

EURECOM INSTITUTE
MULTIMEDIA AND COMMUNICATION DEPARTMENT

PHAN THÀNH TRUNG

**GRAB YOUR FAVORITE TV SHOW
AND SHARE IT**

SEMESTER PROJECT

BIOT, 2013

**EURECOM INSTITUTE
MULTIMEDIA AND COMMUNICATION DEPARTMENT**

PHAN THÀNH TRUNG

**GRAB YOUR FAVORITE TV SHOW
AND SHARE IT**

SEMESTER PROJECT

SUPERVISORS

Assistant Prof. Raphaël Troncy

Ph.D. Student José Luis Redondo García

Post Doc Giuseppe Rizzo

FOREWORD

I would like to thank the Faculty of Multimedia and Communication, Eurecom Institute has created favorable conditions for me to learn and implement this semester project.

I would like to express my deep gratitude to Assistant Prof. Raphael Troncy, Ph.D. student Jose Luis Redondo Garcia and Post-doc Giuseppe Rizzo. You all give me the hearted guide during the process of implementation of the project.

I would like to thank teachers in the Faculty of Multimedia and Communication, Eurecom Institute has dedicated teaching and equipping me with valuable knowledge of the past school year.

I sincerely thank my family always encourage and support physical and mental over time.

Despite trying to complete this semester project within the scope and possible ability, but certainly not free of shortcomings. I look forward to understanding, suggestions and take the advice of teachers and you all.

Biot, February 06, 2013

Master Student

PHAN Thành Trung

TABLE OF CONTENT

Chapter 1 Introduction.....	2
1.1 Context and objective	2
1.2 Problem and solution	3
1.3 Results of this semester project	3
1.4 General structure of this report.....	4
Chapter 2 Kinect device and Kinesis SDK	5
2.1 Introduction to Microsoft Kinect.....	5
2.1.1 What is Kinect	5
2.1.2 Developing on Kinect Device.....	5
2.1.3 Kinect SDK Architecture.....	6
2.1.4 Programming on Kinect for Microsoft Windows.....	7
2.2 Introduction to Kinesis.IO and Kinesis SDK	10
2.2.1 What is Kinesis.IO	10
2.2.2 Kinesis JavaScript SDK.....	11
2.2.3 Developing with Kinesis.IO SDK	11
2.2.4 Programming on Kinect device (API) with Kinesis SDK.....	13
2.2.5 Potential with Kinesis domains	15
Chapter 3 Introduction to WebSocket and Node.js.....	17
3.1 Introduction to WebRTC.....	17
3.1.1 What is WebRTC.....	17
3.1.2 WebRTC architecture	17
3.1.3 Key feature of WebRTC	18
3.1.4 Data channel of WebRTC.....	19

3.2 Introduction to WebSocket.....	19
3.2.1 What is WebSocket.....	19
3.2.2 Websocket architecture.....	20
3.2.3 WebSocket protocol.....	20
3.2.4 Supported web browsers with WebSocket	25
3.2.5 HTML5 Websocket API.....	25
3.2.6 Potential WebSocket Use case.....	25
3.3 BinaryJS	26
3.3.1 What is BinaryJS	26
3.3.2 BinaryJS benefits	26
3.4 Node JS and its potential	27
3.4.1 What is Node.js.....	27
3.4.2 Developing on Node.js	27
3.4.3 Making programming on Node.js.....	28
3.4.4 Potential of Node.js	28
Chapter 4 Application architecture, main features and experiments.....	30
4.1 System description.....	30
4.2 System design.....	31
4.2.1 System architecture.....	31
4.2.2 Sequence Diagram	32
4.2.3 Rendering Source Server	33
4.2.4 Broadcast controller.....	33
4.2.5 Client.....	36
4.2.6 Websocket Server	37

4.3 Install and result	37
4.3.1 Install	37
4.3.2 Experiments	38
Chapter 5 Conclusion	40
5.1 The obtained result	40
5.2 Development in future:.....	40
5.3 Personal opinion: what did I learn? Has it been interesting?	41
REFERENCES	42

LIST OF FIGURES

- Figure 1 Kinect Device [<http://en.wikipedia.org/wiki/File:Xbox-360-Kinect-Standalone.png>]
- Figure 2 Sensors on Kinect [<http://blogs.msdn.com/b/kinectforwindows/>]
- Figure 2 Sensors on Kinect [<http://blogs.msdn.com/b/kinectforwindows/>]
- Figure 3 Kinect Architecture Application [<http://msdn.microsoft.com/en-us/library/jj131023.aspx>]
- Figure 4 Kinect SDK Architecture [<http://msdn.microsoft.com/en-us/library/jj131023.aspx>]
- Figure 5 Data Streams
- Figure 6 Skeletal Tracking with 6 people [<http://msdn.microsoft.com/dynimg/IC584841.png>]
- Figure 7 Tracking standing or sitting [<http://msdn.microsoft.com/dynimg/IC584441.png>]
- Figure 8 Skeletal of human [<http://msdn.microsoft.com/dynimg/IC584844.png>]
- Figure 9 Kinesis.IO logo [<http://kinesis.io/>]
- Figure 10 Kinesis SDK in the general Kinect Architecture application
- Figure 11 Piece of initial code of Kinesis [3]
- Figure 12 Screen is supported with Kinesis
- Figure 13 Hello world application
- Figure 14 Cursor tracking
- Figure 15 Result from simulate mouse click [3]
- Figure 16 WebRTC Architecture [<http://www.webrtc.org/reference/architecture>]
- Figure 17 Media stream [<http://dev.w3.org/2011/webrtc/editor/images/media-stream.png>]
- Figure 18 Peer connection illustration [<http://www.html5rocks.com/en/tutorials/webrtc/basics/apprtcArchitecture.png>]
- Figure 19 Websocket logo [5]
- Figure 20 A basic websocket-based architecture [<http://www.websocket.org/img/websocket-architecture.jpg>]
- Figure 21 HTTP Polling [<http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpPolling.gif>]
- Figure 22 HTTP Long Polling [<http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpLongPolling.gif>]

Figure 23 HTTP Streaming
[\[http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpStreaming.gif\]](http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpStreaming.gif)

Figure 24 The complexity of Comet applications
[\[http://www.websocket.org/img/comet-apps.jpg\]](http://www.websocket.org/img/comet-apps.jpg)

Figure 25 Comparison between Polling and Web sockets
[\[http://www.websocket.org/img/latency-comparison.gif\]](http://www.websocket.org/img/latency-comparison.gif)

Figure 26 Websocket protocol illustration [\[http://web2.sys-con.com/node/2184408\]](http://web2.sys-con.com/node/2184408)

Figure 27 Node.js Logo [4]

Figure 28 System architecture

Figure 29 Sequence diagram of project application

Figure 30 Broadcast controller UI

Figure 31 Client UI

LIST OF TABLE

Table 1List of supported languages

Table 2Hardware and software requirements

Table 3List of supported web browsers with Websockets

Table 4Websocket use case

Table 5Table of latency between client and controller

Chapter 1 Introduction

✎ In this chapter, we first provide the objectives of this semester project. The problems are figured out and we briefly describe the results we obtained. We provide then the structure of this report.

1.1 Context and objective

The way the user watches television (TV) is changing. Let's suppose you're sitting on your couch and you're watching television. If something interesting comes on and you want to share it with your friends, you can just tweet about it or take a photo of the screen and upload it. But instead, imagine you reach out towards your TV with your arm and "grab" what's on the screen by making a interact with the programs you love and are beginning to expect more than a one-directional, passive viewing experience. Various research projects such as LinkedTV [1] are currently focused on these interesting challenges gesture. You're holding the image in your hand. To transfer it to your phone, simply tap the screen. Magically, the picture is now on your phone so you can share it with your friends in some of your favorite Social Networks.

That is our first scenario for this project. Now, we promote the project to the higher scenario and wider technique. Let's suppose you're a video broadcast controller. You broadcast video channels to all your clients which are concurrently connecting to the server. You control by using a Microsoft Kinect [2] with your hand gesture: you can choose which channel to broadcast on the TV screen and after that you swipe left to broadcast, show your hand opening widely as a sign of pause to pause. In this circumstance, we can use TV as a screen, Microsoft Kinect is used as a tool realizes our hand gesture for playing or pausing broadcasting channel.

The objective of this project is develop a controller application that is able to detect these kinds of hand gestures and to broadcast video channels and the normal playback commands to all clients which are connecting to this controller. The main objectives focus on these parts:

- Building the broadcast controller web application in which Microsoft Kinect is connected as the gesture receiver can get the users' hand gestures which play as the commands.
- Building the client web application which receives the video channel and play video in synchronization with the broadcast controller.
- Build a server application to process and transfer information between the broadcast controller and all clients.

1.2 Problem and solution

Kinect for Windows consists of the Kinect for Windows sensor, the Kinect for Windows software development kit (SDK), and the commercial licensing necessary for application deployment. The Kinect for Windows SDK supports applications built with C++, C#, or Visual Basic using Microsoft Visual Studio 2010 or 2012. In this project, we use Kinect for web browser and we need a framework for communicating between the Kinect device and a web browser. Microsoft does not provide a framework for Kinect on web browser. Kinesis.IO [3] is a native framework for developing gesture based web/desktop apps using JavaScript, HTML and CSS. This framework is written by a group of Indian developers and it helps us a lot in this situation.

We need to write a broadcast controller and clients which are communicating each other through the server. In this situation, we use the Node.js [4] as the basis for coding our server. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. In the meanwhile, websocket [5] is the solution for coding clients and broadcast controller. Websocket defines a full-duplex single socket connection over which messages can be sent between client and server. The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management.

