following technologies are used in the implemention of our Network Speech Recognition;

- CmuSphinx, an open source speech recognition engine
- HTML5 Live Web Audio [5] feature to record audio.
- Jetty WebServer to host Web application to receive audio stream for post processing.
- WebSocket to send/receive plain text messages or binary audio stream.

2. BACKGROUND

Speech recognition, network based speech recognition, Web-based network speech recognition and the technologies that are used to enable network speech recognition across network have been discussed in this chapter. Research on this topic has been done in collaboration with an Ericsson team and the open source community.

2.1 Speech Recognition

Speech Recognition enables a user's speech and spoken words to be translated into text. Speech Recognition uses a set of complex modules and sub-systems to recognize input audio Speech. It's also known as ASR (Automatic Speech Recognition) or STT (Speech to Text).

There are many third party speech recognition systems which use various audio formats for efficient speech recognition. To deal with this situation usually encoders/decoders are placed before the speech recognition system to provide it with usable audio format stream.

Speech recognition engine uses grammar or language models for recognizing speech. A grammar is a set of word pattern or small sentences built with sets of rules, provided to speech recognition system to tell it the expectation of meaning of spoken words. Speech recognition systems usually allows user to input grammar with certain rules.

A group of actions can be performed upon recognized speech results based on the design of the relevant application. For example a user inputs search queries in Google's search box field using Google's speech recognition system.

Speech recognition is not only about transcribing but can be extended to the needs. Some of the important applications of Speech Recognition are in many areas such as:

- Aerospace (space exploration, spacecraft: Mars Polar Lander used Speech Recognition from Sensory, Inc)
- Automatic translation
- Automotive speech recognition
- Court reporting (speech writing)
- Hands-free computing
- Home automation

- Health Care etc.

2.1.1 Speech Recognition Components

- Open Source Speech Recognition Engine

CmuSphinx (also known as Sphinx) is an open source speech recognition toolkit developed at Carnegie Mellon University [2]. It includes a series of Speech recognizers such as Sphinx, Sphinx 2, Sphinx 3 and Sphinx4. It has an acoustic-model-trainer also known as SphinxTrain.

Acoustic model [6] is created by the input speech recordings and their transcription. Certain tools are used to create a statistical representation of sounds that are mapped with the words. Sphinx in general is composed of the following libraries and tools:

- Pocketsphinx: lightweight recognizer library written in C.
- Sphinxbase: support library required by Pocketsphinx
- Sphinx4: adjustable, modifiable recognizer written in Java
- CMUclmtk: language model tools
- Sphinxtrain: acoustic model training tools
- Sphinx3: decoder for speech recognition research written in C.

- Grammar

The grammar describes the language based on rules of the type command and control. They can be created manually or can be software generated.
Here we have an example of writing a grammar of phone contacts with command control:
Command: John, Bil, Ralph, Lisa, Einstein
Control: Call, Hangup, Text, Sms, Send MMS, email to, to, Send
Rule: Command + Control
So the final grammar can be: Call John, Call Bil, Hangup John, Text Lisa, Send MMS to Einstein, Send Sms to Ralph, Send Text to John, Send Text John etc.

- JSGF (Java Speech Grammar Format)

CmuSphinx supports JSGF based grammar [7] . It follows Java programming language conventions developed by Sun Microsystems. CmuSphinx JSGF based grammar example is following:

JSGF Grammar Basic Example
#JSGF V1.0; grammar phonecontrol;

grammar <phonecontrol> = <command> <object> <optional> <entity> <command> = (Send | Open | Close | Please) <object> = (Text | Sms | Email | Call) <optional> = [to | of | for] <entity> = (John | Bil | John | Lisa | Einstein)
Resulting Grammar: Send Text to John Open Sms of Lisa Please Call Bil etc.

- Statistical Language Model

The Statistical Language Model [8] provides probability distribution P(s) over strings S, which tries to define how frequently a string S occurs in a sentence. They are used in many applications such as natural language processing, speech recognition, machine translation, Speech tagging, parsing and information retrieval. CmuSphinx provides CMUSLM (CmuSphinx Statisical Language Modelling) toolkit for building Statistical Language Model.

- Third Party Speech Recognition engines

Speech recognition engines are developed by many companies around the globe. Most of those companies have developed their own engines for their own products rather than to sell to third parties. Some of the high calibre ASR engines including commercial and open source are Nuance, iSpeech, Lumenvox, Voxsigma, CmuSphinx.

This thesis main goal is to enable network speech recognition, not the implementation of recognition engine itself. A general overview of CmuSphinx will be presented in this thesis. The details of its integration into the network will be discussed later in the following chapters. However the detailed description of CmuSphinx is out of this thesis scope. The Sphinx4 recognizer [9] has been used in this thesis project.

The recognition engine relies on the grammar and the statistical language model in order to successfully recognize the input speech pattern. The statistical language model is out of the scope of this thesis and only JSGF based grammar has been used in research.

2.2 Network Speech Recognition

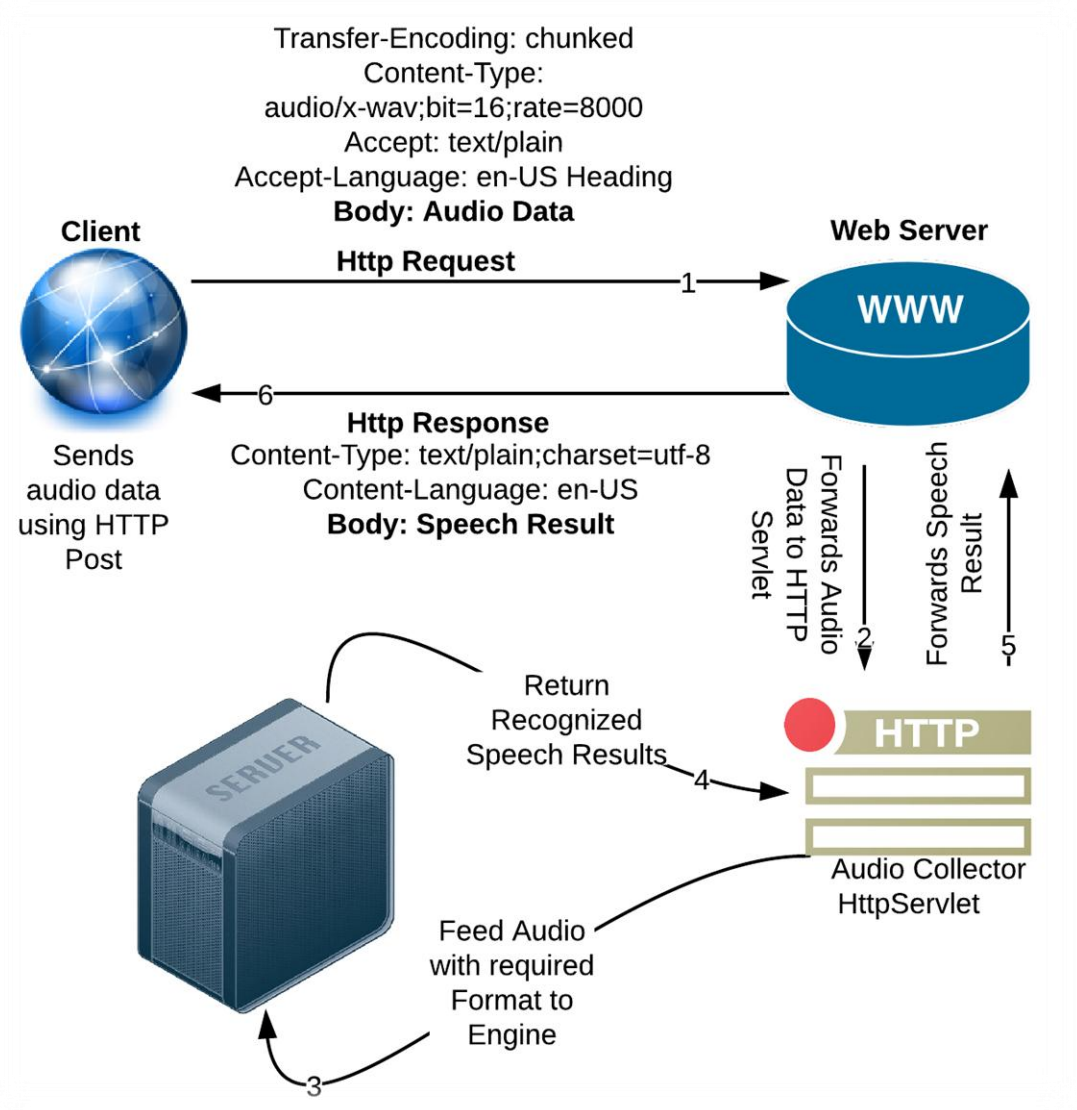

Figure 2.1: Network Speech Recognition Architecture using HTTP Request / Response

Figure 2.1 shows the Network Speech Recognition system using HTTP Request / Response system. The Network Speech Recognition system provides a more efficient way for more accurate speech recognition results. It includes a recognition engine, sitting on top of a very high calibre processing computer, having intense collection of grammar, to provide more accurate results than the local recognizing engine in a phone or any electronic device which has the speech recognition capabilities and has limited grammar and outputting less accurate results.

The Network Speech Recognition engine has the capability to serve multiple users at the same time while still providing a robust output. It can work in a LAN, WAN / MAN enviornment depending on its network implementation design.

Many companies use the above architecture to serve speech recognition requests. Before giving a detailed explaination of how the above network speech recognition works, some of the terms in the aforementioned figure need to be explained.

The client side can be implemented in any language such as Java, Ruby, Python, Perl, C/C++, etc. The client side might also record audio using the microphone or send an audio stream over the network to the server by reading recorded audio from file. It uses HTTP Request / Response mechanism to send audio and to receive Speech results.

HTTP Post Request is send to the Webserver with specific set of HTTP headers and a body message. It may include the following headers or additional headers depending on the requirements of third party companies who are on the receiving side of the audio Speech Recognition requests. HTTP Header contains the following entities;

- `Content-Type`
- `Content-Language`
- `Accept-Language`
- `Accept`
- `Accept-Topic`
- `Transfer-Encoding`

A typical example of HTTP Headers can be:

```
Content-Type: audio/x-wav;codec=pcm;bit=16;rate=16000
Content-Language: en_US
Accept-Language: en_US
Accept: text/plain
Accept-Topic: en_US
Transfer-Encoding: chunked
```

The WebServer receives user's requests as HTTP Requests and after post processing sends the result back as HTTP Response. HTTP Servlets [10] are Web applications that are deployed by the Web server and used for special purposes. In the above figure when the audio is collected from the client it is then forwarded to the HTTP Servlet "Audio-Collector" where post processing may be done to ensure the input stream fits the recognition engine and after which it is fed into the engine for recognition processing.

The Recognition Server is the main part of the whole system which takes an audio input stream, processes it for recognition and returns the result back in plain text. It has a versatile list of grammar for enhanced throughput.

