Making Speech Recognition Work on the Web

by

Christopher J. Varenhorst

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2011

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2011

Certified by .....
James R. Glass
Principal Research Scientist
Thesis Supervisor

Certified by .....
Scott Cyphers
Research Scientist
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Department Committee on Graduate Students

Making Speech Recognition Work on the Web

by

Christopher J. Varenhorst

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2011, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

We present an improved Audio Controller for **W**eb-**A**ccessible **M**ultimodal **I**nterface toolkit – a system that provides a simple way for developers to add speech recognition to web pages. Our improved system offers increased usability and performance for users and greater flexibility for developers. Tests performed showed a %36 increase in recognition response time in the best possible networking conditions. Preliminary tests shows a markedly improved users experience. The new Wowza platform also provides a means of upgrading other Audio Controllers easily.

Thesis Supervisor: James R. Glass
Title: Principal Research Scientist

Thesis Supervisor: Scott Cyphers
Title: Research Scientist

Contents

List of Figures

Chapter 1

Introduction and Background

With the widespread proliferation of the web, there is pressure to extend its functionality. A key area of opportunity is in the area of speech based applications. Existing browsers and standards have little support for audio or microphone access, let alone speech applications. The WAMI Toolkit ([11]) was created in 2008 to address this need. Built on top of existing web technologies, WAMI allows developers to deploy multimodal interfaces incorporating voice to any user with a web browser. WAMI has been very successful internally and with a few larger external users. It has been regularly used for data collection, and several papers have been published that make use of data collected using the WAMI toolkit[12]. Despite is successes, WAMI has been slow to receive support from a broad base of web application developers in general. Much of this stems from frustration end-users can experience when recording their voice for WAMI to process. In this thesis I describe my extension to WAMI's desktop speech input system, alleviating much of this frustration and discuss the problem of

speech input on the web in general.

Before describing our final solution I outline WAMI in its current state, its drawbacks, and discuss various related work.

1.1 WAMI - Web Accessible Multimodal Toolkit

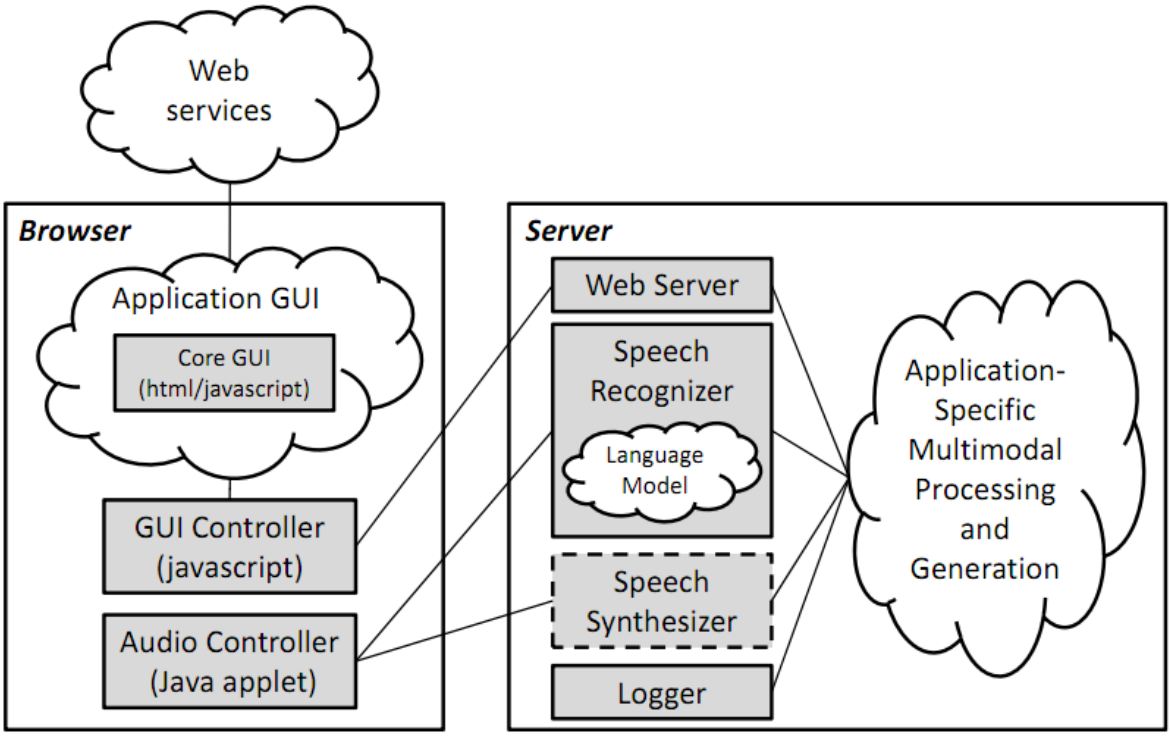

Figure 1-1: WAMI toolkit core platform. Shaded boxes show standard core components. White clouds indicate application-specic components. [11]

WAMI, the Spoken Language System Group's **W**eb-**A**ccessible **M**ultimodal **I**nterface

Toolkit makes it relatively simple to bring a speech based application to the web. By making this process easy and offering it as a service to the public, WAMI can promote the development of speech based applications, and support further development of speech recognition technology.

WAMI makes building speech applications easy by abstracting away the difficult parts away from the developer. WAMI handles the speech recognition and the audio recording. All the developer needs to make a WAMI application is to include a Javascript library and specify the grammar that they want to match the speech from the users to. The grammar defines the universe of possibilities the user could say, limiting what the recognizer will match the audio with. The user also must specify a response handler that is called with the recognition results after a user has recorded their voice.

WAMI itself is divided into a few key systems. (See Figure 1-1). A Javascript component is responsible for interfacing with the developer's code and inserting the Audio Controller. The Audio Controller is an embedded Java applet responsible for recording the user's audio from within the browser, sending this audio to the speech recognizer, and playing back audio. The speech recognizer accepts audio and matches it against the grammar. The WAMI web server manages the session of each user and logs information.