When the broadcast controller broadcasts video channel to all clients, we will face potential synchronization problems. We solve it by communicating the current playback timestamp at regular interval in broadcast controller and threshold in clients to sync the playing time.

1.3 Results of this semester project

We have built the above applications mentioned above:

- A HTML Generating Source Server: A server generates the HTML source code for the web pages of the broadcast controller and all clients.
- A Websocket server: A server will process and transfer information from broadcast controller to all clients and vice versa.
- The broadcast controller is commanded by the Kinect. Users can use hand gesture to control: play, pause, or change the video channel to all clients at the same time.

- The client gets commands from the broadcast controller and plays in synchronization with the controller.

1.4 General structure of this report

The content of this report includes the following chapters:

Chapter 1. It is introduced above.

Chapter 2. Microsoft Kinect and Kinesis SDK: presents the Kinect device and the Kinesis SDK - the recent platform for making program connecting the Kinect to a web browser.

Chapter 3. Introduction to WebSocket and Node.js: presents the WebSocket technology. Then we give an introduction to Node.js as well as its benefits when building the server application.

Chapter 4. Application architecture, main features and experiment: In this chapter, we will describe our contribution for this semester project. We will detail the system design including the system architecture, system sequence diagram, and description of each relevant servers and clients. After that, we will show the interface of the broadcast controller as well as the client interface. Finally, we will show the results obtained.

Chapter 5. Conclusion: This chapter concludes this report and outlines future work.

Chapter 2 Kinect device and Kinesis SDK

✎ In this chapter we will make an introduction to the Kinect device. Also, Kinesis SDK will be shown as an alternative to develop Web-Based applications that use Kinect.

2.1 Introduction to Microsoft Kinect

2.1.1 What is Kinect

Kinect [6] is a motion sensing input device which enables users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken commands



Figure 1 Kinect Device [<http://en.wikipedia.org/wiki/File:Xbox-360-Kinect-Standalone.png>]

2.1.2 Developing on Kinect Device

To develop on Kinect device, we should have the Kinect for Windows SDK [7] which supports applications built with C++, C#, or Visual Basic using Microsoft Visual Studio 2010 or 2012.

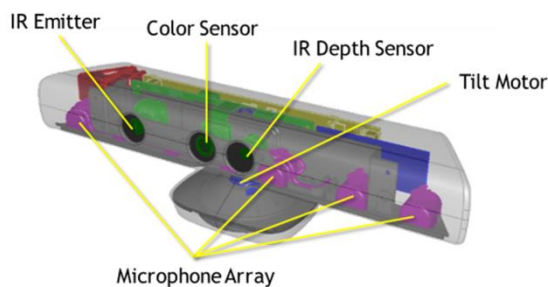


Figure 2 Sensors on Kinect [<http://blogs.msdn.com/b/kinectforwindows/>]

In normal, if we want to develop applications by using the Kinect for Windows SDK, we should pass two steps [8]:

1. **Setting up Kinect for Windows SDK:** The SDK includes drivers for using the Kinect for Windows sensor on Windows and APIs and device interfaces.
2. **Setting up Kinect Windows Developer Toolkit:** The toolkit contains updated source code samples, Kinect Studio, Face Tracking SDK, and other resources. This developers can get familiar with developing applications by using the Kinect for Windows SDK

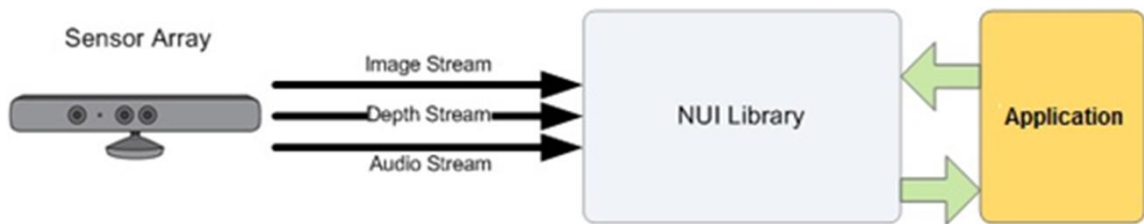


Figure 3 Kinect Architecture Application [<http://msdn.microsoft.com/en-us/library/jj131023.aspx>]

In the next part, we will give the general introduction to the Kinect SDK Architecture.

2.1.3 Kinect SDK Architecture

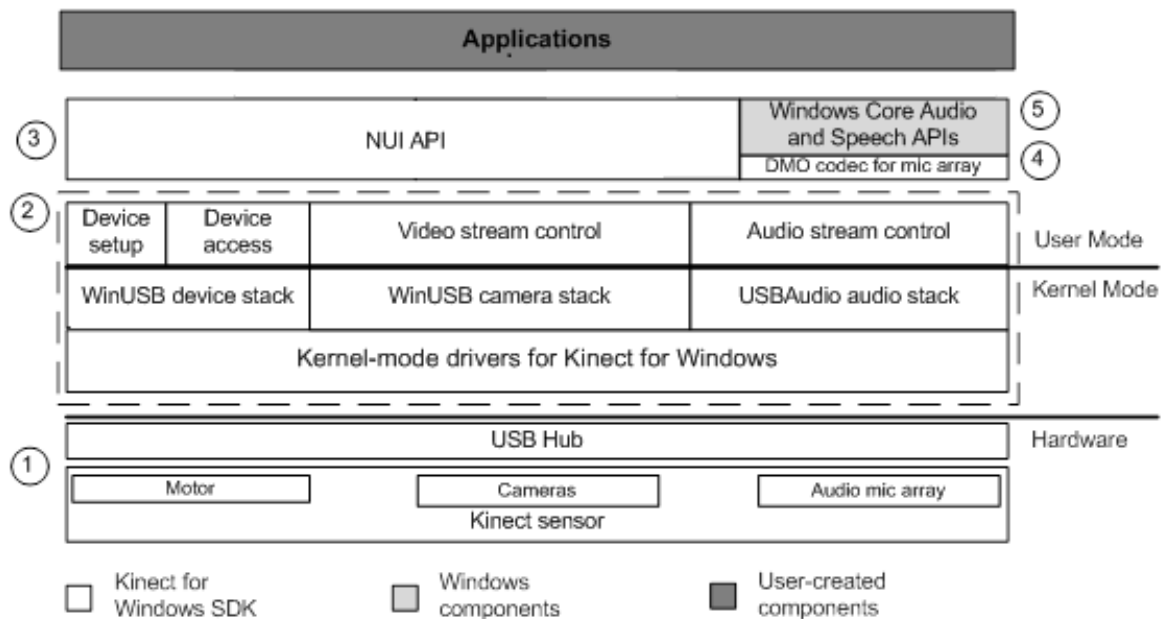


Figure 4 Kinect SDK Architecture [<http://msdn.microsoft.com/en-us/library/jj131023.aspx>]

These components include the following:

1. Kinect hardware: Kinect sensor and the USB hub connecting the sensor to the computer.

2. Kinect drivers. The Kinect drivers manage the audio and video streaming controls for streaming audio and video (color, depth, and skeleton).
3. Audio and Video Components
 - Kinect natural user interface for skeleton tracking, audio, and color and depth imaging
4. DirectX Media Object (DMO) for microphone array beam forming and audio source localization.
5. The audio, speech, and media APIs in Windows 7.

We will move on the issue of making programming on Kinect SDK to know the general picture of developing this semester project with Kinect device.

2.1.4 Programming on Kinect for Microsoft Windows

Windows Software Development Kit (SDK) provides the tools and APIs to develop Kinect-enabled applications for Microsoft Windows.

The SDK supports the functions (APIs) or interface to process and the sample codes of application with:

1. *Data streams:*

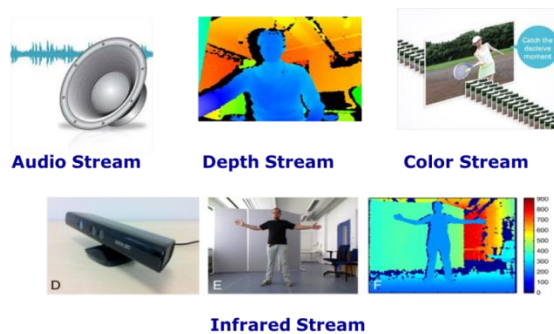


Figure 5 Data Streams

2. *Skeletal Tracking:*

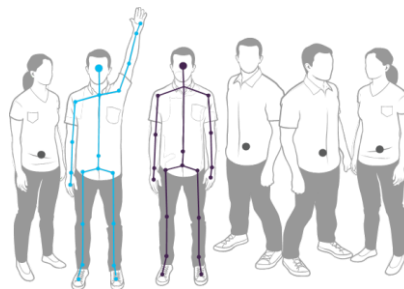


Figure 6 Skeletal Tracking with 6 people[<http://msdn.microsoft.com/dynimg/IC584841.png>]

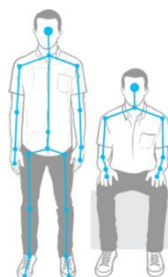


Figure 7 Tracking standing or sitting[<http://msdn.microsoft.com/dynimg/IC584441.png>]

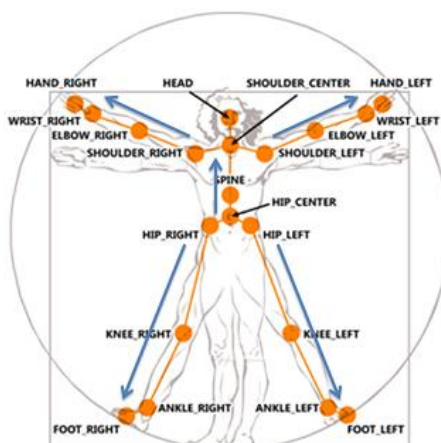






Figure 8 Skeletal of human[<http://msdn.microsoft.com/dynimg/IC584844.png>]

3. *Speech:*

Kinect Device APIs support the list of the following languages:

Language	Country	Language	Country
de-DE		en-AU	
en-CA		en-GB	
en-IE		en-NZ	
es-ES		es-MX	



fr-CA		fr-FR	
it-IT		ja-JP	

Table 1List of supported languages

Based on the above information, if we want to develop application on Kinect, we could use the supported languages (C++, C# or VB) to develop with the Kinect SDK. In this time, the “power of Kinect” cannot stop at the purpose of playing games so that numerous developers are researching possible applications of Kinect. The development third party has been trying many ways to develop their own platform to bridge the existing Kinect SDK and Kinect device to their aimed application. Right now, Kinesis.IO SDK is a kind of that purpose. In the next part, we will give the detailed information about the Kinesis SDK and its potential strength.