A client records audio from the microphone or reads an input stream from an audio file, sets up HTTP request with audio stream as a body message and sends it to the remote Web server hosting speech recognition engine. The stream is received in the Web server where it is forwarded to HTTP Servlet "AudioCollector" for post processing (maybe) and then fed into the recognition engine. The input speech is processed and the results are provided back to the HTTP Servlet. HTTP Servlet sends the result back by

creating a custom HTTP Response with the speech result as plain/text body message. Client receives the result from the Web server as HTTP Response.

2.2.1 Network Speech Recognition Components

- Browser and Webserver Concept

In client server terminology a browser is a client application used for retrieving, traversing, presenting information resources, acquiring files from remote over plain or secure connection with HTTP or HTTPS protocol. Information resources can be images, text, Web pages, files, etc. There are many Web browsers available on Internet. The most popular ones are Mozilla Firefox, Chrome, Internet Explorer, Safari, Opera, etc.

A Web server on the other hand serves browsers requests over the network. Web server holds information of contents deployed to it. It may contain Web pages, pictures, text contents or HTTP Servlets to do specific tasks. As mentioned earlier it gets a request from Web browser, processes it and sends the response back to the browser. There are many Webservers present on the Internet. Most popular ones are Apache, IIS and Jetty [11] etc.

A resource is identified by URI (uniform resource identifier) and the mechanism used in establishing connection with remote server and presenting information sources is done on HTTP Request / Response Mechanism.

HTTP Request is a request made by Web browser to the Web server for certain information resources and HTTP Response is the response by Webserver providing the Web browser with resource information (if available) pointed by URI when it was making an HTTP Request. HTTP Request contains relevant headers and body message to be sent to Webserver. HTTP Response has acknowledgment headers with a body message to be sent back to the Web browser on successful processing of requests. An HTTP requests may contain request for some information resources which may not be present at the specific URI in which case the Webserver sends an HTTP Error response stating the Error type.

Typical Example of HTTP Request Header:

```
GET HTTP://www.example.com/ HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/5.0 [en] (X11; I; Linux 2.2.3 i686)
Host: www.example.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, im-
age/png, */
*

Accept-Encoding: gzip
Accept-Language: en
```

Typical Example of its HTTP Response Header:

```
HTTP/1.0 200 OK
Date: Fri, 13 Nov 2009 06:57:43 GMT
Content-Location: http://www.example.com/index.html
Etag: "07db14afa76be1:1074"
Last-Modified: Mon, 05 Nov 2012 20:01:38 GMT
Content-Length: 7931
Content-Type: text/html
Server: Microsoft-IIS/4.0
Age: 922
Proxy-Connection: close
```

- Web server and Get/Post Requests

Webserver serves requests for the user clients in HTTP Request / Response mechanism. It uses a Web root directory for the Web pages to be placed from where the contents are usually served. It also provides a way of putting Web contents outside Web root directory known as virtual-directories where the contents can be placed other than Web root directory.

A Web browser can send HTTP request in the form of Get and Post request. A Get request contains a URI to specific content information which upon processing by Web server is shown to the user on the Web browser where as in Post request the information is send to a Web server for further processing.

If one opens a Web site, for example www.example.com Webpage, the request made by the Web browser is a HTTP Get request. A Web page containing a form to be filled and submitted to the Web server is an HTTP Post request.

- HTTP Servlets / Web Applications

An HTTP Servlet or Web Application is a server-side Web technology based on Java. It is a Java class, coded in Java EE based on Java-Servlet-API [12]. HTTP Servlet can respond to any type of requests but usually they are used with HTTP protocol. Dynamic Web contents can be deployed to Webserver using Servlets [11]. Web container is used to deploy a Servlet [13].

A Servlet overrides two methods doGet() and doPost() while inheriting from HTTP servlet Class in Java using . Both of these methods used in the Servlet are used for interacting with HTTP Get and HTTP Post requests generated by the user in the Web browser.

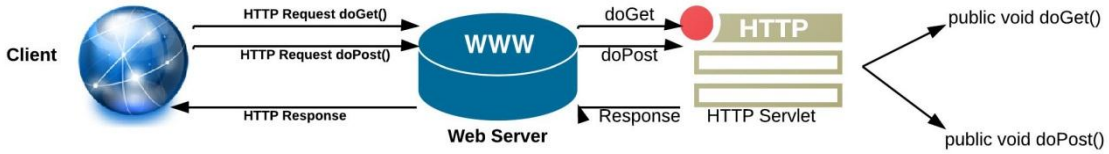

Figure 2.2: HTTP Request / Response & doGet, doPost Interpretation to Servlet

Figure 2.2 shows the HTTP Request interpretation at the HTTP Servlet. The HTTP Get and Post Request are received at the HTTP Servlet in *doGet()* and *doPost()* methods respectively. Following is a sample code of a very basic HTTP Servlet:

Basic HTTP Servlet

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;

public class HelloWorld extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\"
\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">");
    out.println("<html xmlns=\"http://www.w3.org/1999/xhtml\" xml:lang=\"en\" lang=\"en\">");
    out.println("<head><title>Hello World</title></head>");
    out.println("<body>");
    out.println("<h1>Hello World</h1>");
    out.println("</body></html>");
  }

 public void doPost(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {

 }
}
```

- Webserver Instantiation using API

Web servers can be run using their provided API in addition to its binary provided in a package. Configurations can be coded for configuring connectors, handlers, Servlets, etc.

For example a very basic Web server instance without any configuration can be created using Web server API as follows:

```
WebServer server = new WebServer (8080);
server.run ();
```

Assuming Web server is the class provided by its API we are creating an instance of Server with 8080 as a port parameter. We then run it using server.run () method. Web server can be configured through their API like setting up connectors, handlers and Servlets [14].

2.3 Emerging Standards and Web Technologies

- HTML 5

Since 1999 when Hyper Text Mark-up Language (HTML) 4.01 came, the Web has changed a lot from displaying static contents to dynamic contents. The progress shift of the Web has brought new ways of presenting contents such as Flash or even live media contents from a streaming server using third party plugins. However there has been always a hassle to install certain plugins or applications, so that the rich media content could be presented properly on the Web. Such examples could be, for Java applets, one needs to have Java runtime and for Flash contents one needs to have Flash plugin installed in the machine. There are many other examples like watching a stream from Microsoft media streaming server, one needs to have windows media plugins in the browser to watch the contents.

To overcome such issue and with new features demand, a new standard HTML5 [15] [28] is developed. It provides access to multimedia resources natively inside the browser without any plugins or extensions. One of the exciting features of HTML5 is WebSocket, which provides communication between between server and client.

The HTML5 is developed with the focus on how the World Wide Web is going to evolve in the future. There has been tremendous amount of interest growing for Web applications that can communicate with each other. This has been taken in to account in HTML5 which defines API [16] to support such Web application development. HTML5 is backward compatible and still supports XML syntax.

HTML5 has introduced many new elements to the language such as canvas, audio, video for displaying webcam, playing audio and video respectively. The new elements have made it easy for the Web developers to embed audio and video contents.

The Live Web Audio feature [5] which provides access to microphone for capturing audio has been introduced in HTML5.

WebSocket [16] is the bi-directional communication protocol introduced in HTML5. It allows communication between server and client on a full duplex TCP channel. Bi-directional communication here means that the server and client can establish connection, send and receive data on the same channel at will, thus allowing the server and client to transmit data on need basis.

Currently HTML5 is not official standard yet and not all the browser supports HTML5 features. However there are number of browsers who have HTML5 support. They include Firefox, chrome, Safari, Opera and Internet Explorer.

- Web Real Time Communication (WebRTC)

WebRTC [30] is an emerging standard of HTML5 for real-time communication between Web browsers. The open source project Webrtc.org [17] is supported by Google, Mozilla and Opera. It enables developer to create rich content, high quality RTC (Real Time Communication) applications using JavaScript API and HTML5. Some examples of the RTC application are P2P file sharing, Video Chat, Audio Streaming, Video Conferencing, etc. Its API is drafted by World Wide Web Consortium (W3C) [18].

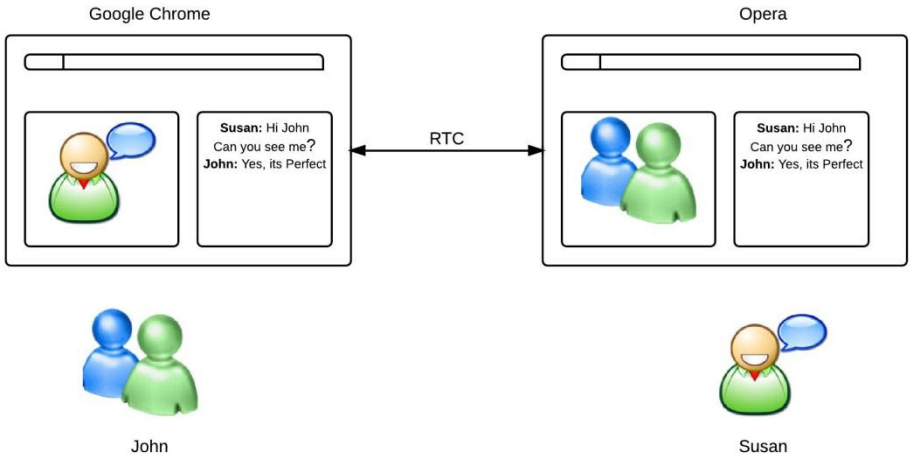

Figure 2.3: WebRTC using Web Browser

Figure 2.3 shows a general scenario of WebRTC based chat application in the Web browsers.

With the provided easy API, it's now very convenient to access microphone, webcam and stream the audio and video in real time to the client. Many audio visual effects can be applied to the stream according to the needs and then finally send it in real time to the client browser.

Code snippet below shows how to access microphone using JavaScript.