The Audio Controller is a critical part of the WAMI Toolkit. It is responsible for transferring speech data between the client and the server. There are separate Audio Controllers for different devices because web technologies do not yet support a standard way to accept

```
<html>
<head>
<title>WAMI Parrot</title>
<script src="http://wami.csail.mit.edu/portal/wami.js"> </script>

<script>
var myWamiApp;
function onLoad() {

// JSGF grammar the speech will be matched on
   var jsgf =
       "#JSGF V1.0;\n" +
       "grammar parrot;\n" +
       "public <top> = hello wami | i want a cracker | feed me;\n";

    var handlers = {"onRecognitionResult" : onWamiRecognitionResult};
    myWamiApp = new Wami.App(document.getElementById('AudioContainer'),
           handlers, "json", {}, {}, {"language" : "en-us", "grammar" : jsgf });
}
 function onWamiRecognitionResult(result) {
    alert("You said: " + result.hyps[0].text);
    setTimeout("myWamiApp.replayLastRecording()", 500); //playback audio
}

</script>
</head>

<body onload="onLoad()">
    <div id="AudioContainer"></div> <!--  The record button will be put here -->
    Hold the button and say: "Hello WAMI" or "I want a cracker!" or "Feed me!"
</body>
</html>
```

Figure 1-2: This is the code for complete web page that uses WAMI. Developers need to only use a minimal amount of Javascript to employ web based speech recognition.

speech from a user. WAMI has various Audio Controllers for Android, iPhone, and the desktop browser. For this thesis I focus on my development on the desktop browser based Audio Controller

1.1.1 Existing Java applet

A central task of any speech recognition system is accepting audio input from the speaker. This presents a particular challenge over the web as no standards exist supporting microphone or device input. The W3C is drafting support for a `<device>` tag that will eventually support microphone input inside HTML but the arrival of its implementation in browsers is still unclear.

To accept microphone input over the web, WAMI's existing Audio Controller is based on Java applet technology. Java-enabled web browsers are able to run specially packaged Java programs called applets inside of a web page. These fall outside of the HTML standard. WAMI's original Audio Controller used this technology because it's a straight forward way to provide access to the system's microphone. The applet works well within a laboratory environment but unfortunately can lead to a poor user experience "in the wild."

Developers are reluctant to make use of applets because not all user environments can run them reliably. A consistent pain point in user experience has always been WAMI's Java applet based Audio Controller. Despite a %91 market penetration, only %57 of the users of Quizlet.com successfully complete the WAMI test page. The current Java applet can have

a considerable startup time, requires a separate security approval from the user, and is not allowed in some environments. Moreover, major web browsers seem reluctant to continue support for Java applets. The OS X version of Google Chrome has only marginal applet support (WAMI does not function). The existing Java applet also has unnecessarily larger bandwidth requirements, limiting its use in low bandwidth low latency environments and causing subsequent frustration for the users. The applet also dictates the interface with which the user will record the voice, preventing application developers from customizing the experience to suit their own specific needs. These downsides primarily motivate my work to improve this experience.

1.2 SALT

One of the first pushed to bring speech recognition to the web was Kuansan Wang's Speech Application Language Tags or SALT[14]. It described an XML based markup language embedded in HTML that adds voice recognition properties to web application[7]. This was one of the first major attempts to bring a standard web speech recognition format to the web. SALT was heavily promoted by Microsoft, but unfortunately never gained acceptance from the W3C and has eventually replaced by VoiceXML. Microsoft created an add-in for Internet Explorer that allowed the casual user's browser to interpret SALT tags, but these never became widespread.

1.3 VoiceXML

SALT's "successor", VoiceXML is an W3C approved standard in XML format for specifying interactive voice dialogues between a human and a computer[10]. Though possible, VoiceXML implementation have failed to have much support for integrating dialogue systems directly into web pages, and more often relay on a secondary Voice Browser to interact with VoiceXML applications.

1.4 Google Chrome Speech API

The W3C HTML Speech Incubator Group was formed in August 2010 with the goal of integrating speech technologies into HTML5, while avoiding fragmentation and continuing to leverage the capabilities of an open web. Developers at Google submitted a Speech Input API Specification Draft to this group in October 2010. Google has implemented most of this draft in the Google Chrome browser. [8]

The Google implementation differs from WAMI in several key ways:

- The initial implementation of this draft only provides generic speech-to-text support for filling out forms. It does not provide powerful grammar matching.
- It cannot be used in a multimodal way. While Chrome is accepting speech from a user, no other activity from the user is allowed.
- Finally, and perhaps most importantly, it can only be used in recent versions of the

Chrome browser. The specification also does not say who is to provide the speech recognition service, and at this time it seems unlikely the major browser vendors are about to develop their own recognition service.

This specification also leap frogs over attempts to first even allow web developers microphone access from within a browser without the use of plugins.

1.5 Flash

Flash is a platform for developing rich internet applications. It is an embedded plugin included in browsers allowing them to view and use Flash content. Flash offers several capabilities that HTML alone does not, and, most importantly, it has become the most widely used method for accepting microphone input over the web. Flash has several advantages over Java applets including better performance, nearly ubiquitous browser support, and a smoother user experience.

Flash also makes use of the new patent and license free speech codec Speex. This codec is designed for capturing human speech, so its suits our needs quite well. Speex accomplishes this by using information about what sounds are important for the understanding of speech. For example, background noise and ambient silence are not relevant to speech comprehension so Speex can identify these components and de-value them. Speex can also identify the more important sounds like vowels and encode more information for these, while including less information for less important parts of speech sounds such as fricatives (the “f” and

"s" sounds). By communicating in Speex we can also reduce bandwidth requirements (see Section 4.2).

Unfortunately, Flash has several differences from Java applets that make development of a Flash based Audio Controller for WAMI a less straightforward process. Flash runs in a sandboxed environment with only limited access to the underlying system and with only limited functionality. This is part of what permits Flash to achieve such a smooth experience, though, because a browser can permit a Flash object to immediately run and expect it will not harm the user's environment. Therefore, before we can access the user's microphone to record information for WAMI, we must present a security panel to the user and they must approve access. I explain some of the idiosyncrasies and necessary workarounds of this process in Section 2.2.5.

In Flash's sandboxed environment, the only direct way to export audio data recorded from a microphone is to send it across a network using Adobe's proprietary RTMP protocol. Adobe sells a server product intended to accept this data but its cost is prohibitive[1] and its support for Linux is minimal. Originally RTMP was maintained as a proprietary protocol by Adobe, prompting numerous developers to reverse engineer it and release open source and commercial alternatives to Adobe's own Flash Media Server. In 2009 Adobe did release an RTMP specification but it failed to included numerous details essential to a complete implementation. In the work discussed later, we selected the Wowza Media Server[9] for its relative cheap price, high quality support, and extensibility.