On Linux, we can also develop application with Kinect device but we must use the OpenKinect as the third party. OpenKinect [13] is a free and open source libraries to help developers to make program with Kinect on Windows, Linux, and Mac.

Moreover, we can have many references about many kinds of gestures from communities of developers. For example, Kinect Hand process [14] is an example. Based on the Kinect Windows SDK, it gives us the sample codes of detecting the fingers gestures which enables complex gestures such as the grab one? In addition, there are also Kinect gesture Library [15], Air Kinect gesture [16], Qt Air Cursor [17], and Xkin [18] which are some ready libraries supporting the complex gesture. Most of them use the OpenCV, OpenNI and written on C++ or C#.

For example, Air Kinect Gestures [16] supports these gestures and movements:

- Gestures :
 - GestureDirection.SWIPE_HORIZONTAL;
 - GestureDirection.SWIPE_RIGHT;
 - GestureDirection.SWIPE_LEFT;
 - GestureDirection.SWIPE_VERTICAL;
 - GestureDirection.SWIPE_UP;
 - GestureDirection.SWIPE_DOWN;
 - GestureDirection.SWIPE_DEPTH;

- GestureDirection.SWIPE_FORWARD;
- GestureDirection.SWIPE_BACKWARD;
- Movements :
 - JumpMovement;
 - BendDownMovement;

The way of watching television and controlling the television are changed with the exist of Kinect device. Kinect make users be free from remote control with few hand gesture or speech command. So the Kinect device is very useful in a television scenario.

2.2 Introduction to Kinesis.IO and Kinesis SDK

2.2.1 What is Kinesis.IO

In simplest words, Kinesis.IO helps us to build gesture driven web apps with HTML, JS & Kinect.



Figure 9 Kinesis.IO logo[<http://kinesis.io/>]

Kinesis.IO transfers raw data from Kinect device to actionable data. In addition, Kinesis adds gestures, speech or depth data in minutes for the Kinect powered web / desktop application.

- ***Gesture***: Skeleton tracking, hand movement, hand swipes, head tracking, jump, crouch and more are just a function call away.
- ***Speech***: Speech recognition based actionable keywords plus skeleton actions complete the interaction package.
- ***3D Depth modeling***: Dabble with building 3D models of objects or augmented reality apps.

Kinesis.IO leverages technologies web developers already know best HTML/CSS/JavaScript. So the developers could transfer their existing web applications into the Kinect-powered web application easily. Kinesis.IO is similar to the Phonegap. Instead of using the Kinect SDK to build windows application, we use Kinesis.IO SDK to build Kinect-powered web application

2.2.2 Kinesis JavaScript SDK

Kinesis.IO is a native framework that lets us to reuse our existing web languages and skills to build gesture, speech and 3D depth modeling based applications using Kinect. We focus on building a simple, powerful, new interaction platform so that we can focus on bringing our applications to life

JavaScript development framework gives access to Gesture Recognition like hand swipe, multiple joint tracking, jumping, body movement; and Speech Recognition to provide actionable keywords.

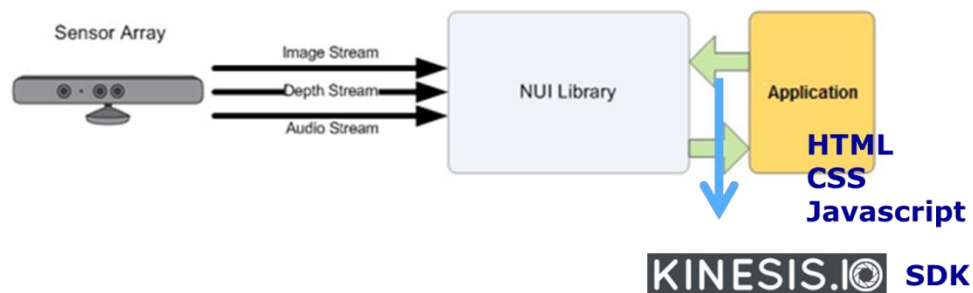


Figure 10 Kinesis SDK in the general Kinect Architecture application

Kinesis JavaScript SDK has these following characteristics:

- Take advantage of web standards, HTML5 and CSS3
- Use JavaScript to write code or any library we prefer
- Web applications. Supports new versions of popular browsers but it doesn't support the old versions of some browsers.
- Easy sample code of UI elements for quick development cycle
- Gives access to native features - Camera, motor etc.

Based on the above information, we can see that Kinesis SDK is simple, powerful and free. It's total free with no hidden cost or contract. Even we don't have Kinect device, Kinesis SDK can support the Kinect simulator for developers when they develop Kinect-powered web application. Moreover, Kinesis also provides the UI elements which help us to make a rapid prototyping easily and conveniently.

In the next part, we will explain how to develop Web applications using Kinesis SDK

2.2.3 Developing with Kinesis.IO SDK

Before making programming with Kinesis.IO SDK, we should:

- Download the Microsoft Kinect SDK for windows and install it on our computer.
- Then download Kinesis.IO SDK and run installer on our computer. Also we can download the whole Kinesis SDK package with ready-Microsoft Kinect SDK for Windows and install it.
- After that, if we don't have Kinect device, we can use the simulator by installing the Kinesis simulator which is supported by the Kinesis.IO. With the available sample demo, we can take an idea of what is possible to do when we start to use Kinesis.IO.

We also pay attention to the hardware and software requirements:

Hardware Requirements	Software Requirements
1. Windows 7 enabled PC. 2. Kinect for Windows or Microsoft XBOX 360 Kinect	1. Windows 7 or above. 2. Kinesis SDK

Table 2Hardware and software requirements

This is the piece of code for the Hello world example. This initial tiny code allows our Web application to have a pointer on screen that can be moved by using our hand in front of the Kinect.

```

Initialize! Few lines of code
1  <head>
2    <!-- kinesis stylesheet -->
3    <link rel="stylesheet" type="text/css"
4    href="/lib/css/kinesis.css">
5  </head>
6  <body>
7    <!-- kinesis js sdk -->
8    <script src="/lib/js/kinesis-js-sdk.min.js">
9    </script>
10   <script>
11     // initialize kinesis
12     var kinesis = new Kinesis;
13     // start adding gestures from here //
14   </script>
15 </body>

```

Figure 11Piece of initial code of Kinesis [3]

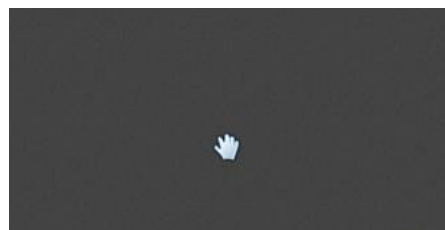


Figure 12Screen is supported with Kinesis

2.2.4 Programming on Kinect device (API) with Kinesis SDK

Initialization:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- kinesis stylesheet -->
    <link rel="stylesheet" type="text/css" href="http://cdn.kinesis.io/kinesis.css">
  </head>
  <body>
    <!-- kinesis js sdk -->
    <script src="http://cdn.kinesis.io/kinesis-js-sdk.min.js"></script>
    <script>
      var kinesis = new Kinesis; // initialize kinesis
      // start adding gestures from here //
    </script>
  </body>
</html>
```

Figure 13Hello world application

Cursor

tracking:



Call to get the current cursor position

```
<script src="http://cdn.kinesis.io/kinesis-js-sdk.min.js"></script>
<script type="text/javascript">
  function movement(position)
  {
    console.info(position.x);
    console.info(position.y);
    console.info(position.z);
  }
  kinesis = new Kinesis;
  Kinesis.cursor = movement;
</script>
```

Figure 14Cursor tracking

Gestures: Gesture Recognizer – Defining all events which Kinesis JS SDK listens and can trigger custom events. This acts as the base class of all gesture classes. We make instance variables, customize the gesture and then start recognizing them.

Create swipe gesture: “mySwipeGesture” is any name given to the variable for our reference.