```
Navigator.WebkitGetUserMedia({audio:true, video:true}, function(stream){
// stream is the stream while capturing from mic and Webcam.
// Do rest of stuff here
});
```

- WebSockets

Sockets are a way to communicate between applications or different servers/workstations. In UNIX server / client programming, a socket using TCP connection is said to be a full duplex connection.

There had not been any sockets concept in Web technologies, until recently when full duplex communication channel over a single TCP connection has been implemented known as WebSockets. A WebSocket can be used between two Web browsers or by a Web application deployed by Webserver. By using full duplex communication channels over a single TCP connection it produces a real time data transfer [29]. Protocol used by WebSocket is an independent TCP-based protocol. The data transmitted between server and client can be either UTF-8 text, binary frames or special control frames which are used for connection handling. The handshake mechanism follows the routine:

- WebSocket sends a handshake Request to the server.
- Server interprets the handshake Request as an Upgrade request.
- Server sends back handshake Response to the WebSocket client.

To initiate a WebSocket connection, a URI for the remote server is required. URI is identified by "ws" (Web Socket) following by a port number. An example of URI to connect can be; ws://somehost.com:portnumber (replace port number with the port number of your choice).

Let's assume we have request URL ws://somehost.com:9999/ and our Request Method is GET, the handshake mechanism is then as follows:

WebSocket to Server Handshake Request
Get ws://somehost.com:9999 HTTP/1.1 Connection: Upgrade Host: localhost:8080 Origin: http://localhost:8080 Sec-WebSocket-Key1: xxx xxx xxx Sec-WebSocket-Key2:xxx xxx xxx Upgrade: WebSocket (Key3): XX:XX:XX:XX:XX:XX:XX:XX

Server to WebSocket Handshake Response
Connection: Upgrade Sec-WebSocket-Location: ws://somehost.com:9999/ Sec-WebSocket-Origin: http://localhost:8080 Upgrade: WebSocket (Challenge Response): XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX

WebSocket protocol has been standardized as RFC 6455 by the Internet Engineering Task Force (IETF) and its API [19] is being standardized by the World Wide Web Consortium (W3C). Following is an example of WebSocket client opening a connection to a server:

WebSocket JavaScript Example

```
<script type="text/JavaScript">
var wsURL = ws://someipaddressorhost:7777;
var wsocket = new WebSocket(wsURL);

// When Web socket is opened, this handler is called
wsocket.onopen = function(evt) {
                    // do something here
                    // send plain text or binary data
                    wsocket.send("send plain/text or binary data as blobs[1]");
}

// When we socket is closed, this handler is called
wsocket.onclose = function(evt) {
                    // do something here
}

// when incoming data arrives; plain/text or binary
// this handler is called
wsocket.onmessage = function(evt) {
                    // do something here
}

// If error occurs, this handler will be called
wsocket.onerror = function(evt) {
                    // do something here
}