[1]The middle tier Flash Media Server is $4,410

Chapter 2

Design

In this chapter I describe the goals of our improved Audio Controller, the solution's high-level architecture, and final implementation details and related technical information. The flow of information through the components of the system is as follows. (See Figure 2-1 for a sketch of the system design)

- To prompt the user for audio information, a WAMI Flash Audio Controller is presented to the user on a WAMI enabled site.
- The user interacts with the controller in their browser, recording their voice with a microphone.
- Audio is encoded as Speex on the client's browser and sent to the Wowza media server using RTMP.
- A Wowza module determines the record URL for posting data to, and passes the Speex

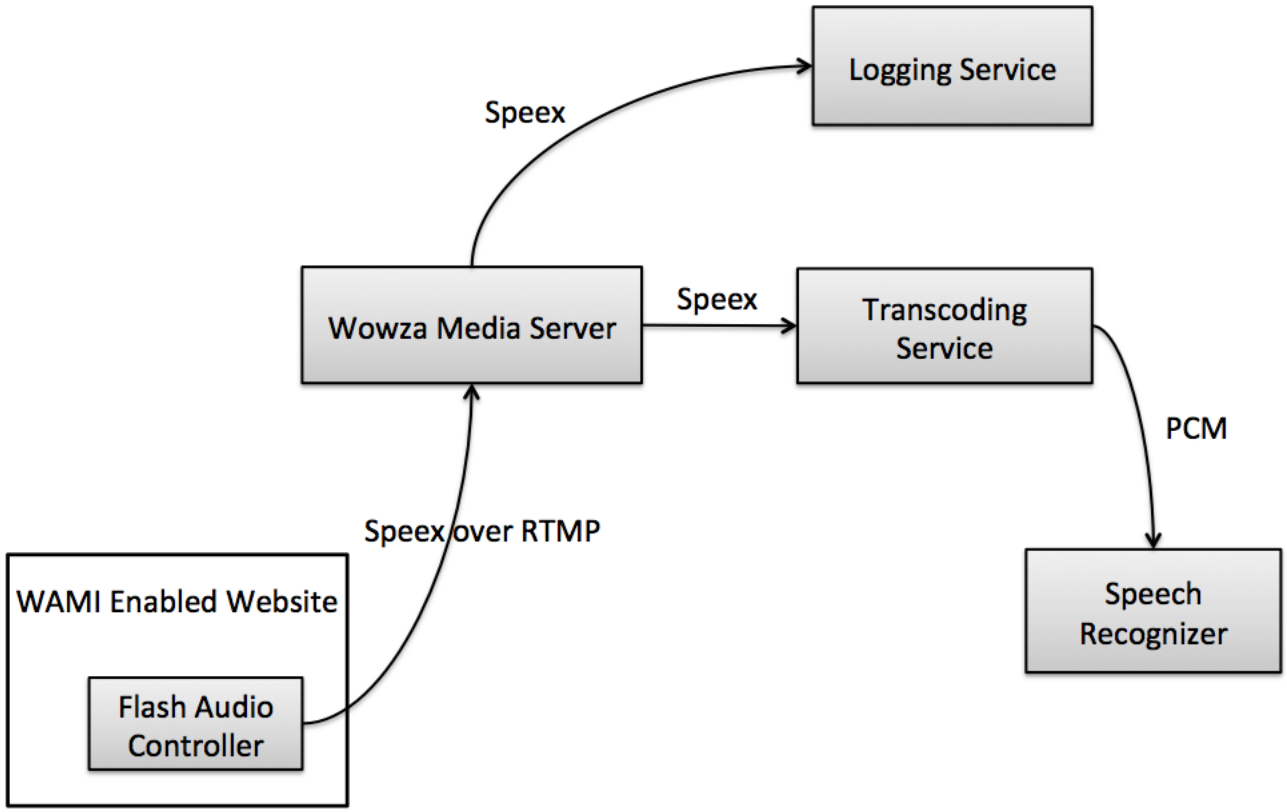

Figure 2-1: Preliminary design of of WAMI extension. Audio data is taken from Flash client over RTMP, transcoded to PCM, and sent to the speech recognition system.

data to the transcoding service.

- The trancoding service converts the waveform into 8kHz sampled unsigned 16-bit little endian sampled PCM, and forwards this to a speech recognizer server.

At no point in this process is the information substantially buffered or batched. In good conditions, the speech recognizer will start receiving audio to process one half second after the user starts recording. The delay is mainly there for FFmpeg to start transcoding audio. The other components of the WAMI architecture are unchanged.

2.1 Design Goals

The goal of the Flash based Audio Controller is to provide an excellent user experience right away while making it easy for web developers to customize WAMI to suit their own needs and desired user experience. To this end, the specific audio interface GUI is not specified by WAMI but is entirely configurable by the user. WAMI exists as the "pipes" providing speech recording and recognition in the background but not dictating any user experience. However, to encourage rapid prototyping, a default UI is provided if the developer asks for it. This allows powerful customization for those that need it, but instant usability for those that don't. A particularly difficult aspect of a GUI implementation is dealing with the Flash microphone permission panel. To make things easier on the developer, WAMI allows users to customize the UI, but still takes much of the difficultly out of dealing with this particular aspect.

2.2 Flash

The Flash object communicates with the WAMI Javascript application through the API specified below. Flash is able to accept calls and insert functions directly into Javascript, enabling simple and tight integration.

These are javascript functions the Flash object exposes to the WAMI Javascript library.

- `startRecording(startedCallback, doneCallback, failCallback)` - When called this starts recording audio from the microphone. Once recording starts, it calls the `startedCallback` in the Javascript. `doneCallback` is called when recording is stopped.
- `stopRecording()` - This stops the audio recording and triggers the `doneCallback`.
- `getActivityLevel()` - This returns the microphone activity level (volume) between 1 and 100.
- `playWAVFromURL(url, doneCallback)` - This plays a WAV file at the specified URL. If the URL is on a different domain, the root of the URL must contain a valid `crossdomain.xml`. See Section 2.2.3.
- `stopPlaying()` - stops WAV playback and triggers the `doneCallback`.

The Flash object must also be initialized with a number of parameters that are important for controlling its behavior.

- `onSecurityGranted` - This is the name of function to be called if the microphone security checks succeeds. See Section 2.2.5.