```
var swipeGesture = new SwipeGestureListener("mySwipeGesture");
```

Callback: Call to action for swipe control

```
swipeGesture.toFire = swipeControl; //swipeControl is a callback method
```

Direction: Allowed direction for gesture

```
swipeGesture.directions = [GestureDirections.GestureDirectionLeft];
```

The supported directions are

- GestureDirectionLeft
- GestureDirectionRight
- GestureDirectionUp
- GestureDirectionDown

Joint:

- This is used to set the joints to be tracked for gesture

```
swipeGesture.joints = [JointTypes.JointTypeHandRight];
```

- This is an array of joints, so if we want to track a swipe gesture with both left and right hands

```
swipeGesture.joints = [JointTypes.JointTypeHandRight, JointTypes.JointTypeHandLeft];
```

- The joints which are currently used for Swipes are
 - JointTypeHandRight
 - JointTypeHandLeft

Bounds (Optional): This is required to specify the area where gesture tracking should be enabled. Only if the origin of a gesture is in the specified bounds, the gesture will be recognized. Values in percentage (%)

```
swipeGesture.bounds = {min: {x: 80, y: 0, z: 0}};
```

Simulate Mouse Click Via Hold Gesture: Just add ‘*interactive*’ class to an element. When we hold our hand over any element with ‘*interactive*’ class, the ‘*onclick*’ event for its parent is fired. We should have prerequisites that we have to include the Kinesis CSS, Kinesis JS and initialized kinesis as outlines in the webpage.