// if socket is closed, then this handler is called
wsocket.onclose = function(evt) {
                    // do something here
}</script>
```

Data that can be sent via WebSockets are plain/text and binary data in the form of blobs [20]. WebSocket has been implemented in Firefox, IE, Opera, Safari and Google Chrome. It is required that the application hosted by server also supports WebSocket.

3. SYSTEM IMPLEMENTATION

This chapter describes the implementation of a Web-based Network Speech Recognition system. The general architecture, detailed architecture and system flow are presented in the following diagrams of this chapter.

While designing our system we came to a crossroad when choosing the Network Speech Recognition systems, we had to choose between one of the available alternatives. One of the alternatives were using WebSockets with Network Speech Recognition and the other HTTP Request / Response mechanism by embedding audio in the HTTP Post message. In the following table we present the main advantages and drawbacks of the two alternatives.

	WebSocket based Network Speech Recognition	**HTTP Request / Response Based Network Speech Recognition**
Handshake	Uses standard WebSocket connection to the server using "Upgrade request" in HTTP Header during handshake.	Uses standard HTTP request headers.
Real time live Content	WebSocket connection to the server provides real time contents transmission.	Doesn't provide real time contents transmission.
Transmission mechanism	Full-duplex communication channels over a single TCP connection.	Not full-duplex.
Audio Transmission	Audio transmitted over socket to socket. (Audio received as byte stream without any headers)	Audio transmitted in HTTP Post body message is received via HTTP request header object in the server side.
Low overhead	Data can be transmitted as like on normal sockets i.e. no headers overhead.	Headers overhead.
Live Web Audio	Audio can be recorded by standardized HTML 5 Live Web Audio input feature.	Not possible (certain non-standard mechanism by third parties like embedding FLASH applet to record audio present)
Audio sending to server	WebSocket is created, audio is sent over it as in normal socket.	a. HTTP client is created. b. HTTP Post is created. c. Header is added to HTTP Post. d. Audio is embedded in HTTP Post.
Audio receiving at the server	Audio is received as byte stream on client side.	a. Audio is received in HTTP Request header object. b. Audio is stored in Byte stream by reading HTTP Request header body.
Audio Forwarding	Audio is fed into the Recognition engine.	Audio is fed into the Recognition engine.
Speech Recognition	Speech recognized returns result as string array.	Speech recognized returns result as string array.
Returning Result	Return result to the client on WebSocket.	Return result to the client by embedding "result" in the body message of the header.
Result receiving	Results retrieved in real time as string array.	Results retrieved in the form of HTTP header body message.

Table 3.1: Comparison of WebSocket and HTTP Request / Response based Network Speech Recognition

As we can see that HTTP Request / Response mechanism lacks substantial features and has an overhead of headers. Therefore we came to conclusion that Web-based Network Speech Recognition engine using WebSocket was much more reliable, faster and efficient method than traditional HTTP Request / Response mechanism.

3.1 General Architecture

In this subsection we will elaborate on the general architecture of our system using WebSockets, as it is shown in figure 3.1.

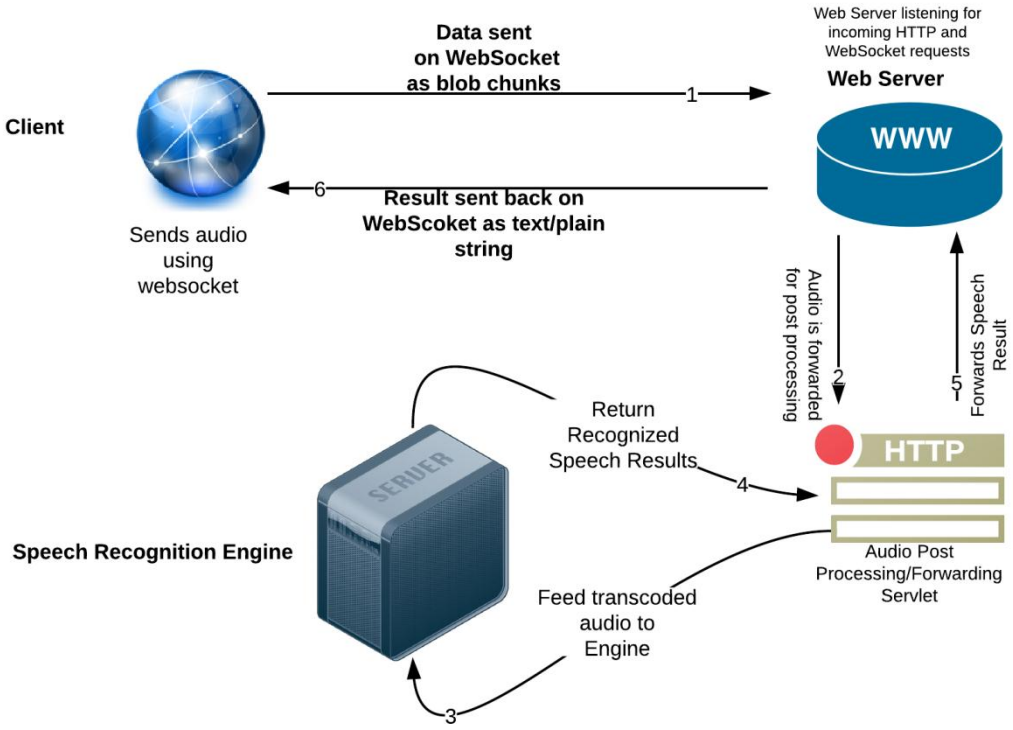

Figure 3.1: High Level Network Speech Recognition System Architecture using WebSocket

In the aforementioned figure 3.1, a user uses Google Canary Web Browser (in this case the custom development version) having the capability of recording audio from microphone using JavaScript.

Jetty is our Web server accepting incoming WebSocket connections; it forwards data between client and deployed Servlet.

We have AudioCollector HTTP Audio Post Processing / Forwarding Servlet deployed by Jetty. The main purpose of the AudioCollector is to post process audio data to the required format for the Speech Recognition engine. It also serves the Recognition engine with audio and returns recognized result back via WebSocket to the client.

Our speech recognition engine is using CmuSphinx for recognizing speech and returning recognized results back to the Servlet.

The client creates a WebSocket connection to the Jetty Webserver. The Google Canary browser can capture Live Web Audio from the microphone using simple JavaScript, this is a new feature enabled by HTML5. Audio is recorded in the form of blob chunks using JavaScript and is sent over the Web socket to the Jetty Webserver.

The AudioCollector listens for incoming audio data to be fetched. Audio blobs are fetched by the AudioCollector Servlet. Audio is post processed to the format required by the Speech Recognition engine. Once audio is transcoded to the required format, it is forwarded/fed into the CmuSphinx Recognition engine. CmuSphinx processes Speech and returns the result in the form of string array to the AudioCollector Servlet which is then send back to the client on the WebSocket connection.

3.2. System Architecture Overview

The system architecture of the Web-based Network Speech Recognition has been shown in figure 3.2., detailed client design flow is shown in figure 3.5 and the server design flow is detailed in figure 3.6.

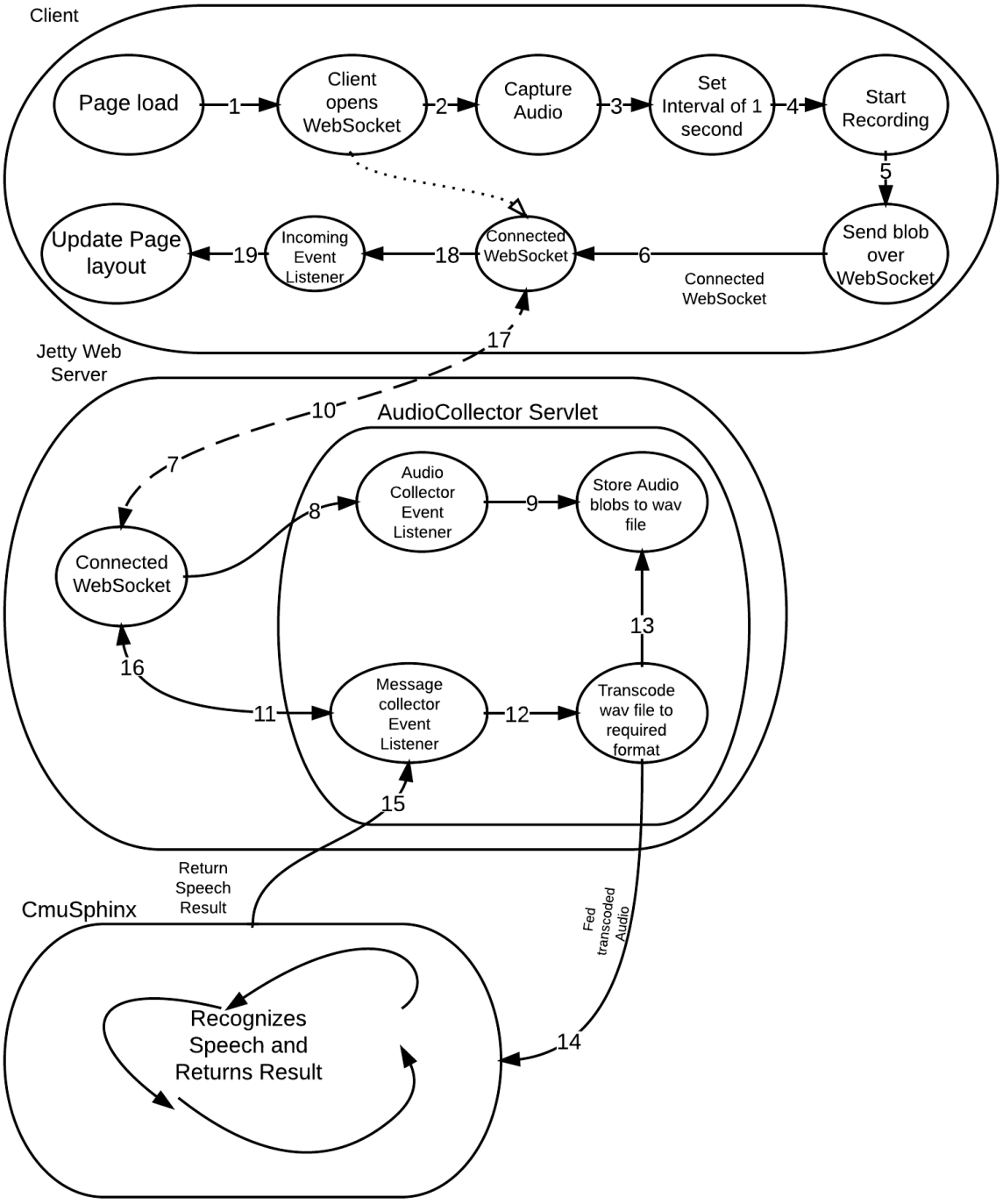

Figure 3.2: Component Level Architecture of Network Speech Recognition System using WebSocket

As can be seen from figure 3.2 the system architecture is divided into three parts. i.e. User Client, Jetty Webserver holding the AudioCollector Servlet and the CmuSphinx engine. Jetty Webserver serves client pages in addition to the AudioCollector Servlet.

The client part is responsible for creating the WebSocket connection to the server, setting up incoming event listener (for incoming messages via WebSocket), capturing audio from microphone and sending audio data as blobs with 1 second interval to the Jetty Webserver using WebSocket. It also updates the page layout with the real time information of speech recognition or any error information.

The Jetty Webserver deploys AudioCollector Servlet. It serves incoming request for client Webpages and also listens for incoming WebSocket connections. When a Web-Socket connection request is received, it registers the WebSocket connection. The deployed Servlet is the main core which does multiple functionalities. AudioCollector Servlet implements two event listeners named "onMessage" which has same names but different signatures i.e. *onMessage(byte[] stream, int offset, int length)* can be referred to as binary data event listener, which is used for collecting binary data such as audio blob in our case, *onMessage(String message)* is used for sending / receiving plain text messages. Audio is collected in the form of blobs every second on binary data event listener as mentioned above. The collected blobs are appended to a *wav* file until there are no more audio blobs (which is stopped by client). Recognition engines usually have a limitation of processing input audio with certain audio format, thus it is necessary to check collected audio at the server against the required audio format. For such purpose there is transcoding functionality to transcode the *wav* file containing audio blobs to the Recognition engine's required format. The AudioCollector Servlet is directly connected with Speech Recognition engine in order to provide it with transcoded audio Speech file and receive the processed Speech result.