- `onSecurityWarning` - This is the name of function to be called if the microphone security checks fails. See Section 2.2.5.

- `recordURL` - This is the WAMI URL that recorded audio should be posted to. Its is encoded in a special way, as described in Section 2.3.1.

- `wowzaLocation` - This is the IP or hostname of the Wowza server.

All callbacks are specified as strings since Flash cannot accept anonymous Javascript functions as parameters. Its important to note that this API is a private API only used for communication between the Flash object and the WAMI Javascript library. Developers do not interact directly with it.

2.2.1 Invisible Flash Object

Since we don't want to dictate any particular interface for developers, we design our Flash object responsible for microphone access to be "invisible" to the user. Literally this means that a transparent, 1 by 1 pixel Flash object is inserted into the page. This object then communicates with the rest of page through a specified Javascript API.

2.2.2 Speex

Flash 10 supports encoding audio in the Speex format. Speex is a patent and license free codec designed for encoding human speech. Flash's Speex implementation samples at 16kHz and allows the encode quality to be adjusted between 0 and 10. At an encode quality of 10, Speex data is encoded at 42.2kilobits/sec, while at 0 only 3.95 kilobits/sec is required. For our initial implementation, an encode quality of 7 (23.8 kbps) was chosen. Since the intention of Speex is the efficiently capture the "important" parts of human speech, the encode quality's effects on recognition quality is not yet fully understood. This is discussed further in Section 5.1.

2.2.3 WAV Playback

The WAMI Audio Controller is also responsible for the playback of audio when necessary. Unfortunately no universal method of WAV playback exists in today's browser landscape. Recent versions of most major browsers support the playback of WAV files natively in the browser using the `<audio>` described in the latest HTML5 specfication. Unfortunately though, many users have browsers that do not yet support these features so an alternative needs be explored. Flash cannot natively play WAV files either, however Flash is powerful enough to allow the implementation of a WAV player. Using the popforge library ([5]) we include a WAV player in our Audio Controller.

However, because our WAV player processes files by first requesting them in a generic

way, our requests are subject to the Flash's cross-domain policy, which by default forbids generic access to data on the domains that the Flash object is not hosted. This would not be an issue if we used Flash's native media player API, but because we need to access the WAV file in a generic way, our requests are subject to this policy. To permit Flash to access other domains, those domains must include a `crossdomain.xml` file at their root directory permitting Flash client's access. For example, the `crossdomain.xml` file for facebook.com is located at `http://facebook.com/crossdomain.xml`

See Figure 2-2 for an example of a `crossdomain.xml` file that permits access to it from all domains. This is the file that currently exists at the root directory of our WAMI server.

```
<cross-domain-policy>
<site-control permitted-cross-domain-policies="master-only"/>
<allow-access-from domain="*"/>
<allow-http-request-headers-from domain="*" headers="*"/>
</cross-domain-policy>
```

Figure 2-2: Example of a `crossdomain.xml` served at a web domain's root that permits Flash clients from all domains to access this domain's contents.

2.2.4 Object Insertion

Flash is a browser extension, so the means of including it in a web page can vary across browsers. When inserting the WAMI Flash objects into a webpage, we must take browser vendor and version into account. We also need to ensure that the user has Flash installed already, and that the version they do have installed is adequate. To make this easier to

Figure 2-3: The Flash security dialogue. The user must grant our application access to their microphone to use WAMI. This panel is 220x145 pixels and this cannot be changed.

manage, we use the popular open source project SWFObject Javascript library to managing Flash insertion. When minified SWFObject is less than 1k bytes, keeping its overhead minimal. SWFObject also aids in configuring the initialization parameters given to the Flash objects.

2.2.5 Security Panel

The Flash platform requires approval from the user before access to a microphone is granted. To prevent malicious manipulation of this process, Flash offers the developer very little control over how this process is presented to the user. Unfortunately this requires us to implement a workaround to provide a good user experience.

As documented above, the Flash object that is responsible for providing the WAMI functionality is hidden in the page and not visible to the user. However, the only way that Flash permits the microphone security dialogue to be displayed to the user is inside a Flash

window (see Figure 2-3). This window must be at least the size of the security dialogue, which happens to be 220x145 pixels.

To support this, we have to show the user another Flash object when it's been detected that permission is required. Unfortunately in the permissions dialogue, granting permission without clicking "Remember" only grants permission to that particular Flash object, so for our hidden Flash object to ever receive permission, the "Remember" button must be clicked by the user.

Complicating things further, a Flash object initialized without microphone permission will never have microphone permission until a security dialogue within that object grants it. Because our hidden object can never show this dialogue, we must reinitialize when permission is granted. Complicating thing once again, Flash does not provide a way for an application to know when the user has finished with the security panel. This is a known bug and has was first listed on Adobe's support site 4/15/2008. Regrettably it still remains unaddressed. However a workaround is posted that continually performs an event that throws an exception if the Security Panel is visible to detect when it is closed. We implement this workaround to determine when the panel is closed and our hidden Flash object will have security permission.

Its also important to note that the 220x145 security panel does not take into account the current zoom level of the browser. Zoom level is a common browser feature that allows arbitrary scaling of webpage content. Since the contents of Flash are not rendered by the browser, it does not scale as its container does and often times the contents of a Flash object

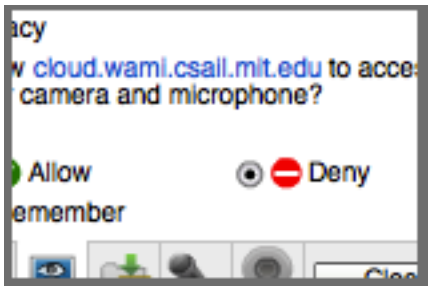

Figure 2-4: This is the Flash Security Panel after the browser's zoom level has been changed. Because Flash content does not scale like the Flash object's container, the contents are 'too big' for the container.

will appear cut off or obscured. (See Figure 2-4 for an example of this.) Complicating things further, no standard means of detecting a browser's zoom level exists, making a workarounds more difficult. There is no established workaround for this, though its remains an infrequent problem. Adobe's own website does not take browser zoom level into account. For a relevant discussion on browser zoom level see [1].