```
<div id="content">
```

```

<a class="grouped_elements" rel="group1" href="images/1_b.jpg">
  
</a>
<a class="grouped_elements" rel="group1" href="images/2_b.jpg">
  
</a>
<a class="grouped_elements" rel="group1" href="images/5_b.jpg">
  
</a>
<a class="grouped_elements" rel="group1" href="images/4_b.jpg">
  
</a>
</div>

```

And the result likes the following figure 16. Whenever we hold our hand over any image, we can see a feedback circle loading and consequently the “onclick” for the link is fired and we see an image pop-up.



Figure 15 Result from simulate mouse click [3]

In the next part, we will discover the potential of domains when we use Kinesis to develop web applications.

2.2.5 Potential with Kinesis domains

Make any web application with gesture-command: Make a gesture driven web application for a site in minutes. All we need is a line of JavaScript along with the Kinesis SDK. Building the Kinect-powered web application is the trend of

replacing using big screens with touch interactions. This trend of interaction will be popular in future.

Gaming: Kinect has revolutionized gaming. For example, playing King Chess...

Augmented reality apps with 3D Depth Data: building 3D objects or as navigation module for robots.

Speech: using actionable keyword (supported APIs) to recognize speech.

TV: Feel free from remote commander and no more of “couch potato”-staying on sofa and don’t have any interaction with television..

Nowadays, there are some applications [19] built on top of kinesis.io such as:

- The Tweet Show: Browse twitter with gestures. Use hand swipes to interact with tweets, hold on images to open them in a lightbox.
- Mix Flicks - Movies catalogue: Movies catalogue concept, which can be displayed inside a movie rental store to drive engagement and personalization. Hand swipes along with speech forms the basis of engagement.
- Google map: Looking at maps on small screens is difficult and big screens lack proper interactions to provide a uniform experience. Use gesturs to browse maps.
- Google Street view: Stop scrolling through streets, start walking through them. The Google Street view lets you do just that, use your hands to roam around streets.
- Instagram: Browse popular images from instagram

Based on the above potentials, making any site with gesture, user interaction and TV are familiar with our objectives mentioned in the chapter 1. We would like to build the site which use the Kinect device as the commander with the command gesture (play, pause or load video channel).

In this chapter, we give the general introduction to Kinect device and Kinesis SDK supporting developer to make application connecting Kinect device to the web browser as well as it potential domains. In the next chapter, we will give introduction to the web technologies: WebRTC, WebSocket and Node.js. These things play as the second main part in this semester project.

Chapter 3 Introduction to WebSocket and Node.js

✎ In this chapter, we will give an introduction to the WebRTC. Then we will mention Websocket as the technology we will use in this project. After that, we will explain how BinaryJS transfers binary data between client and server. Finally, we will see how Node.js allows us to create the main application server for this semester project application.

3.1 Introduction to WebRTC

3.1.1 What is WebRTC

- **Real Time Communication** meets the web.
- The **state-of-the-art** audio/video communication stack in your web browser.
- A cross-industry effort to create **a new communication platform**

We can use WebRTC to build web application with JavaScript and HTML5. This application can get the media data from computer device (e.g. voice from microphone, or video stream from camera), process it and show it on the web browser or even send the processed output to its peer through peer connection.

3.1.2 WebRTC architecture

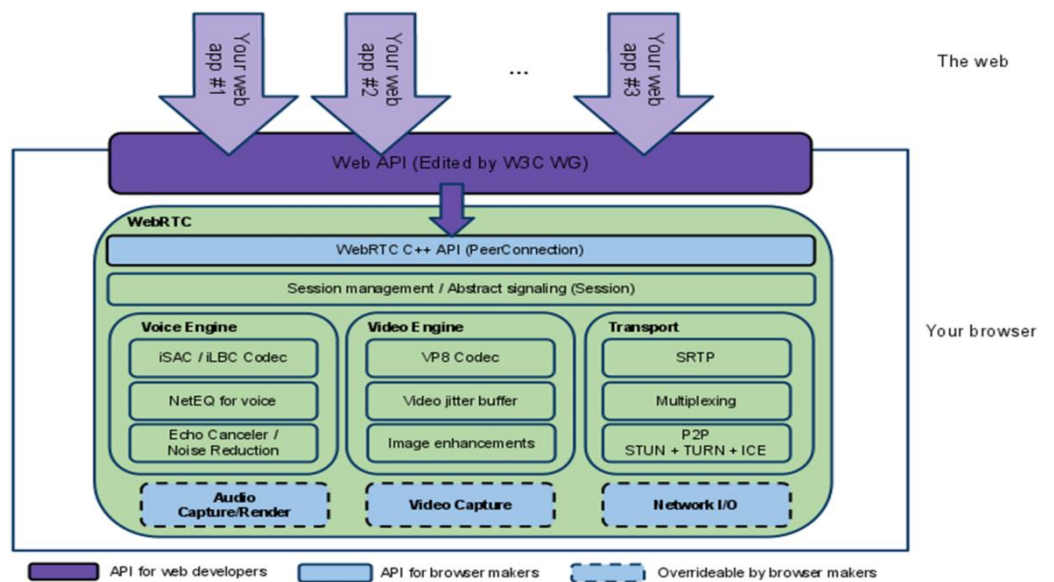


Figure 16 WebRTC Architecture [<http://www.webrtc.org/reference/architecture>]

Based on the above architecture, the developer can use Web API to reach the media data such as voice, or video from the video capture device, audio capture device and send it to peer through the Network I/O in the current computer.

3.1.3 Key feature of WebRTC

WebRTC as implemented uses the following APIs.

- **MediaStream**: get access to data streams, such as from the user's camera and microphone.

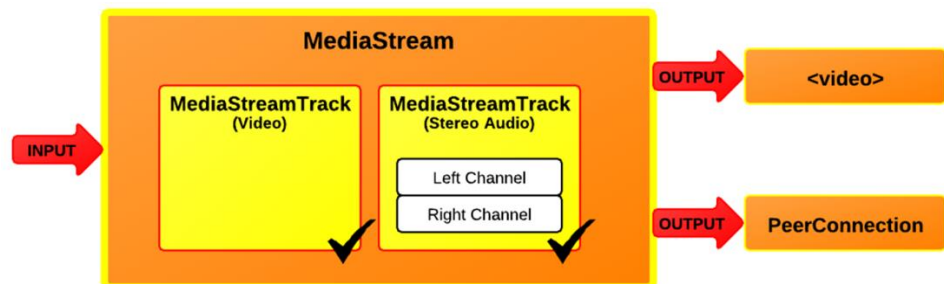


Figure 17 Media stream [<http://dev.w3.org/2011/webrtc/editor/images/media-stream.png>]

- We can get video stream and show on the website or send it through peer connection.
- **PeerConnection**: audio or video calling, with facilities for encryption and bandwidth

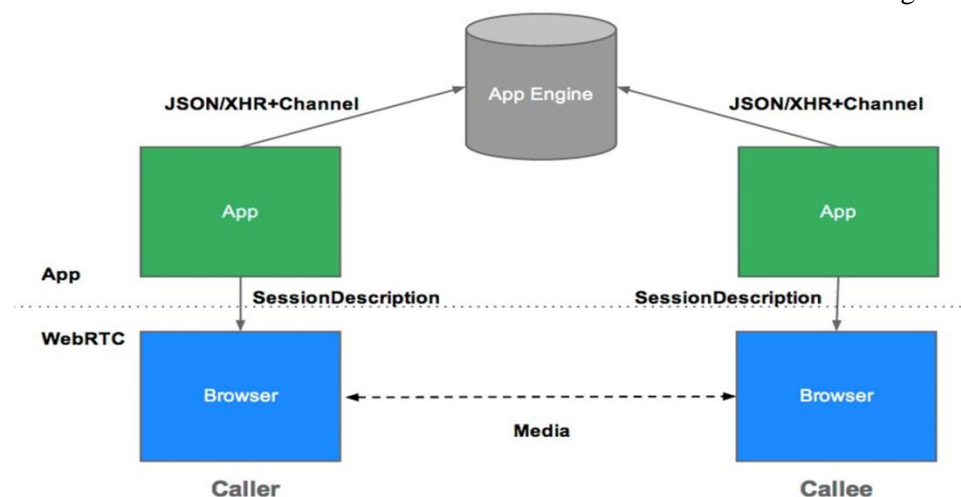


Figure 18 Peer connection illustration
[<http://www.html5rocks.com/en/tutorials/webrtc/basics/apprtcArchitecture.png>]

In peer connection, WebRTC needs servers, however it is very simple, so the

following can happen: Users discover each other → Users send their details to each other → Users communicate data together through network.

3.1.4 Data channel of WebRTC

Data Channel: peer-to-peer communication of generic data. Thanks to the peer connection, we can set up the data channel among the users such as: gaming, remote data, file transfer, video conference ...

In the scope of this semester project, our controller and clients are playing the same video which has the remote source link. So, the problem of sync between all clients and controller can be solved by send the time stamp from controller to clients. We don't need to send the video data from controller to client. This is very resource consuming and leads to network congestions. So WebRTC is not a solution for our semester project requirements.

3.2 Introduction to WebSocket

3.2.1 What is WebSocket

The WebSocket [5] is a JavaScript interface, which defines a full-duplex single socket connection over which messages can be sent between client and server. The websocket is used on the HTML5 environment. In addition, the WebSocket standard reduces the complexity around bi-directional web communication and connection management.



Figure 19 Websocket logo [5]

WebSocket represents the next evolutionary step in web communication compared to Comet and Ajax. They have their own advantages and disadvantages. We should know their technologies well so we can make the right choice.

3.2.2 WebSocket architecture

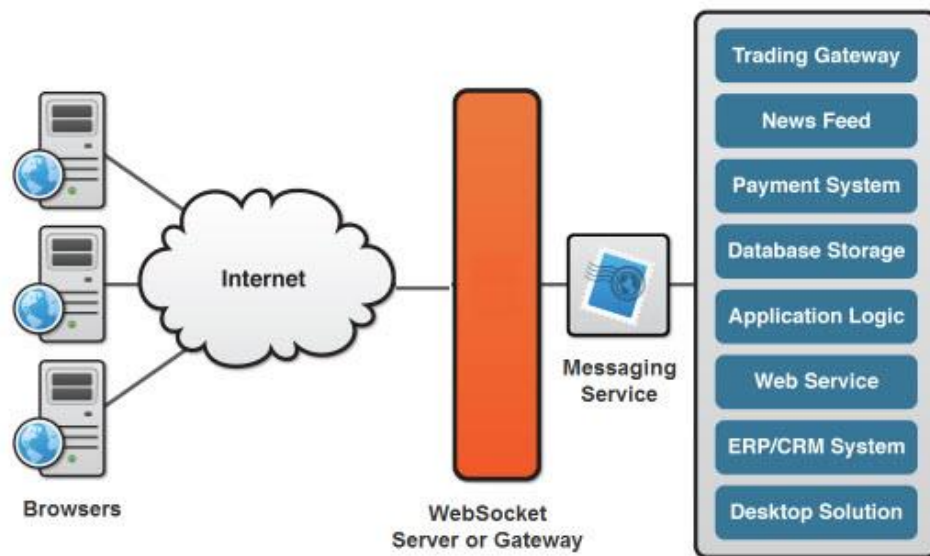


Figure 20 A basic websocket-based architecture [<http://www.websocket.org/img/websocket-architecture.jpg>]

The websocket plays a role as a switcher to send message to the outside service from all clients and vice versa. In addition, it is also reprocess the message before sending them to clients or outside service.

In general, HTML5 websocket has these characteristics:

- Making streaming possible over any connection (support upstream and downstream communications over a single connection)
- Placing fewer burdens on servers. Ability to traverse firewalls and proxies
- Huge reduction in unnecessary network traffic and latency
- Detecting the presence of a proxy server and automatically sets up a tunnel to pass through the proxy.

3.2.3 WebSocket protocol

Before going to the websocket protocol, we should know the problems of web 2.0: when server has the new data, how the client can get that data automatically without refreshing the webpage. We have the following treatments:

1. **Polling:** The browser sends HTTP requests at regular intervals and immediately receives a response. However, real-time data is often not that predictable, making unnecessary requests inevitable and as a result, many connections are opened and closed needlessly in low-message-rate situations. [11]

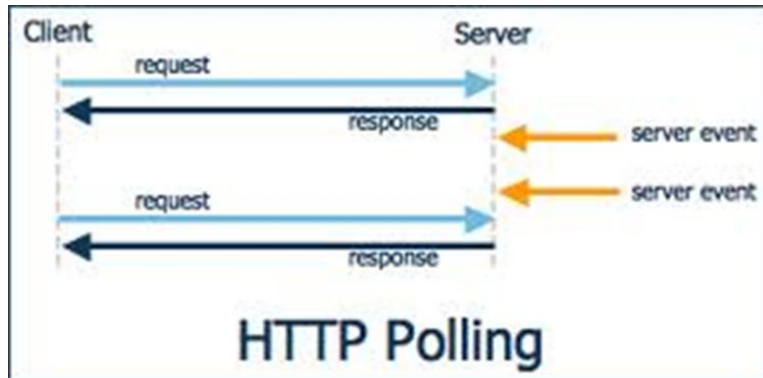


Figure 21 HTTP Polling

[<http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpPolling.gif>]

2. **Long polling [11]:** The browser sends a request to the server and the server keeps the request open for a set period. However, there is a disadvantage with a high message volume; long-polling does not provide any substantial performance improvements over traditional polling. In fact, it could be worse, because the long-polling might spin out of control into a non-throttled, continuous loop of immediate polls.

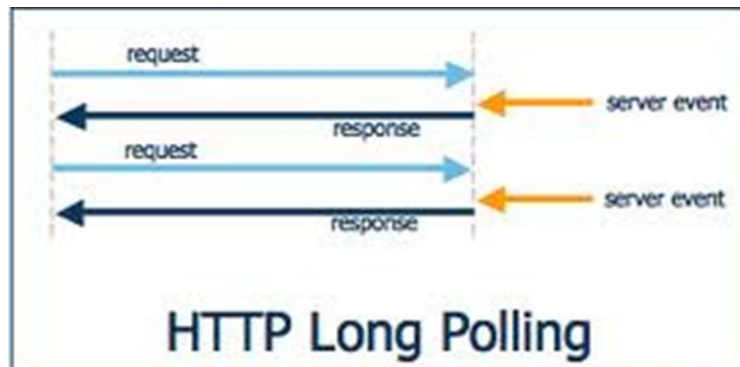


Figure 22 HTTP Long Polling

[<http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpLongPolling.gif>]

3. **HTTP Streaming:** The browser sends a complete request, but the server sends and maintains an open response that is continuously updated and kept open indefinitely (or for a set period of time). The response is then updated whenever a message is ready to be sent, but the server never signals to complete the response, thus keeping the connection open to deliver future messages. However, streaming on HTTP, intervening firewalls and proxy servers may increase the latency of the message delivery.

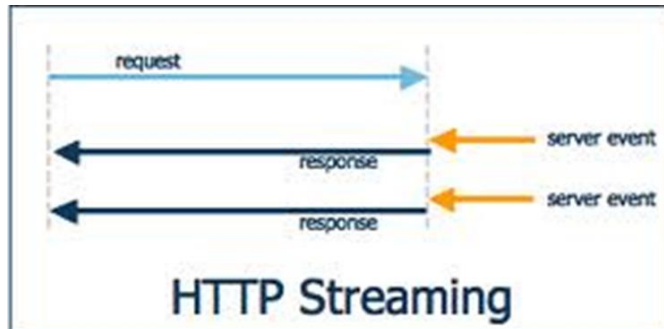


Figure 23 HTTP Streaming

[<http://media.techtarget.com/tss/static/articles/content/WhatistheAsynchronousWeb/HttpStreaming.gif>]

1

All of those above methods for providing real-time data involve HTTP request and response headers, which contain lots of additional, unnecessary header data and introduce latency [11]. We can easily see the complexity of Comet application in the following figure:

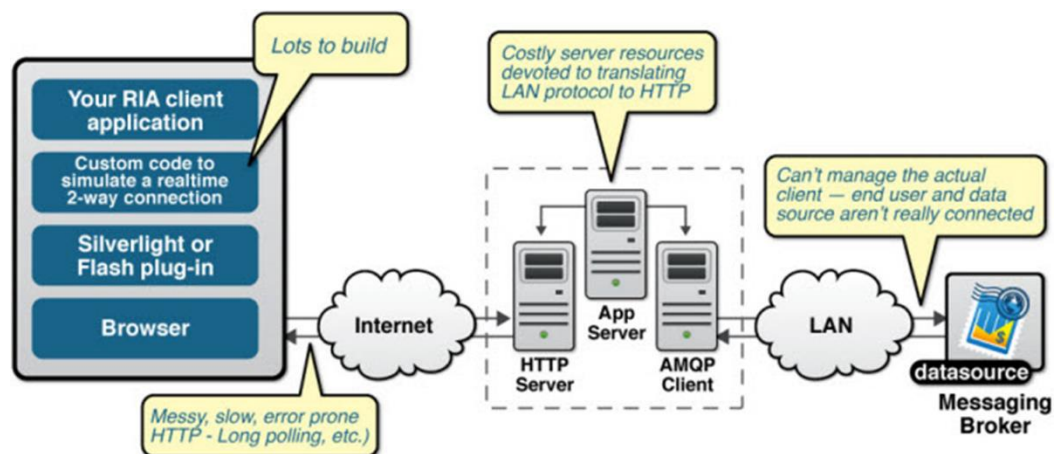


Figure 24 The complexity of Comet applications [<http://www.websocket.org/img/comet-apps.jpg>]

Comet solution is worse and worse when the issue is scaled out. When user experiences the web page that looks like real time web application by using comet solution, this experience will pay the “high price” for the additional latency, unnecessary network traffic and a drag on CPU performance. Our problem is

transferring messages from controllers to clients through servers happening frequently. In addition, when the numbers of clients increases, the comet solution is not a good choice for our project.

Based on above the problems, Websocket appears as the rescue. HTML5 Web Sockets with a full-duplex, bidirectional communications channel can build scalable, real-time web applications. In addition, it not only gives up the problems of Comet but also dramatically reduces complexity.

To establish a WebSocket connection, the client and server upgrade from the HTTP protocol to the WebSocket protocol during their initial handshake, as shown in the following example:

```
GET /text HTTP/1.1\r\n
Upgrade: WebSocket\r\n
Connection: Upgrade\r\n
Host: www.websocket.org\r\n
...\r\n
HTTP/1.1 101 WebSocket Protocol Handshake\r\n
Upgrade: WebSocket\r\n
Connection: Upgrade\r\n
...\r\n
```

Once established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode. Both text and binary frames can be sent full-duplex, in either direction at the same time.

In conclusion, Websocket exists with these main benefits:

- A full-duplex, bidirectional communications channel that operates through a single socket over the Web build scalable, real-time web applications.
- In addition, since it provides a socket that is native to the browser, it eliminates many of the problems Comet solutions are prone to.
- WebSockets removes the overhead and dramatically reduces complexity.
- WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode. Both text and binary frames

We just give out the small comparison between polling and Websocket. We can easily see the better result of Websocket about the time and efficiency:

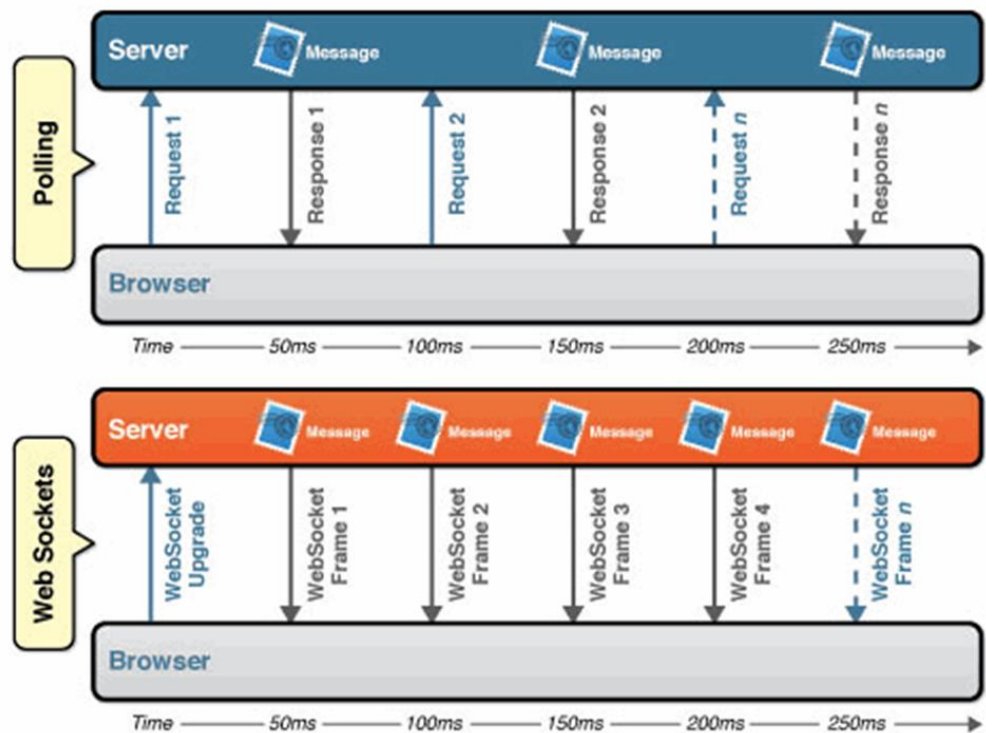


Figure 25 Comparison between Polling and Web sockets [<http://www.websocket.org/img/latency-comparison.gif>]

Based on the strength of websockets as we mentioned above, server based on web sockets can send more messages than polling.

We can also imagine the websocket protocol as the following figure:

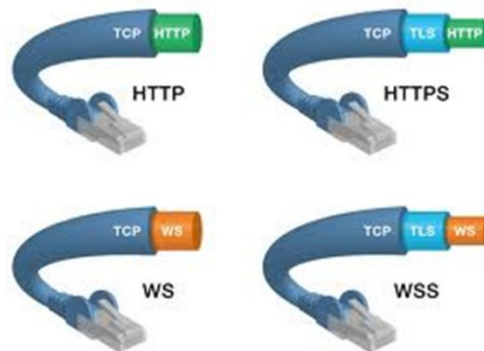


Figure 26 Websocket protocol illustration [<http://web2.sys-con.com/node/2184408>]

In the next part, we will know the list of web browsers which are supported by web sockets.

3.2.4 Supported web browsers with WebSocket

We have the table of these supported browsers:

				
Google Chrome v.14	Internet Explorer v.10	Firefox v.6	Safari v.6	Opera v.12

Table 3List of supported web browsers with Websockets

3.2.5 HTML5 WebSocket API

Using the WebSocket interface is very simple. From a Web page, we can use JavaScript functions to connect to the server. We just create a new WebSocket instance with the value of parameter as a URL that represents the server to which we want to connect, as shown in the following example.

Note that a `ws://` and `wss://` prefix are proposed to indicate a WebSocket and a secure WebSocket connection, respectively.

```
var myWebSocket = new WebSocket("ws://www.websockets.org");
```

A WebSocket connection is established by upgrading from the HTTP protocol to the WebSockets protocol during the initial handshake between the client and the server.

The connection itself is exposed via the `"onmessage"` and `"send"` functions defined by the WebSocket interface. We can use the series of supported event listener to catch up each phase of connection happening in the following example.

```
myWebSocket.onopen = function(evt) { alert("Connection open ..."); };  
myWebSocket.onmessage = function(evt) { alert( "Received Message: " +  
evt.data); }; myWebSocket.onclose = function(evt) { alert("Connection  
closed."); };
```

To send a message to the server, simply call `"send"` and provide the content you wish to deliver. After sending the message, call `"close"` to terminate the connection, as shown in the following example. As we can see, it really couldn't be much easier.

```
myWebSocket.send("Hello WebSockets!"); myWebSocket.close();
```

3.2.6 Potential WebSocket Use case

We have the table of the possible use case of websockets in some domains:





Multiplayer online games	Live sports ticker	Chat applications	Real-time updating social streams
			

Table 4 Websocket use case

Based on the benefits of using Websocket, we easily see that this technology can be used for our semester project as the full-duplex single socket connection over the messages can be sent between client and server. Our controller just needs send the time stamp and actions (play, pause, load video) to it all clients through server in bi-directional web communication. That is the reason we choose Websocket in this semester project's scope.

In the next part, we would like to introduce BinaryJS, which also allows bi-directional web communication. Even so, it is not used in our semester project due to some reasons.

3.3 BinaryJS

3.3.1 What is BinaryJS

BinaryJS [12] is a **lightweight framework** that *utilizes Websockets to send, stream, and pipe binary data bi-directionally* between browser JavaScript and Node.js. This is created by Eric Zhang, a student at UC Berkeley.

3.3.2 BinaryJS benefits

In simple understanding, the BinaryJS can help us to transfer the binary data using JavaScript on the web browser with these benefits:

- BinaryPack serialization format is fast, binary, and JSON-type compatible. Data stays binary end to end
- Automatically chunks large binary data buffers for streaming performance
- Send multiple streams of data concurrently over multiplexed websocket connection

- API implements Node.js read/write Streams. You can pipe any stream into BinaryJS streams (and vice-versa)
- "pause," "resume," and "end" as in the Streams API

Based on the above benefits of BinaryJS, we see that these benefits don't exactly fit the requirements of our semester project. It allows us to transfer the binary data through web browser, but we just would like to transfer the timestamp as well as the type of actions (play, pause, load video) from controller to clients. In the other words, the main point of BinaryJS is aiming to transfer the binary data such as media data or streaming data. This circumstance of this semester project is not like that. We need the other one which is simpler and smaller. So BinaryJS is not the solution for us in this situation.

We use web socket as the main web communication between web page and server. We don't choose which of platform to be used to set up the server in this semester project. In the next part, we give the introduction to Node.js.

3.4 Node JS and its potential

3.4.1 What is Node.js

Node.js [4] is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. Current Version: v0.8.17 (Definition on the website of Node.js).



Figure 27Node.js Logo [4]

In the other words, Node.js is the event server-side JavaScript which is good at handling lots of different kinds of I/O at the same time.

3.4.2 Developing on Node.js

Before starting with Node.js, we should download the Node.js package and install it on the computer.

We can use the following command to start the server:

```
Node localpath/exampleserver.js
```

3.4.3 Making programming on Node.js

This simple web server written in Node responds with "Hello World" for every request.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

To run the server, put the code into a file example.js and execute it with the Node.js:

```
% node example.js
Server running at http://127.0.0.1:1337/
```

Here is an example of a simple TCP server which listens on port 1337 and echoes whatever we send it:

```
var net = require('net');
var server = net.createServer(function (socket) {
  socket.write('Echo server\r\n');
  socket.pipe(socket);
});
server.listen(1337, '127.0.0.1');
```

3.4.4 Potential of Node.js

- Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

These good points help Node.js to become the best candidate for building the lightweight server that transfers information between the clients that take part in this semester project.

In the context of this semester project, we need a small server to transfer commands from video broadcast controller to all clients (play, pause or load video channel). Node.js use JavaScript, light and it's easy to use. These conditions lead to the decision to choose Node.js as the main part to build the server in my semester project architecture.

In the next chapter, we will go into the details of the semester project application that has been developed, explaining how it combines the Websocket, the Node.js with Kinect device in a single television solution.

Chapter 4 Application architecture, main features and experiments

✎ *In this chapter, we will mention the statement of application and the system design in detail. Then we guide the step of installing and its result.*

4.1 System description

The original objective of this semester project is building system application that allows the users to grab their favorite TV shows and share it. But this scope has been widened, and now the viewer can use the Kinect device being connected to a particular application on web browser. In this situation, we named it Broadcast Controller. It broadcasts video channel to all clients and synchronizes it during the time it is being watched.

We divide the system into these parts:

- User can control the Broadcast Controller by hand gesture :
 - **Swipe left**: user stands in front of Kinect in a distance in range of 2 m to 3 m. User swipes the right hand from right to left. This action will trigger the video playing and transferring the timestamps to all clients
 - **Open hand openly**: user also stands as the same. Then user gives his hand forward and opens widely. This action will trigger the video pausing and transferring the message “pause” to all clients
 - **Hand hold**: user moves his hand such that the pointer moves to the channel logo button. This action will “click” on the channel image button and transfer the message of video name to all clients
- **HTML Rendering Source Server**: renders the source of one site of Broadcast Controller and one or many site(s) of clients. This server store and ship the file source HTML of controller and clients to users’ web browsers. Its purpose is the server managing all the web sites in the system.
- **Websocket server** processes and transfers information from Broadcast Controller to client(s) and vice-versa.
- User use keeps the hand over an object to trigger a click event on a video channel. Then:
 - Websocket server sends the name of a video channel to all clients through Websocket server.
 - The broadcast controller and all clients wait until all of them have buffered enough data for playing.
 - All the clients will notify the Websocket server.
 - Websocket server will process this information and give notification to all clients and broadcast controller.
 - It’s ready for playing.

- Every time when user swipes left, the broadcast controller transfer command play to all clients through Websocket server.
- While playing, the broadcast controller sends the timestamp to all clients within the given window interval/ Then the clients will update the current time when its timestamp and broadcast controller's timestamp are not fit in the interval threshold.

4.2 System design

4.2.1 System architecture

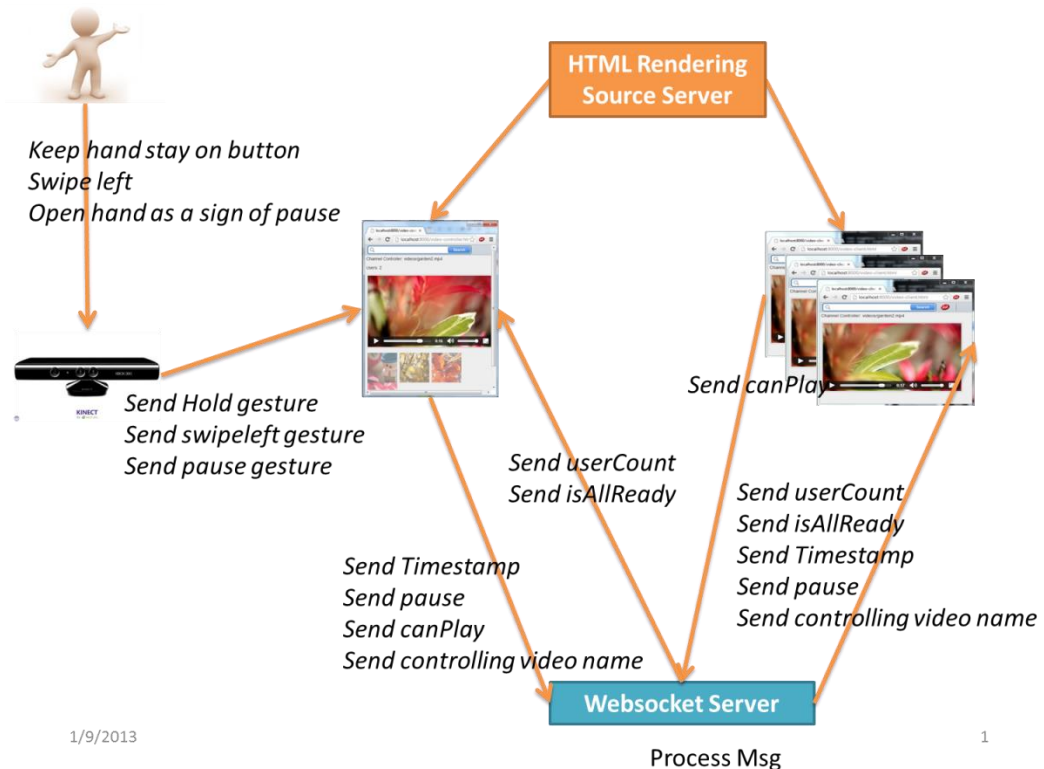


Figure 28 System architecture

Based on the above figure, there are four main parts, which we have been developed in this semester project:

- **HTML rendering Source Server:** plays as the source of keeping media source and rendering HTML source as well as media source for the Broadcast Controller and client(s).
- **Websocket server:** plays as the center component that gets the message from controller, processes it and transfers that message to all clients.
- **Controller:** sends the actions (play, pause, load video) and receives number of users and the message of being ready all. When user swipe left, it send "play" message to all clients. Or user open hand widely, it sends "pause" to all clients.

And user holds on the channel button, it sends the “loadvideo” message to all clients.

- **Clients:** receive all messages from controller through Websocket server and send the message of “canplay” to server. The message “canplay” informs the server that the video has been correctly buffered.

4.2.2 Sequence Diagram

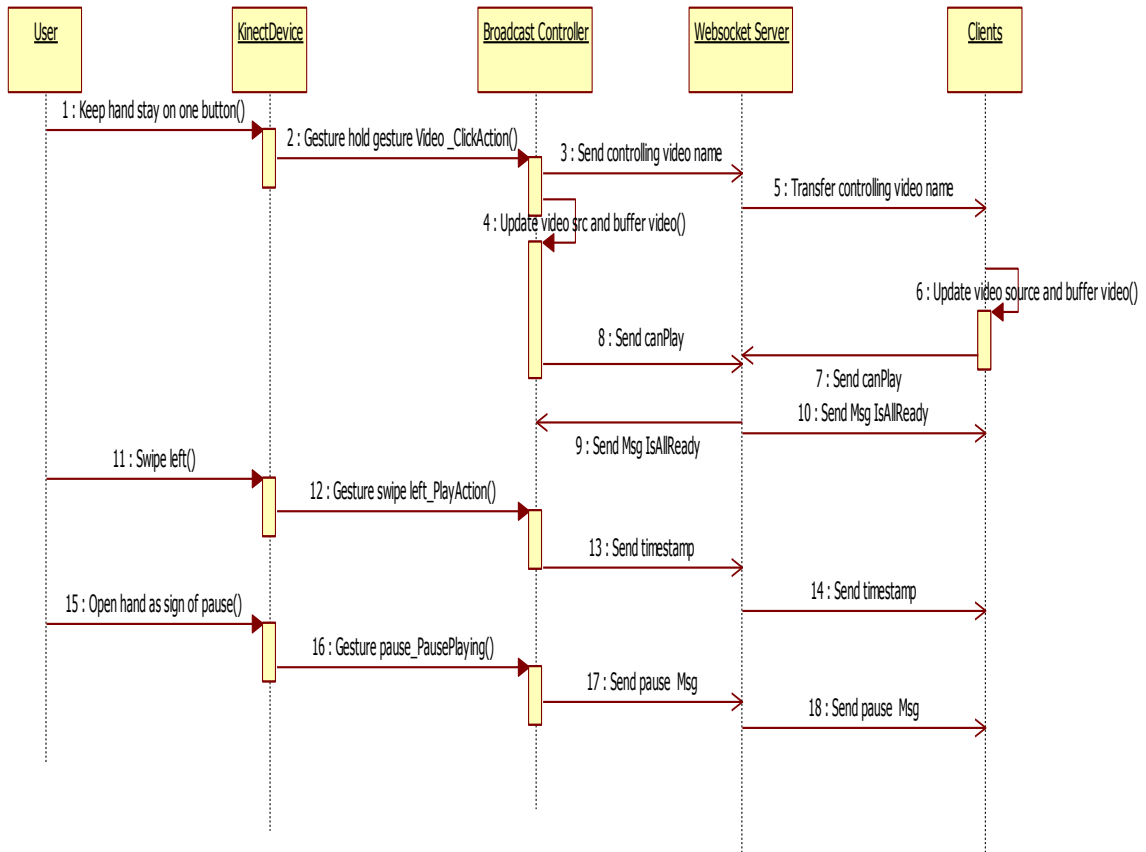


Figure 29 Sequence diagram of project application

User interacts with the Kinect device by gesture:

- Placing the pointer over the area of a button: Controller sends the video name through server and server transfers that video name to all clients. When all clients and controller buffer the video enough, they will send the message “isAllReady” to websocket server. This server will inform all clients and the controller that it is ready for playing from controller.
- Swipe left: playing the video. Send the timestamp to all clients through websocket server.
- Open hand widely as sign of pause: pausing the video. Send the signal of pause to all clients.

4.2.3 Rendering Source Server

The Source Server plays as the source of keeping media source and rendering HTML source as well as media source for the Broadcast Controller and client(s).

- *Code:*