CmuSphinx is an open source engine responsible for serving multiple user client speech recognition requests. It takes transcoded audio from AudioCollector, processes it and returns the result back to AudioCollector.

All communication including binary data transmission or plain text messages are done through WebSocket connection.

- Tools and Technologies

In the thesis implementation part the following tools and technologies are used:

- Server side technologies
  - Jetty Webserver, HTTP Servlets, WebSocket, Java language, CmuSphinx Speech Recognition server, Java Audio System [22]

- Server side tools
  - Maven2 [23], Eclipse, Java compiler

- Client side technologies
  - HTML 5, JavaScript, WebSocket, Live Web Audio, Jetty Webserver

- Client side tools
  - Eclipse, Maven2

- Client Design Flow

Client is the starting point of speech recognition process where audio is captured, sent to the Jetty Webserver over WebSocket.
Figure 3.3 shows the client user interface.

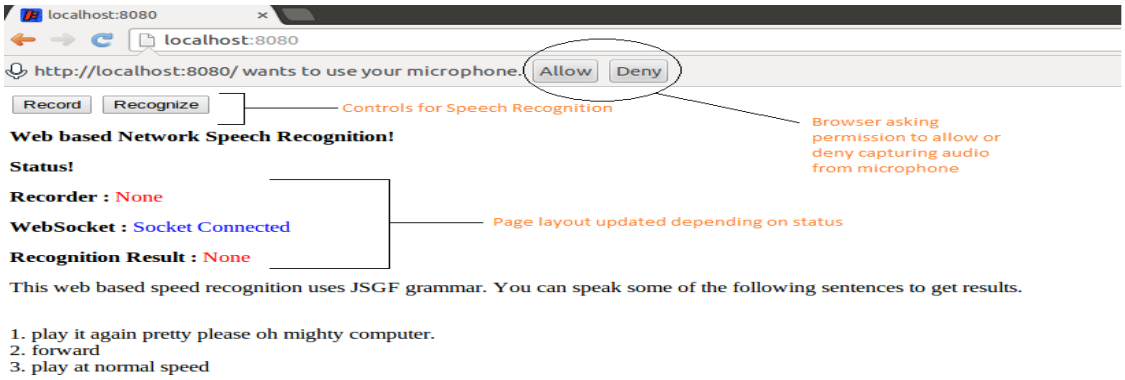

Figure 3.3: Network Speech Recognition Client

- The client page resides in Jetty Webserver and is served at 8080 port.
- The page is loaded in the Google Canary Web browser.
- User is asked to allow/deny capturing audio from microphone.
- Record and Recognize button serves the functionality of recording and recognizing Speech respectively.
- Status frame shows the current status of overall system.
- User is entitled to speak the default grammar shown at the end of the page.

The part of the page layout that gets updated on different events is shown in above figure. It shows status such as recording, WebSocket status, and speech recognition status and its results.

One important entity that can be seen in above figure is the WebSocket status, which is connected as soon the client page loading is completed. The following procedure leads to a successful Recognition using client page.

- Google Canary Web Browser asks user to allow / deny permission for capturing audio from microphone.
- User clicks on “Allow” button to allow capturing audio from microphone.
- User clicks “Record” button and start speaking.

- User clicks “Recognize” button to stop recording and start Speech Recognition process.
- The Speech is processed by the Network Speech Recognition server.
- Results are returned back to the user client.
- Client page “Recognition Result” status label is updated with the result.

Figure 3.4 shows page layout update for Recognition in progress phase and successful Speech Recognition phase.

Figure 3.4: Network Speech Recognition Client status

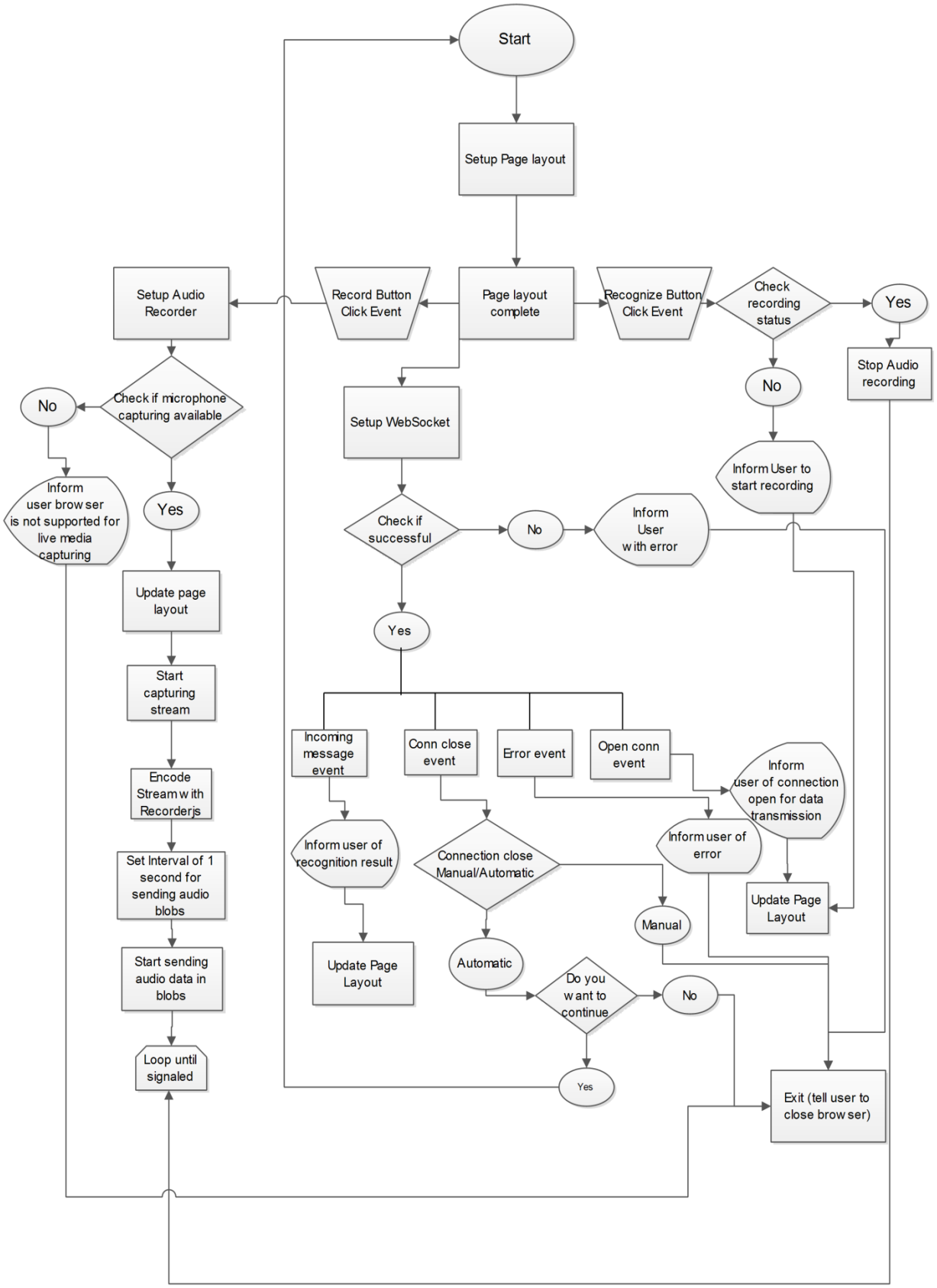

Figure 3.5: Network Speech Recognition Client Design Flow

Figure 3.5 shows detailed client design flow. This design flow is tested to work with MAC OSX due to the limitation in Microsoft Windows and Linux, in which the “Live Web Audio Input” feature is still under development. Also, there has been no way of accessing raw audio data, so to realize this RecorderJS [24] JavaScript library is used. It allows the user to capture audio, set an interval of time and returns an audio blob worth of the specified interval amount of data.

Figure 3.5 shows that user loads the speech recognition client Web page served by Jetty Webserver in Google Canary development version. The page has two buttons “Record” and “Recognize” linked to record and recognize method respectively.

When the Webpage loading is completed, a WebSocket connection is tried to the remote server. A condition is set to check if the connection is successful. Upon failure the user is informed about the error information (mostly if Web browser doesn’t support WebSocket) and to exit the Web browser. If the connection is successful, four event listeners are setup for listening different events:

- **Incoming Message Event Listener**
  This event listens for incoming messages via connected WebSockets from the remote Jetty Webserver. It may include binary data or simple text messages. In this design we have used it for simple plain text messages. Message includes ‘Speech Recognized’ result. Page layout is updated for user with recognized speech result.

- **Connection Close Event Listener**
  This event listens for WebSocket closing status. WebSocket may close due to inactivity, manually or some error. The user is informed with appropriate message.
  a. If it is closed manually, the user is informed about closed WebSocket status. The user is guided to close Web browser.
  b. If it is closed due to inactivity or error, the user is informed with appropriate information. The user is asked if he/she wants to continue. If the user wants to continue the user is guided to refresh the page which will setup the client page again. If user intends to not continue, the user is guided to close the Web browser.

- **Error Event Listener**
  In case of error during WebSocket connection creation, the user is informed and is guided to close Web browser. The error may be caused of no support of WebSocket in current Web browser.

- **Open Connection Event**
  When a Web socket connection is opened to remote Jetty Webserver, this event gets called informing user of Web socket connection opened. Page layout is updated with Web socket connection open status information.

As the user is now connected to the remote Jetty Webserver using WebSocket, the user can use two buttons for interaction to start the speech recognition process.

User clicks on Record Button and recording setup proceeds with checking if the user client has the ability to capture audio i.e. checking Live Web Audio Input feature. In unsuccessful case the user is informed with the information that the user client is not supporting "Live Web Audio Input" feature and guides it to close the user client. In case of success, the page layout is updated.

An instance of RecorderJS is created. Audio capturing is started and a stream object is provided to RecorderJS instance method "record" to record and encode audio and return audio blobs in its call back method. The audio blobs are sent on WebSocket to the Webserver every second until it is stopped by RecorderJS instance "stop" method, page layout is updated.

When user clicks on Recognize button, it checks for condition to see if it is already recording or not. In case of no recording, user is informed to start recording by clicking on the "Record" button and the page layout is updated or else a signal is sent to RecorderJS instance to stop recording. The audio blob sending is stopped.

- Server Design Flow