The Flash security panel is particularly cautious to protect itself from "clickjacking attacks". Clickjacking, which can be understand as an instance of the "confused deputy" problem, is when a user is tricked into interacting with a web application in a way the user does not intend [13]. In this context, a clickjacking attack would be one that deceives the user into granting a malicious Flash application microphone permission without intending to do so. To avoid this, the security panel will not respond to clicks if any part of it is covered by another other element, and it will not accept clicks until it has been fully visible for at least 2 seconds. These protections have a number of side affects that can negatively affect user experience:

- **If tightly bound, the security panel will not appear if a brower's zoom level is below 1.** This is because unlike most other elements of a web page, Flash objects do not scale with the zoom level. This causes the Flash container to be smaller than its contents, preventing it from being displayed

- **The security panel is unresponsive for the first two seconds** To ensure that the user has seen the security panel before it can be used, it will not accept clicks for the first 2 seconds. This is an undocumented feature of the security panel and was discovered after testing.

2.3 Wowza module

After audio has left the client's browser over RTMP, the RTMP must be decoded and the media processed by the server. To handle the multiple simultaneous RTMP streams we make use of the Wowza Media Server[9].

Wowza provides us with a scalable platform for managing the streaming of audio communicated in Adobe's Flash's RTMP communication. Wowza also provides the RTMPT and RTMPS implementation that forwards RTMP over HTTP and SSL+HTTP respectively for bypassing network restrictions.

After accepting the audio information at Wowza, we need to forward it to the WAMI service. WAMI needs audio in an unencoded 8kHz 16-bit PCM format. To accommodate this, we developed a Wowza module that uses FFmpeg[4] to decode the Speex data to PCM,

and then posts it over HTTP to that WAMI session's particular record URL. An overview of this process follows:

1. **Start FFmpeg process**. See 2.3.2 for the exact FFmpeg parameters.
2. **Decode record URL** from the Wowza stream instance name. (Section **??**)
3. **Open HTTP connection** to record URL and start an HTTP POST.
4. **Start `pipeMover` thread** to continuously move incoming audio to FFmpeg, and the FFmpeg output to the HTTP connection
5. **Write first FLV headers** to buffer. These are used to let FFmpeg know what type of media follows in the rest of the stream.
6. **Continue to write incoming audio data to buffer**
7. **Stop streaming**, flush buffers and close.
8. **Wait for FFmpeg** to finish conversion.
9. **Flush final FFmpeg output** to HTTP connection and close the connection.
10. **Kill FFmpeg process** if it still exists.

All of the above are done in memory without on-disk buffering. Unix pipes, STDIN, and STDOUT are used to put Speex in and bring PCM out of the FFmpeg process.

2.3.1 Record URL

Audio data coming in to Wowza needs to be posted to the correct record URL for WAMI to process. This record URL contains a session specific ID. To keep the communication simple, the record url is "piggy-backed" to Wowza by encoding it in the Wowza instance name. This is just a piece of information associated with a particular stream. There are character restrictions on this instance name, so the record URL is transformed.

The transformation is identical to a base 64[2] encoding with the exception that "/" in the base64 encoded result is turned into "=S". This is because a "/" cannot be escaped directly, and cannot be part of a valid instance name. "=S" can never occur in a base64 encoding.

2.3.2 Decoding Speex

Wowza takes care of re-assembling the RTMP traffic, but offers no capability to transcode the media. To accomplish the media conversion we use the powerful and open source FFmpeg[4] media processor. Though interfacing more directly with FFmpeg's media processing library, `libavcodec`, would have allowed us more control over the conversion, calling FFmpeg directly meets our requirements without adding unnecessary complexity.

The parameters given to FFmpeg can be seen in Figure 2-5.

- `analyzeduration .5` Instructs FFmpeg to only examine the incoming data for half a second before it starts decoding. (note: this option is not documented [3])

```
ffmpeg -analyzeduration .5 -i - -acodec pcm_s16le -ar 8000 -f s16le -
```

Figure 2-5: These are the parameters given to the FFmpeg process. See Section 2.3.2 for an explanation.

- `-i -` indicates the incoming data is coming over the STDIN stream.
- `-acodec pcm_s16le` tells FFmpeg the output should be PCM format with unsigned 16-bit sample size little-endian.
- `-ar 8000` Sets the audio sampling rate to 8000Hz.
- `-f s16le` Sets the output format to raw PCM unsigned 16 bit little-endian. [1]
- The final `-` sets the output to be the STDOUT.

2.3.3 Posting data to WAMI

While FFmpeg is decoding Speex data to PCM that information is continuously sent over an HTTP POST connection to the WAMI record URL. All audio data is posted continuously over the same HTTP connection, allowing WAMI to start processing the information as soon as it comes in. The Wowza and WAMI servers are located on the same local network so network communication delays are minimal.

[1] The codec and the container format happen to be redundant in this case, but FFmpeg usually needs both so we set them explicitly

2.4 Default UI

WAMI provides a default microphone GUI to ensure an easy start for developers. We attempted to create a modern looking GUI that would be usable in most standard WAMI applications. There are three important states we need to represent. (See Figure 2-6)

- **Idle state** – At this point the UI is fully initialized and ready to start recording audio. To start recording, users just click once. When the mouse comes over the audio, the button lightens slightly in reaction. It appears as an un-pressed button.

- **Connecting state** – This the how the UI looks while establish connection. This is the same look when its opening up its initial connection to the Wowza server when started, and briefly during the delay between when the user clicks, and when the microphone starts sending audio data.

- **Recording state** – This is the state of the UI while audio is being recorded and being sent to the Wowza server. The microphone volume level is displayed by the raising and falling of red over the black microphone icon. The button also appears depressed.

The UI appears as a button that is depressed when clicked by the user. The microphone icon is familiar to most users that have experience with voice application. The minimal size lets developers embedded it in application unobtrusively. It will also appear in the connection

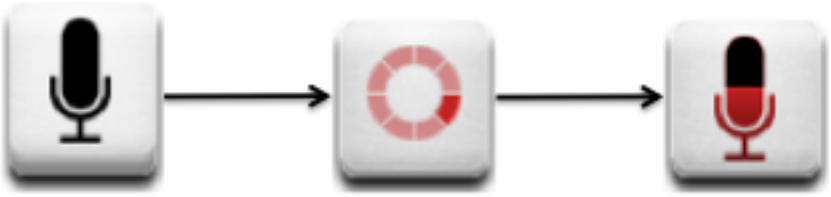

Figure 2-6: The Idle, Connecting, and Recording states of the default microphone UI. The button also lightens slight on mouse over.

state while the security dialogue is shown since it has not yet been granted permission to record audio.

Chapter 3

Applications

The improved Flash Audio Controller can be used with only minimal changes on all pre-existing WAMI systems. These changes could be even simpler, but the in process of updating the audio controller, the WAMI Javascript interface was also implemented to ease work for developers. The work to convert existing applications still remains minimal. To demonstrate this, initially the WAMI Voice TicTacToe example application was converted to use the updated controller. See a screenshot of this in Figure 3-1.

To demonstrate this, Quizlet.com was converted to use the improved Flash Audio Controller. This was a minimal upgrade requiring that we only change the javascript library reference and adjust a few configuration parameters. Some alterations had to be made to accommodate the Flash security panel. For the the improved Voice Scatter game see Figure 3-2. For the improved Voice Race game see Figure 3-3.

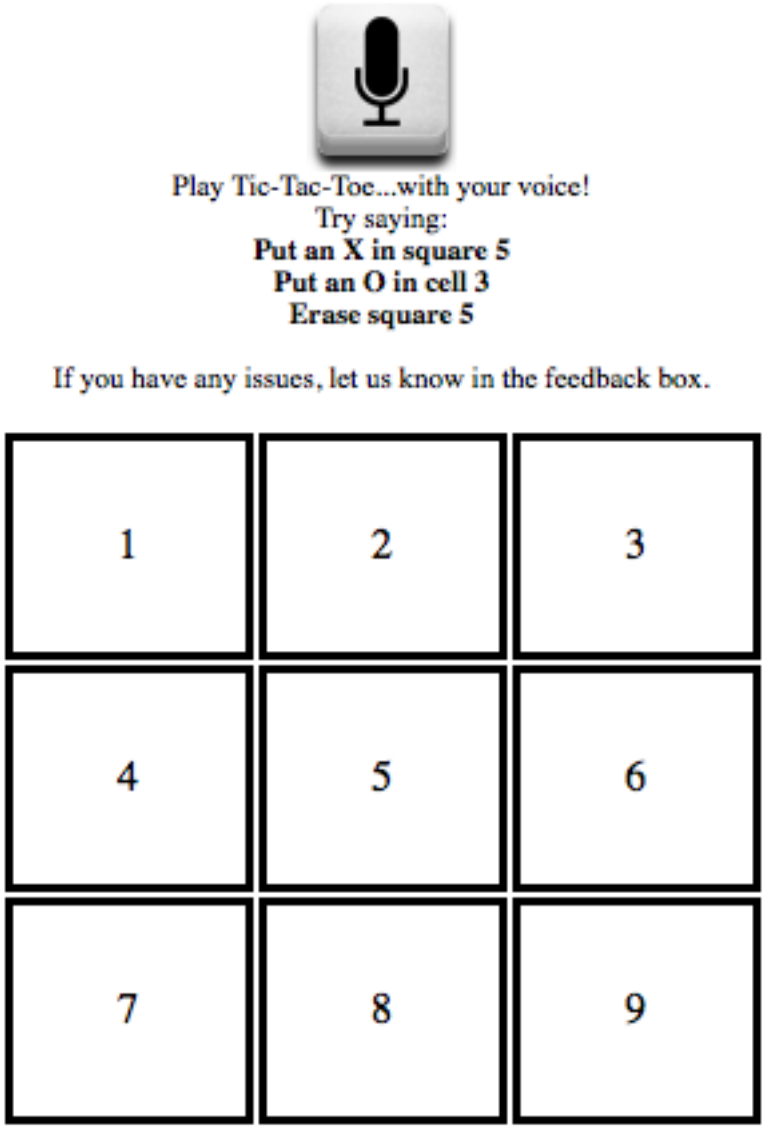

Figure 3-1: A screenshot of WAMI TicTacToe example application. Users use simple phrases to fill in the square on a TicTacToe board with their symbols.

Figure 3-2: A screenshot of Quizlet.com's Voice Scatter using the Flash Based audio controller. In this instance of the game, users use their voice to match state to their capitals