The detailed server design flow is shown in figure 3.6. Few terms have to be explained before going through elaboration of detailed design flow. SphinxServer is main Java class responsible for executing Webserver instance on Port 9999. It listens for incoming WebSocket connection and forwards the connection requests to the WebSocket Handler. WebSocket Handler is responsible for handling incoming WebSocket connection and registers them for tracking. SphinxRecognizer [25] is the Java class responsible for executing audio speech queries from WebSocket clients to CmuSphinx to process speech and return result back to the WebSocket client. The term Recognizer is used by recognition engine as a recognition engine object to recognize speech.

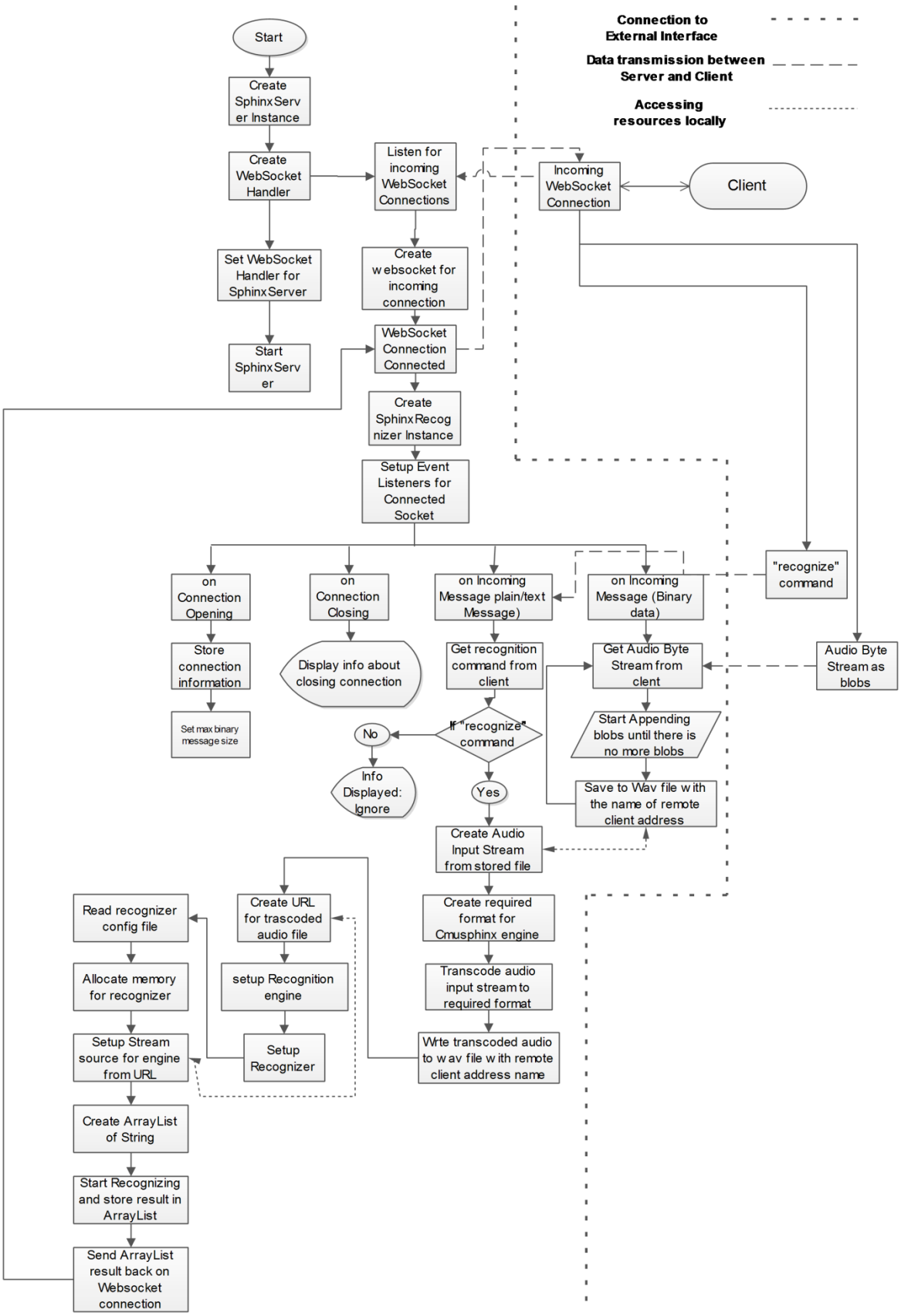

Figure 3.6: Network Speech Recognition Server Design Flow

The SphinxServer is initiated by creating a *SphinxServer* instance on Port 9999 (in our implementation). The WebSocket Handler for handling incoming WebSocket connection is created immediately. As mentioned above WebSocket handler handles the incoming WebSocket connection and registers them, so it waits for incoming connection and as soon a WebSocket connection request is received, it creates a WebSocket connection and registers its information for tracking purposes. As soon as the WebSocket connection is created for the client, a SphinxRecognizer instance is created for that client to process its Speech requests queries. Every WebSocket client uses its own instance of SphinxRecognizer to avoid usage of each other resources resulting in conflicts during Speech Recognition. When the WebSocket connection is created for the client, its event listeners are setup. Following are the list of event listeners' setup for any events that may occur:

- **On Connection Opening**
  This event gets called when a connection is opened between server and client. We register client information in this event and set the binary message size to enable WebSocket to receive audio blobs over it.

- **On Connection Closing**
  Information is logged and displayed to users for a possible closing of connection with the appropriate information i.e. "WebSocket Connection Closed due to inactivity for 230 ms". It may appear due to inactivity or some error.

- **On Incoming Messages (Binary Audio data)**
  Audio blobs are received in this message event. We get audio blob of 1 second as byte stream and append it to a .wav file with temporary file name. Audio blobs are appended to the same temporary file till the last blob received. Once audio blob receiving is finished (means the User has clicked the "Recognize" button on client Web page results in stopping recording and eventually stopping audio blobs sending) the file is saved to disk with the name of the remote client IP address. The reason that the file is stored as the name of client IP address is due to the fact that, the IP addresses are unique and the audio recordings can be saved with the IP address name to avoid conflict of using same file by multiple clients during Speech Recognition, which may result in erroneous result.

- **On Incoming Messages (Plain/text Messages)**
  As soon as the client clicks on "Recognize" button on the client Web page, the recording is stopped and a string "recognize" is sent to the server which is being received in this event. The check is done for the incoming message if it has a string "recognize". If it fails, the message is ignored and the recognition process is halted. If it succeeds we create an audio input stream from the .wav file

which we created in the above mentioned Incoming Binary Message event listener i.e. .wav file with the name of IP address of remote client. We need to create audio input stream for the reason that CmuSphinx engine requires specific parameter encoded with .waw file i.e. *"*16000, PCM Signed, mono, Big endian false*"*. So to transcode the above saved file we need to create an audio input Stream from above mentioned stored .wav file and transcode it to the required format and save it again as a .wav file with the same name. As AudioInput Stream is created, we create an AudioFormat object provided by Java to create a new required audio format and then write the new .wav file with the newly created audio format to the same remote client IP address name .wav file. Once transcoding of the .wav file is fisnihed an URL object of that transcoded file is created. Before feeding the URL file to the CmuSphinx we need to setup the Recognizer.

We setup Recognizer by reading the recognizer config file [26] which is an xml file and has all the information to setup Recognition engine such as using grammar or language models, tuning parameters, allocating memory for different sub components of the Recognition engine, using different type of stream sources to process (in our case File stream source), etc. Memory is allocated for the recognizer; stream is setup from above mentioned file URL. ArrayList object of string is created for storing processed speech result. Recognition starts and results are returned as ArrayList. ArrayList is converted to String and send it back on the connected WebSocket connection to the client.

- System Sequence Flow

Complete sequence flow diagram is shown in figure 3.7.

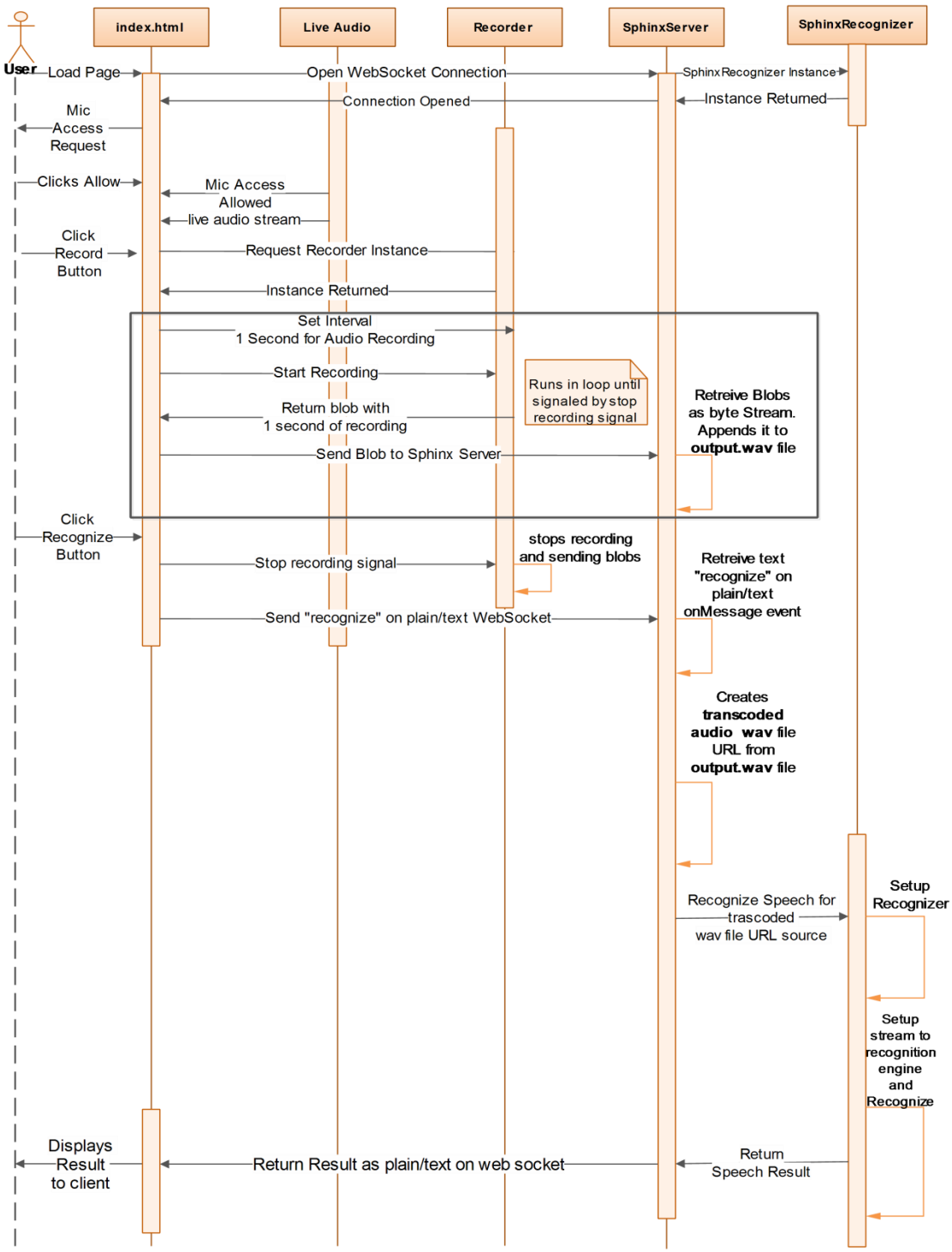

Figure 3.7: Network Speech Recognition System Sequence Flow

Now we are going to demonstrate the sequence flow of an input speech recognition request in aforementioned sequence diagram. User opens Google Canary Web Browser loads the client page i.e. index.html served by Jetty Server running on 8080 port. Once the page is loaded a WebSocket connection is created to the SphinxServer which is running on Port 9999. As soon as the SphinxServer receives the request it gives it to the WebSocket handler [27] and at the same time a SphinxRecognizer instance is created for that particular WebSocket connection. SphinxServer opens the connection for the remote WebSocket client. The client page asks user if he wants access to microphone (Allow/Deny). User clicks "Allow" button allowing microphone access, access is granted and a Live Web Audio stream is created in the form of stream object.

User clicks on "Record" button on the client page, an instance for RecorderJS class is requested. RecorderJS returns an instance to the client. An interval of 1 second is set for audio recording. Audio recording is started by inputting Live Web Audio stream from microphone to the RecorderJS instance method "record" and 1 second worth of audio is return in form of audio blob. The audio blob is sent over the WebSocket connection to the SphinxServer every second. Recording and sending of audio blobs every second is done in a loop until otherwise signalled by User. The Audio blobs sent to SphinxServer every second is received as byte stream and it is appended to a temporary file output.wav file.

User clicks on the "Recognize" button on the client page. It sends a signal to RecorderJS to stop recording by using RecorderJS method "stop" and the audio blob sending is also stopped at the same time. The temporary file "output.wav" is renamed to the remote client IP address name i.e. remote.client.ip.address.wav file. A string "recognize" is also send on the WebSocket connection to the SphinxServer which is received on incoming plain/text message event listener. Once it recognizes the message, a transcoded audio wav file URL is created by transcoding the above mentioned remote.user.ip.address.wav file and then creating a URL object of it.

Recognizer is setup to recognize speech from the above mentioned transcoded wav file. A stream is setup from the transcoded file in recognition engine by input its URL object and then starting recognition.

The results are returned back to te SphinxServer from where it is send back on WebSocket connected connection to the client page. Client page is updated when it receives the result and displayed it to the user of recognized result.

4. EVALUATION

The evaluation has been done in the form of testing our implemented Network Recognition System and traditional HTTP Request / Response based Nuance Network Speech Recognition system. The system throughput has been compared at the end of the test later in this chapter.

4.1 Testing Network Recognition System

In this chapter the tests of our implementation are described. Two systems were prepared for testing, i.e. the server and the client. The server hosts two Jetty Webservers, one for serving the client Website and other for accepting incoming WebSocket connections. Following table shows the hardware used for this testing.

	Server	Client
Processor	Intel Core 2 Duo	Intel iCore 3
RAM	2 GB	4GB
HDD	80 GB	320 GB
OS	Linux	MAC OSX
Connectivity	Wireless (802.11)	Wireless 802.11 (802.11)

Table 4.1: Network Recognition System Hardware and OS

Our client uses Google Canary Web browser on Mac OSX to record audio and send the audio data to the server via WebSocket. The client then waits for the recognized speech result.

The test were also done using third party engine i.e. Nuance Network Speech Recognition engine using HTTP Request / Response using the third party Nuance Network Speech Recognition Client application in order to compare its throughput with our designed system.

4.2 Client

The client part includes the client Web page, responsible for initiating the recording and recognition process. It requires user interaction for allowing access to the microphone or starting / stopping recording.

The main purpose of this test was to check a successful recorded speech recognition request in real time via WebSocket. An audio speech was recorded for 10 seconds. The audio data was then transmitted every second, to the remote server, in the form of audio blobs. The recording is stopped once the 10 seconds have elapsed.

Logging was enabled for debugging purposes. Log shows the flow of application, from allowing microphone until stopping the sending of audio blobs, by signalling recorder to stop recording.

The test has been done on Mac OSX with Google Canary browser running. Logs are collected via console provided in Google Canary browser.

Figure 4.1 shows the client page loaded in the browser. The status layout shows the current status of the page layout. The recognition engine has been set with the grammar listed on the client page, means it can only understand the specific set of grammar. As it was mentioned earlier, the recognition engine can be tuned and extended with a bigger list of grammar upon requirements.

In this test the user clicked the record button, upon which the highlighted text was spoken and the result was returned via the speech recognition engine.

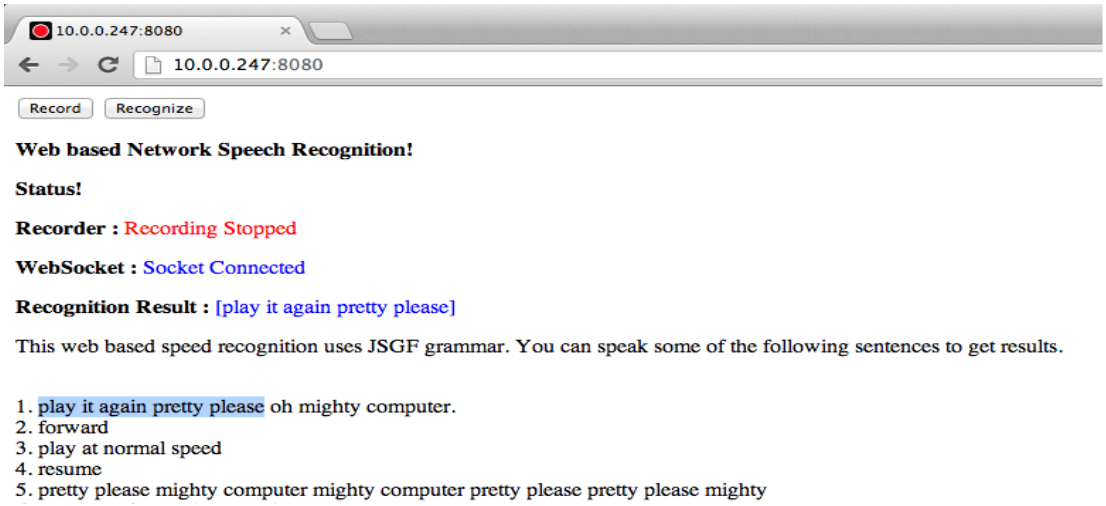

Figure 4.1: Network Speech Recognition Client Test layout

Figure 4.2 shows the detailed log with the flow of application and the audio blobs that were sent to the remote server every second. The user interaction can also be seen in the logs.

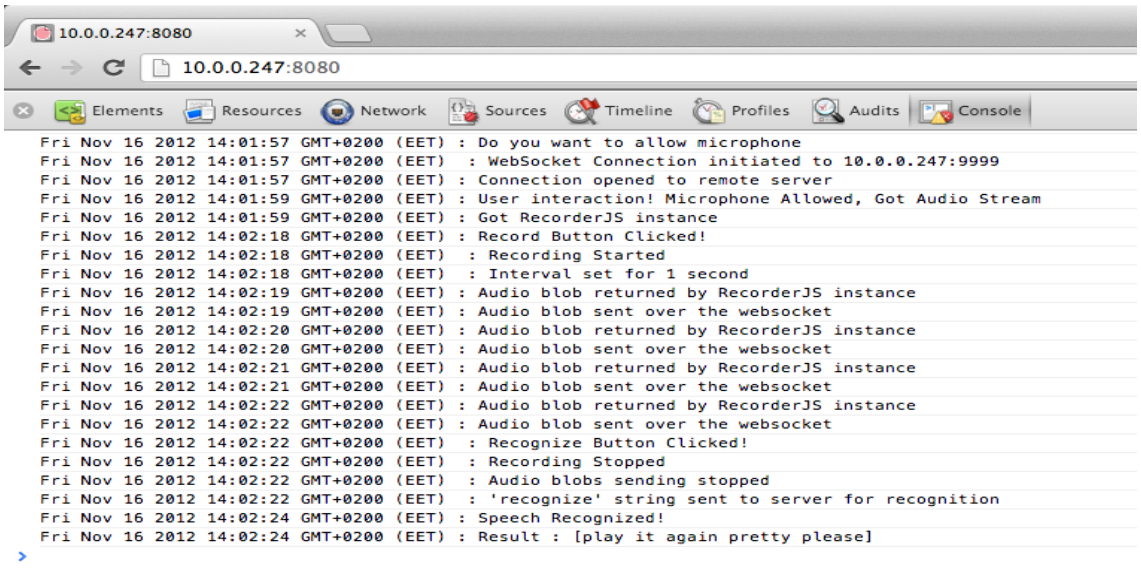

Figure 4.2: Google Canary Debug Console: Client Flow Logs

The following table shows an analysis of the speech recognition engine. Provided all the audio data has arrived at the server, it takes only 2 seconds for recognition engine to recognize 4 seconds worth of input audio speech. The total time elapsed including recording and speech recognition is equivalent to 6 seconds. The speech recognition time can vary with audio speech length, as it requires more processing of the recorded audio in order to accumulate the results.

Audio Speech of 4 second		
4 Second worth of audio Speech Recognition analysis	**Audio Recording Timestamp**	**Speech Recognition Time Stamp**
	14:02:19	14:02:22
	14:02:20	14:02:24
	14:02:21	
	14:02:22	
Time	**0:00:04**	**0:00:02**
Total Time Elapsed	**0:00:04 + 0:00:02 = 0:00:06**	

Table 4.2: Network Speech Recognition test

4.3 Server

The Server part includes two Webservers; a Sphinx Server that processes speech recognition requests and another server to serve the client page to the client. Figure 4.3 shows the directory structure of two servers. It also shows the Webserver running on port 8080 serving client page for audio recording and sending audio blobs to the server.

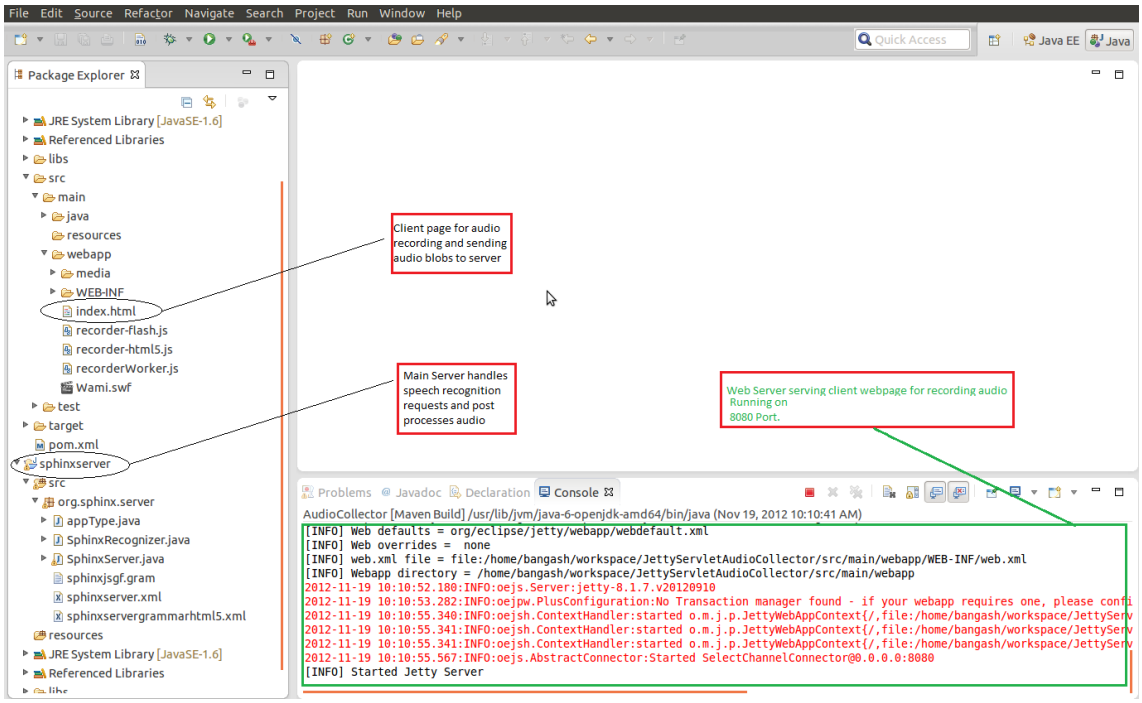

Figure 4.3: Network Speech Recognition System Tree Layout

Figure 4.4 shows the SphinxServer running on Port 9999, waiting for incoming Web-Socket connections. It does the audio post processing as well.

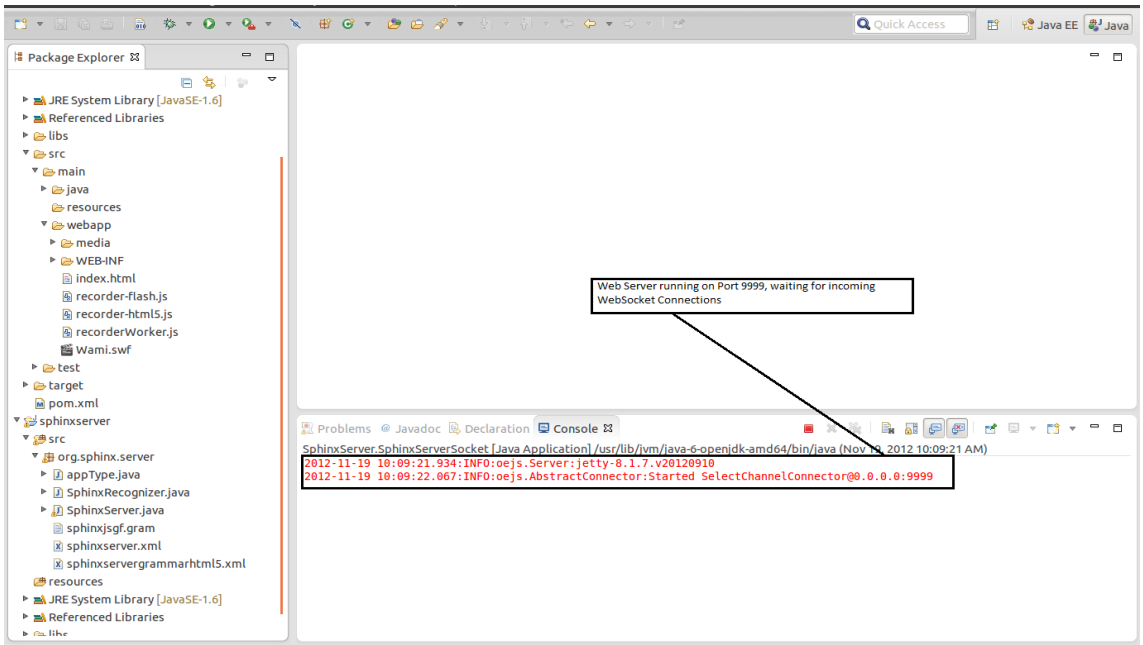

Figure 4.4: Network Speech Recognition Server Instance running on Port 9999

In the test, WebSocket connection was created to the SphinxServer. Audio Speech was recorded through client page and audio blobs were sent to the SphinxServer through

WebSocket connection. Audio was post processed to the CmuSphinx requirement and then fed into the engine to get Recognition result.

The detailed flow of SphinxServer serving a client's Speech Recognition request can be found from the figure 4.5.

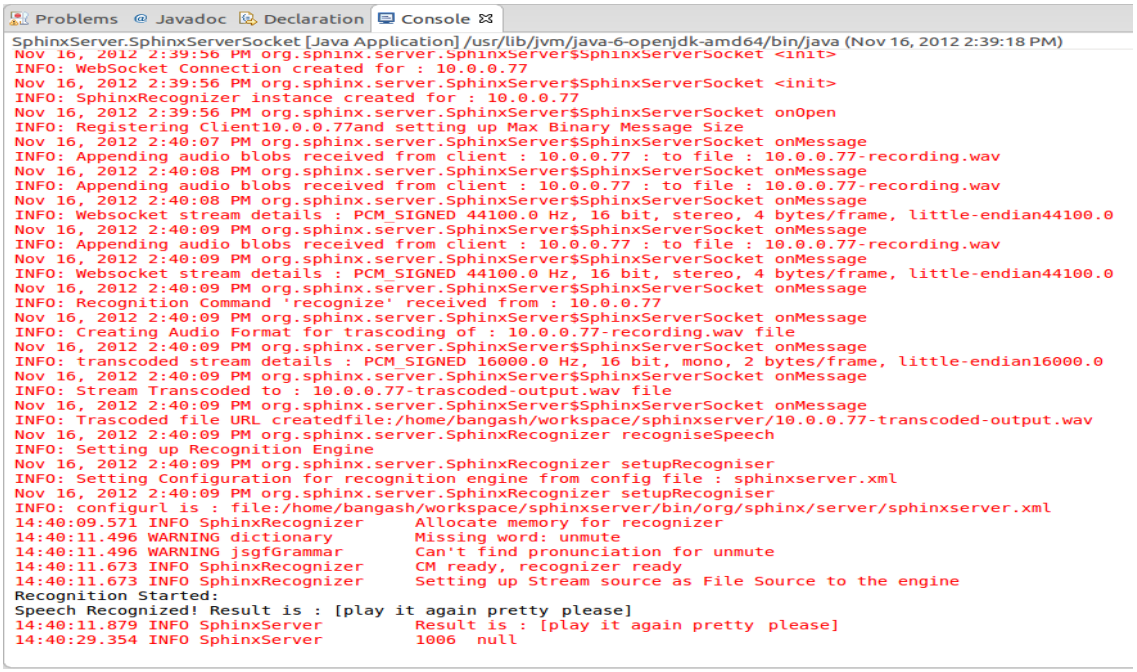

Figure 4.5: Eclipse Debug Console: Server flow logs

Figure 4.5 states the test result for speech recognition request processing. The log includes the detail of appending audio blobs to audio file, transcoding, setting up the Recognition engine and recognizing speech. The results of processed Speech Recognition can be seen at the end. Note: The timestamp differs in the client and server due to the non-synchronization of clocks.

4.2.1 Tests Comparison and Analysis

A group of tests for different audio Speech intervals is performed to estimate the average throughput of speech recognition engines using WebSocket and traditional Nuance HTTP Request / Response mechanism.
Following table shows the audio speech recorded time and its recognition per test.

Test #	Audio Speech (time in seconds)	Elapsed time for Speech Recognition (in seconds)
1	5	3
2	2	2
3	8	3
4	3	2
5	5	3

Table 4.2: Speech Recognition throughput using WebSocket and CmuSphinx Engine

Figure 4.6 shows throughput of speech recognition for above mentioned 5 tests.

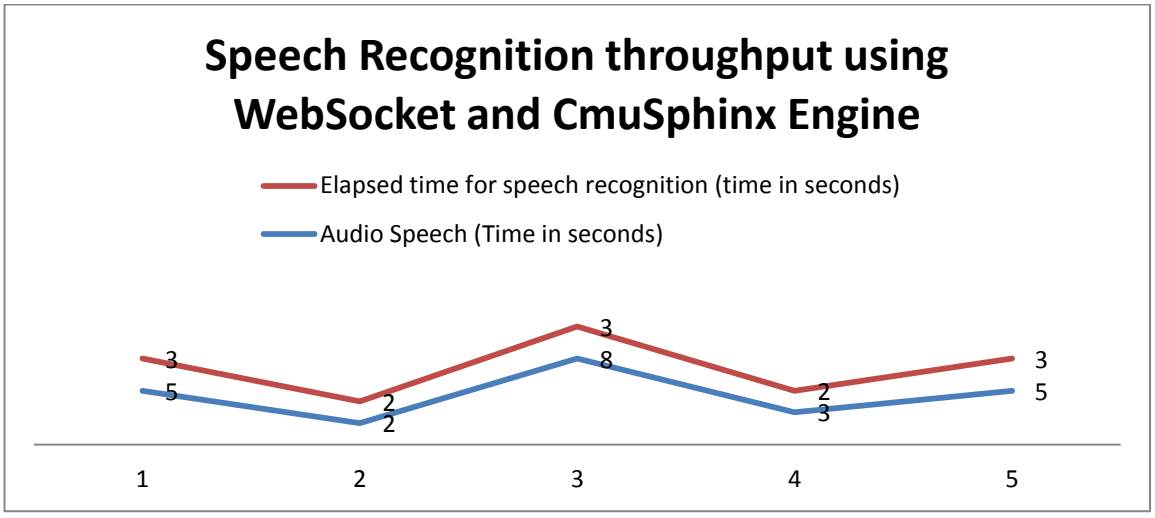

Figure 4.6: Speech Recognition Engine Throughput using WebSocket