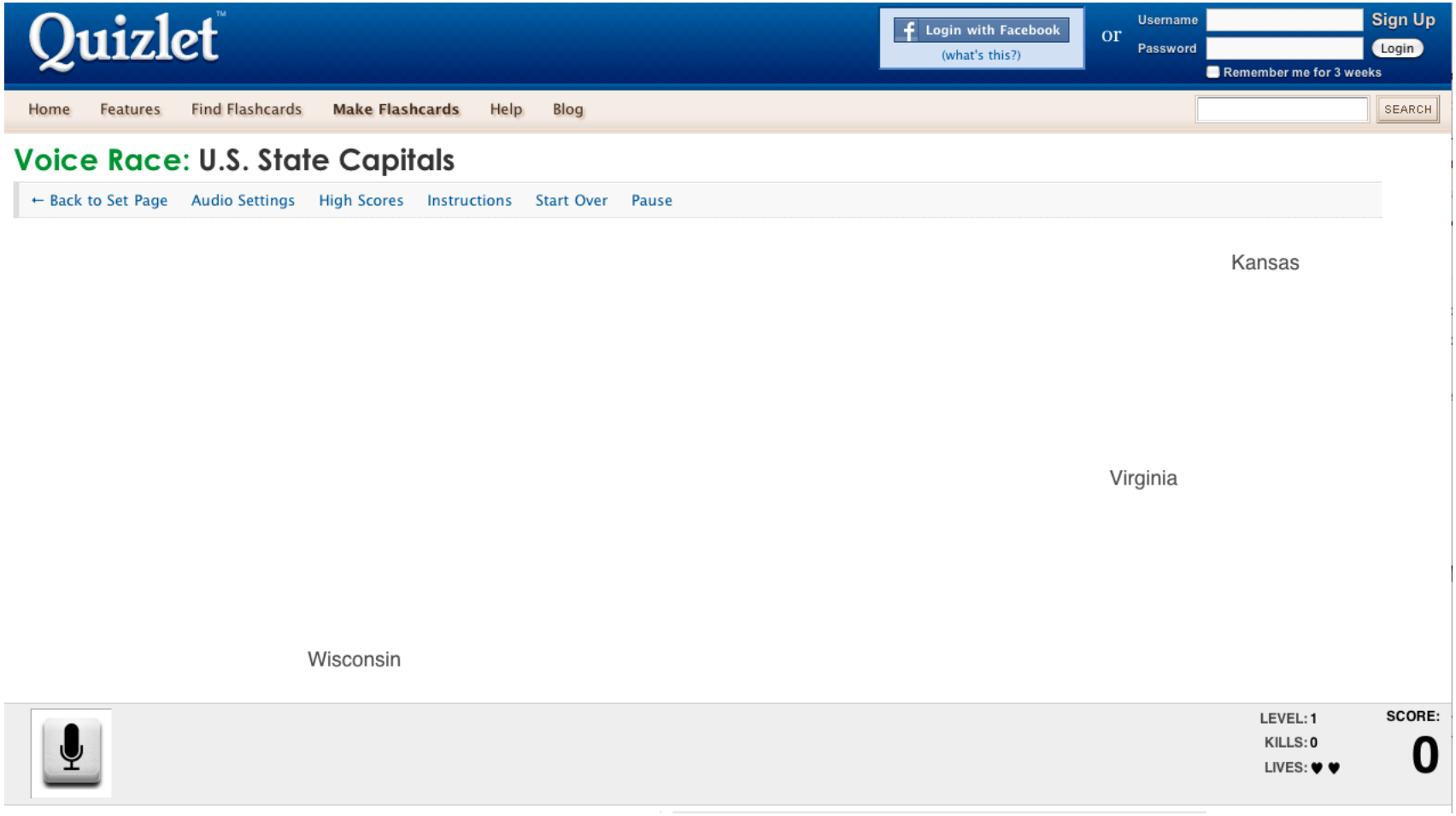

Figure 3-3: A screenshot of Quizlet.com's Voice Race using the Flash Based audio controller. In this instance of the game, users use their voice to state the capital of the state before it moves too far to the right.

Chapter 4

Experiments and Analysis

4.1 Network Performance Experiment

One of the major goals of our improved Audio Controller was to improve WAMI performance in a poor networking environment. One of the key attributes affecting users experience is the time between when a user stops recording their voice and when WAMI responds with recognition results. To demonstrate that Flash will improve this key metric, we measured the recognition response delay for a standard recognition battery in various simulated networking environments and then compared the results. The average utterance was approximately 5 seconds long.

To simulate the networking environment, `ipfw` or the FreeBSD IP packet filter was used. Using `ipfw` allows us to limit bandwidth, simulate random packet loss, network delay, and network queue. The settings for the various environments can be found in Figure 4-1. These

Connection Type	Bandwidth	Packet Loss Rate	Delay
Slow Cellular	350kbit/s	%10	350ms
Fast Cellular	800kbit/s	%3	200ms
DSL	5000kbit/s	%1	75ms
High Speed	100Mb/s	%0	0ms

Figure 4-1: `ipfw` settings used for simulating various networking environments.

settings were applied to both incoming and outgoing traffic.

Slower network environments were simulated but the Java applet Audio Controller would fail to function in these environments. The same application and recognition response instrumentation was used for both the Java applet and the Flash.

4.2 Analysis of Network Performance

As seen in Figure 4-2 our improved Audio Controller improves recognition response time in all tests, including significant response gains in poor networking environments. Even the 36% improvement in response time is significant, and reveals that a larger portion of the prior WAMI overheard was spent preparing and processing audio information on the client. This test also reveals that WAMI will continue to function adequately and degrade gracefully even in the poorest of networking environments.

Though more in-depth research can be done (see Section 5.2) we can speculate on the various causes of this improved performance. The primary cause for improvement is undoubtably the Flash Audio Controller's smaller bandwidth requirements compared to the

applet's (see Figure 4-3). However we still see a 36% improvement in the high speed networking environment, when bandwidth is not a limiting factor.

Another potential source of improvement is changing the transport protocol from HTTP to RTMP. Real Time Media Protocol (RTMP) is designed from the ground up to handle live media streams. RTMP has various features related to the transmission and reception of large streams of video and data but those are mostly irrelevant for our purposes. Compared to HTTP, RTMP has much smaller headers and can dynamically adjust fragment size as needed.[6]

Flash may also simply perform faster than Java in closing the audio transmission, enabling WAMI to send a recognition response faster.