The same audio recordings that were made in above tests were tested against Nuance's Network Speech Recognition using HTTP Request / Response. Following is the table showing the throughput.

Test #	Audio Speech (time in seconds)	Elapsed time for Speech Recognition (in seconds)
1	5	3
2	2	2
3	8	6
4	3	2
5	5	4

Table 4.3: Speech Recognition using HTTP Request / Response and Nuance engine

Figure 4.7 shows throughput for Nuance Network Speech Recognition engine.

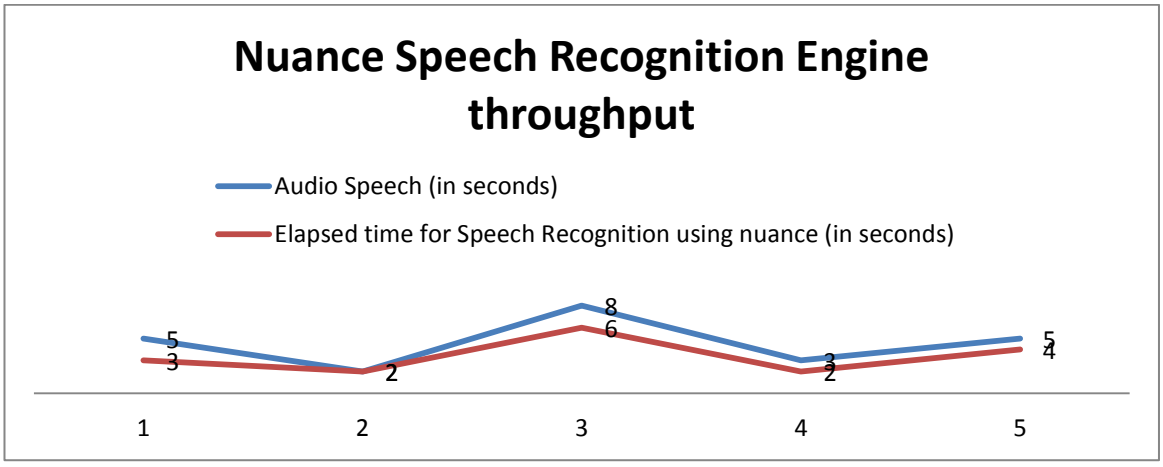

Figure 4.7: Nuance Network Speech Recognition engine throughput

Figure 4.8 shows throughput comparison of CmuSphinx engine and Nuance Network Speech Recognition engine.

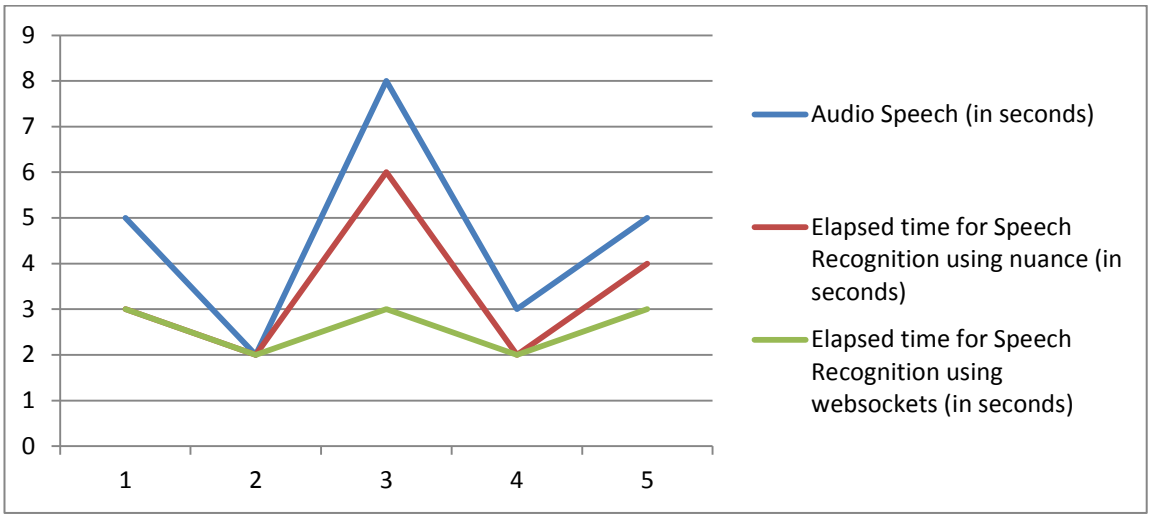

Figure 4.8: Comparison of Our System vs. Nuance Networks Speech Recognition system

Based on the above comparison, it was concluded that the network speech recognition using WebSocket performs better in the lengthy input speech request than the traditional HTTP Request Response mechanism. However it should be noted as well, that these two recognition engines rely on two very different networks. CmuSphinx resides in local network with a lot of other users' network traffic, whereas Nuance engine resides in their own private network, accessible over the Internet.

5. CONCLUSIONS

The Network Speech Recognition system architecture using HTTP Request / Response has limitations and overhead of headers. To overcome such limitations, a Network Speech Recognition system using WebSocket has been implemented. The utilization of new WebSocket feature provides a more robust and efficient way of transmitting data in real time using its full duplex transmission channels without overhead of headers.

The Network Speech Recognition system architecture using WebSockets has been implemented using several new technologies. However, our system architecture is limited by some of them due to their early stage of development.

The Live Web Audio feature for capturing audio, using Google Canary Browser, limits users to access raw PCM frames, which in turn requires transcoding audio to the required format.

To overcome this problem, we have used RecorderJS library to encode audio to a default standard of 44100Hz sampling rate, which is rarely required by the recognition engines. Thus, our solution has added some overhead of having to transcode the audio to the required audio format.

The designed system can be improved with increased throughput and efficiency by adding the feature of transcoding audio on the fly, during capturing the Live Web Audio stream. This will provide option of transcoding audio to the required format on the client side. Thus, it would reduce the overhead of audio post processing on the server side.

Client machines have more processing power nowadays. It is thanks to that extra processing power, that they can overcome the extra computation required to receive the audio blobs of the server side in order to append them to a file. This could have been much improved by creating a live transcoded audio stream with the required format from the client side, and then directly fed it into the speech recognition engine. There would be no need of appending audio blobs to a file at the server side for transcoding and then providing it to the recognition engine.

One of the major improvements in speech recognition is silence detection. Providing that the aforementioned mentioned features have been implemented, the silence detection feature would be a major improvement to the system, enhancing throughput of the Network Speech Recognition systems. The network congestion and high latency of audio frames that are arriving late at the server which are causing a silence to the engine, may lead the speech recognition engine to give wrong results.

A Network Speech Recognition engine that uses WebSockets provides a more robust and versatile way for processing and replying to speech recognition requests from

the clients. It provides clients with a vast vocabulary and language model, and a high processing power to analyse the speech recognition requests in real time.

REFERENCES

[1] Infotainment system, http://www.whyhighend.com/infotainment-system.html

[2] CmuSphinx Open Source Speech Recognition Engine, http://cmusphinx.sourceforge.net/

[3] HTTP Request / Response Mechanism, http://geekexplains.blogspot.com/2008/06/whats-http-explain-http-request-and.html

[4] WebSocket Protocol RFC, http://tools.ietf.org/html/rfc6455

[5] HTML5's Live Web Audio feature, http://updates.html5rocks.com/2012/09/Live-Web-Audio-Input-Enabled

[6] Acoustic Model, http://en.wikipedia.org/wiki/Acoustic_model

[7] Java Speech Grammar Format (JSGF) Grammar, http://www.w3.org/TR/jsgf/

[8] Language Modeling for Speech Recognition, http://research.microsoft.com/en-us/projects/language-modeling/

[9] Sphinx4 Recognizer, http://cmusphinx.sourceforge.net/sphinx4/

[10] HTTP Servlets, http://www.novocode.com/doc/servlet-essentials/chapter1.html

[11] Dynamic Web contents, http://suite101.com/article/static-and-dynamic-web-content-a97262

[12] Jetty Webserver, http://Jetty.codehaus.org/

[13] HTTP Servlet API, http://docs.oracle.com/javaee/1.4/api/javax/servlet/http/HttpServlet.html

[14] Web containers for Servlets, http://www.servletworld.com/servlet-tutorials/j2ee-web-container-introduction.html

[15] Jetty Configuration and example code, http://www.eclipse.org/jetty/documentation/current/

[16] HTML5, http://www.w3schools.com/html/html5_intro.asp

[17] HTML5 API, http://www.w3.org/TR/html5/

[18] Web Real Time Communication (WebRTC) project, http://www.webrtc.org

[19] World Wide Web Consortium (W3C), http://www.w3.org

[20] WebSocket API, http://www.w3.org/TR/2011/WD-websockets-20110419/

[21] Binary Large Object (BLOB), https://developer.mozilla.org/en-US/docs/DOM/Blob

[22] Java Audio System for audio post processing, http://docs.oracle.com/javase/1.4.2/docs/api/javax/sound/sampled/AudioSystem.html

[23] Maven2, HTTP Servlets Deployment tool, http://maven.apache.org/

[24] RecorderJS, JavaScript Library for recording audio, https://github.com/mattdiamond/Recorderjs

[25] SphinxRecognizer class for recognizing Speech, http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/jsapi/SphinxRecognizer.html

[26] CmuSphinx Configuration for setting up Recognition engine, http://CmuSphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/util/props/doc-files/ConfigurationManagement.html

[27] WebSocketHandler class for handling WebSocket connections, http://www.jarvana.com/jarvana/view/org/eclipse/jetty/jetty-websocket/8.0.0.M1/jetty-websocket-8.0.0.M1-javadoc.jar!/org/eclipse/jetty/websocket/WebSocketHandler.html

[28] Xing Yan, Lei Yang, Shanzhen Lan, Xiaolong Tong, Application of HTML5 multimedia, Computer Science and Information Processing (CSIP) 2012 International Conference, pp. 871 – 874, 24-26 August 2012.

[29] Pimentel V, Nickerson B.G, Communicating and Displaying Real-Time Data with WebSocket, IEEE Internet Computing, vol. 16, pp. 45 – 53, July - August 2012.

[30] Loreto S, Romano S.P, Real-Time Communications in the Web: Issues, Achievements, and Ongoing Standardization Efforts, IEEE Internet Computing, vol. 16, pp. 68 - 73, Sept – Oct 2012.