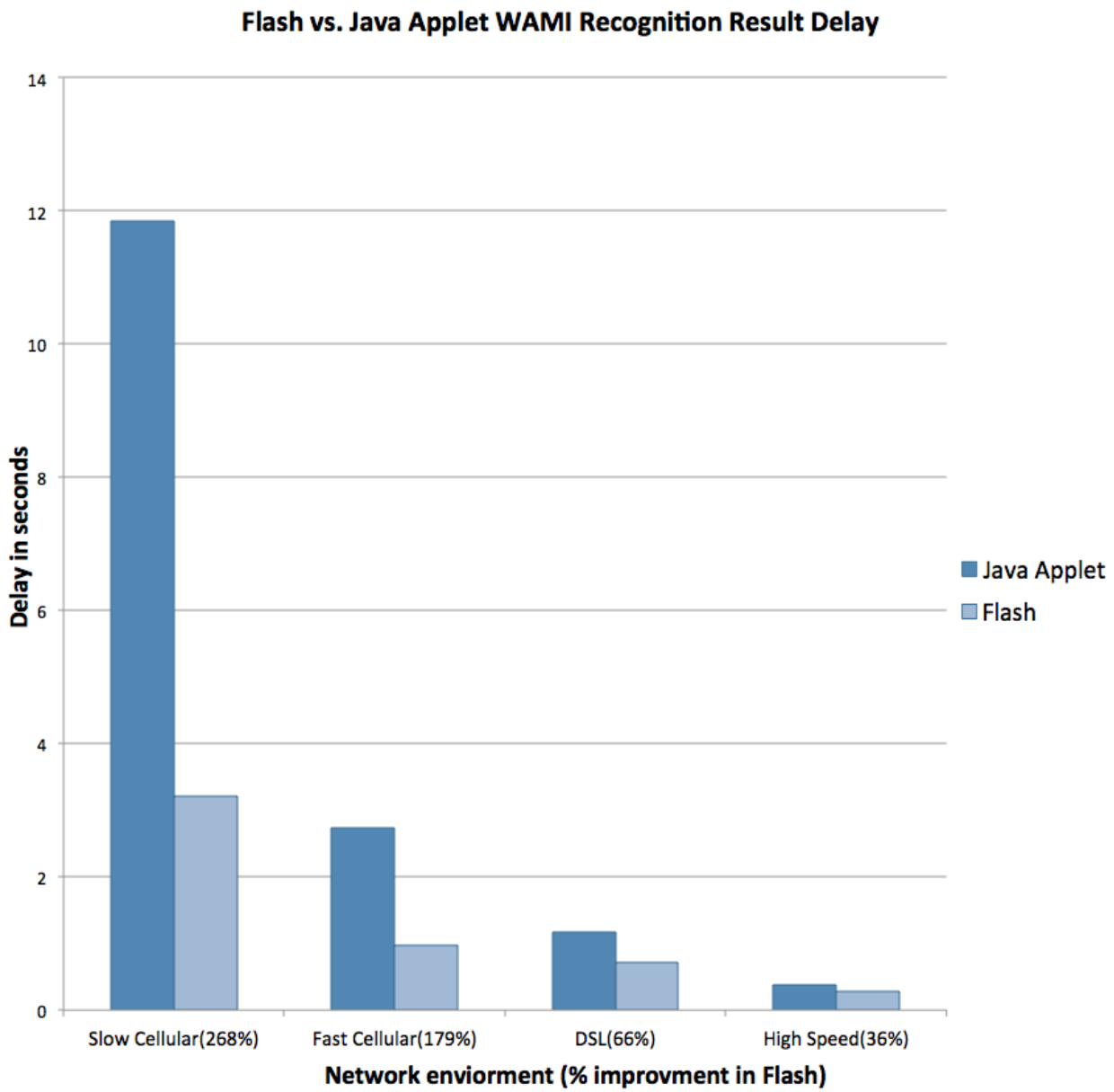

Figure 4-2: Flash vs. Java Applet performance in various networking environments. Flash shows consistent improvement.

	Bandwidth Requirement	Resource size
Java Applet Audio Controller	176 kbits/s	92.6 KB
Flash Audio Controller	64 kbits/s	6.7 KB

Figure 4-3: Network Requirements for the different Audio Controllers.

Chapter 5

Future Work and Conclusion

5.1 Further user study

Though all indicators point to our new Flash based Audio Controller improving the end users' experience, a public re-launch of the WAMI project and continued deployments will ultimately provide data to prove this is so. Over the coming months plans are being made to launch WAMI publicly, and release it on our long time partner Quizlet.com. Quizlet's larger user base will ultimately let us know if we have improved the experience for the end user.

On Quizlet, the key indicators will be the change in the number of users using speech based games, as well as the percentage of users that attempt to play a speech based game and succeed. Overall all the key indicator swill be the number of different WAMI application deployed and the amount of traffic they have. Fortunately these are all easy to measure.

5.2 Speex and Speech Recognition

In the future we will wish to better understand the relationship between Speex and recognition quality. This could potentially lead to gains in performance. At the moment we convert incoming Speex audio to 8kHz PCM, but we could just as easily convert it to 16kHz PCM provided we have models prepared. During the conversion process we're synthesizing or approximating what the audio is thought to be. The relationship between Speex's encode quality and usefulness of the converted PCM for speech recognition is also not understood. It is possible we can substantially reduce Speex's bandwidth requirements even further without a significant effect on recognition quality.

One avenue of interest is to explore Speex's variable-bit-rate encoding technique. This technique, unfortunately not yet implemented in Flash, lets Speex dynamically adjust the bit-rate to accommodate the "difficulty" of the audio. This would allow Speex to have even lower bandwidth requirements while still maintaining roughly the same quality. However, variable-bit-rate's effect on speech recognition quality is not understood. It is encouraging that all of Google's recent distributed speech recognition systems transfer audio to the server using the Speex codec.

5.3 Using Wowza and Speex across Audio Controllers

The Wowza media server setup provides a robust and scalable means of accepting live multimedia streams and transcoding them for speech recognition processing. Though Flash

necessitated this technology, it can still used by non-Flash clients.

Currently the Android and iPhone Audio Controller for WAMI send their audio directly for recognition in the raw format discussed above. This unnecessarily strains the network capacity of these mobile devices. The Wowza architecture is fully capable of accepting other Speex streams (or other audio encoding) over RTMP and sending those on for recognition results. If these clients sent Speex we'd gain performance on the devices and decrease minimum network standards.

Wowza was designed from the ground up as robust and fault tolerant media server platform. It will likely scale and grow smoother than other means of accepting audio from users. Further research and testing needs to be done to demonstrate this.

5.4 Conclusion

Our improved Flash based Audio Controller for the WAMI system provides a greatly improved usability, performance, and flexibility to the developer. Wowza provides us with a robust and reliable platform for receiving and delivering real time audio and video in future WAMI applications.

Bibliography

[1] Stack overflow discussion of detecting browser zoom level. http://stackoverflow.com/questions/1713771/how-to-detect-page-zoom-level-in-all-modern-browsers, 2010.

[2] Base 64 encoding. http://en.wikipedia.org/wiki/Base64, 2011.

[3] Ffmpeg documentation. http://www.ffmpeg.org/ffmpeg-doc.html, 2011.

[4] Ffmpeg project. http://www.ffmpeg.org, 2011.

[5] Popforge actionscript library. http://code.google.com/p/popforge/, 2011.

[6] Rtmp specification (omits important details). http://www.adobe.com/devnet/rtmp.html, 2011.

[7] Salt tags. http://en.wikipedia.org/wiki/Speech_Application_Language_Tags, 2011.

[8] W3c html speech incubator group. http://www.w3.org/2005/Incubator/htmlspeech/, 2011.

[9] Wowza media server. http://www.wowzamedia.com, 2011.

[10] Matt Oshry et. al. Voicexml 2.1. http://www.w3.org/TR/voicexml21/, 2007.

[11] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. The wami toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the 10th international conference on multimodal interfaces*, ICMI '08, pages 141–148, New York, NY, USA, 2008. ACM.

[12] Alexander Gruenstein, Ian Mcgraw, and Andrew Sutherland. A self-transcribing speech corpus: collecting continuous speech with an online educational game. In *the Speech and Language Technology in Education (SLaTE) Workshop*, 2009.

[13] Robert Hansen and Jeremiah Grossman. Clickjacking. http://www.sectheory.com/clickjacking.htm, 2008.

[14] Kuansan Wang. SALT: a spoken language interface for web-based multimodal dialog systems. In *Proceedings of ICSLP—Interspeech 2002: 7th International Conference on Spoken Language Processing, Denver, CO, USA*, pages 2241–2244, 2002.