```
Code:
var sys = require("sys");
var express = require('express');

var app = require('express').createServer();
// in this circumstance , we don't need io here
//var io = require('socket.io').listen(app);

app.listen(8000);
// routing
app.use(express.logger('dev'));
app.use(express.static(__dirname ));
// in this circumstance , we don't need io here
app.use(express.static('socket.io/lib' ));

app.get('/video-controller.html', function (req, res) {
  res.sendFile(__dirname + '/video-controller.html');
});

app.get('/video-client.html', function (req, res) {
  res.sendFile(__dirname + '/video-client.html');
});

var userCount = [];

function log(msg) {
  sys.puts(+new Date + ' - ' + msg.toString());
}
```

4.2.4 Broadcast controller

Broadcast Controller sends the actions (play, pause, load video) and receives number of users and the message of being ready all.

- *UI:*



Figure 30 Broadcast controller UI

- *Code for Kinect interaction:*

```
var connection = new WebSocket('ws://localhost:8888');
    if (connection == null)
        connection.onopen = function () {
    };
connection.onerror = function (error) {};
connection.onmessage = function (message) {
    var matches;
    matches = message.data.split(/\s/g);
    switch (matches[0]){
        case "control":
            $("#controller").innerHTML = matches[1];
            break;
        case "userCount":
            userCount= parseInt(matches[1]);
            userCount=userCount-1;
            document.getElementById("userCount").innerHTML =
userCount;
            break;
        case "isAllReady":
```

```

        isAllReadyFlag=parseInt(matches[1]);
        break;
    case "pause":
        video.currentTime = parseInt(matches[1]);
        video.pause();
        break;;
    case "timestamp":
        if (iAmControlling())
            return;
        break;
    }
};

```

- **Problem of sending timestamps:** When the video is playing, the video sends timestamps to all clients. This is for being sure that the clients are synchronized with the controller. The process is repeated, so sending every time with the threshold. The problem is how we detect the **time-window** in this situation. We have set a time window of 5 seconds as the for controller. Every 5 seconds, controller sends the time stamp to all clients through websocket server. We use the timer in this situation:

```

window.clearInterval(myTimer);
window.clearTimeout(myTimeout);
if(isAllReadyFlag>0){
    myTimeout=setTimeout(function(){connection.send("timestamp "
+ video.currentTime)},0);
    myTimer=setInterval(function(){connection.send("timestamp " +
video.currentTime);},time_interval);
}else{
    alert("Not ready for all users");
}
}, false);

```

- **Problem of waiting for enough buffers:** When the video is buffered enough, controller will send the message “canplay” to the server. Server will collect the “canplay” message and transfer the message “isallready” to all clients when all clients are ready to play :

```

video.addEventListener("canplay", function() {
    connection.send("canplay "+video.currentTime);
}, true);

```

4.2.5 Client

Receive all messages from controller through websocket server and send the message of “canplay” to server. The message “canplay” can inform server that it buffers the video enough

- *UI:*



Figure 31Client UI

- *Code for Websocket:* It is similar to the above controller.
- *Problem of waiting for enough buffers:* the same as the controller.
- *Problem of deciding the threshold to detect the client is sync or not:* the value of threshold is calculated average. This value is calculated in the experiment. The way of calculating this value will be show in 4.3.2.

```
var estimatedTimeOnMaster = parseFloat(matches[1]);
var curTime=parseFloat(video.currentTime);
if (Math.abs(estimatedTimeOnMaster-curTime)>threshold) {
    video.currentTime = estimatedTimeOnMaster;
    main_content="Master:"+estimatedTimeOnMaster+" Cur:"+curTime+"
Threshold: "+threshold+" NotSync - Sync";
    bg_color="bg-color-red fg-color-white";
}else{
    main_content="Master:"+estimatedTimeOnMaster+" Cur:"+curTime+"
Threshold: "+threshold+" Sync - Sync";
}
```

As we know that Broadcast Controller will send the time stamp to client in the 5 seconds time window. The client(s) will compare this timestamp with its own time stamp. If the absolute distance of both of these time stamps is in a threshold value, we will decide that it is in sync. Other wise, it is not in sync.

4.2.6 Websocket Server

The server plays as the center get the message from controller, process it and transfer that message to all clients

- **Problem:** how to detect all clients and controller are ready to broadcast or not. This server calculates the number of users ready. When all clients are ready, server will send the message “isallready” to all clients.

```
connection.on('message', function(message) {
    // broadcast message to all connected clients
    var matches;
    matches = message.utf8Data.split(/\s/g);
    if(matches[0]=="canplay") {
        ++isAllReady;
        console.log("Number of ready clients: "+ isAllReady);
        // If controller and client are ready --> start playing
        if(isAllReady>1 && isAllReady==clients.length){

            for (var i=0; i < clients.length; i++) {
                clients[i].send("isAllReady "+isAllReady);
                console.log("All "+i+" are ready: "+ isAllReady);
            }
        }
    }
});
```

4.3 Install and result

4.3.1 Install

- Preliminary step
 - 1/ Install and run Node.js.
 - 2/ Install Kinesis IO.
 - 3/ Connect the Kinect device to your computer and be sure it works probably.
- Run

- 1/ Start server-controller.js with command: ***node [local path]/server-controller.js***
- 2/ Start server-html.js with command: ***node [local path]/server-html.js***
- 3/ Open web page video-controller.html firstly at port 8000
- 4/ Open web page video-client.html secondly at port 8000
- Gesture
 - Hold your hand on the button --> Click video.
 - Swipe left: Play video.
 - Open hand widely and give forward such as the sign of pause: pause video.

4.3.2 Experiments

Hardware and software of running environment with following system information:

Operating System: Windows 7 Ultimate 32-bit (6.1, Build 7601) Service Pack 1 (7601.win7sp1_gdr.111118-2330)

System Manufacturer: TOSHIBA

System Model: Satellite L505

BIOS: InsydeH2O Version 1.50

Processor: Intel(R) Core(TM)2 Duo CPU T6400 @ 2.00GHz (2 CPUs), ~2.0GHz

Memory: 3072MB RAM

Available OS Memory: 2940MB RAM

Page File: 4584MB used, 1294MB available

DirectX Version: DirectX 11.

We run our system with each video in 3 times. We assume that the video are loaded completely on client and controller. We calculate the latency of time stamp between client and controller in 3 times and get the average of them as the value of following table. We have 3 times: experiment 1, experiment 2 and experiment 3. At experiment 1, we wait for video is ready on all clients and controller. Then we play the video on controller and all clients will play that video at the same time. We calculate the absolute distance between the time stamp of controller and client and take its average. Experiment 2 and experiment 3 are just running again as the same as time 1 does.

<i>No.</i>	<i>Video</i>	<i>Latency between clients and controller (s)</i>
-------------------	---------------------	--

1.	320x180 100kbit mp4	Experiment 1:0.018 Experiment 2:0.002 Experiment 3:0.007
2.	640x360 800kbit mp4	Experiment 1:0.022 Experiment 2:0.006 Experiment 3:0.006
3.	1280x720 1,4Mbit mp4	Experiment 1:0.553 Experiment 2:0.056 Experiment 3:0.138
4.	1920x1080 2,7Mbit mp4	Experiment 1:2.367 Experiment 2:1.942 Experiment 3:1.932

Table 5Table of latency between client and controller

Chapter 5 Conclusion

✎ In this chapter, we give the obtained result and the research lines that can be targeted in the future.

5.1 The obtained result

We build the system of application including 4 main parts:

- A HTML Generating Source Server: A server that generates the HTML source code for the web pages of the broadcast controller and all clients.
- A Websocket server: A server will process and transfer information from broadcast controller to all clients and vice versa.
- The broadcast controller is commanded by the Kinect. Users can use hand gesture to control: play, pause, or change the video channel to all clients at the same time. Every 5 seconds, the controller will send the time stamp to all clients to make sure all clients being synchronized with it.
- The client gets commands from the broadcast controller and plays in synchronization with the controller. Based on the threshold in millisecond, client will detect that it is sync or not with controller. This value of threshold is calculated in the above experiment.

The usefulness of this second screen technology is very huge. Users will like it very much because they can experience the real-time entertainment. For example, user A and user B can see the same football match from a video channel. One of them is client and one of them is controller. Both of them can get the real-time entertainment. Moreover, the second screen technology is just a simple example in our semester project. It will be developed more in future in many aspects from playing game or other entertainments.

5.2 Development in future:

In the future we plan to build a system of application with two servers running on a host. We can use smart TV with Kinect device as the controller and

other computers as the clients. On the smart TV, we can surf the web page controller and control the video channel. In the meanwhile, the web browser on computer will open the client web page. This web page will play video in synchronization with controller. In this circumstance of this semester project, we just build it on the local host server on the same computer. So the threshold will be different on the real system of the network.

We also make many experiment on the different device, different size of video, different of type of video on the different web browsers. With these actions, we can know the latency between clients and controller clearly.

If we have more time, we will make the client screen be scalable on iphone or ipad screen. Based on the size of the terminal device, we can change the user interface size to adapt to the current screen size.

Moreover, we can also share the screenshots on the social network if we feel interesting. This can be done easily but we don't have enough time to finish it.

5.3 Personal opinion: what did I learn? Has it been interesting?

I learn a lot from this semester project: from technical to the management of working.

Technical:

- Know and practice the javascript and HTML5 in this project.
- Using CSS as well as the trigger event of objects in HTML tags.
- Know and use the Node.js, WebRTC, BinaryJS, Websocket, and its specification.

Management of working:

- Using Github for tracing the steps of project.
- Writing report weekly and how to show and explain the problems.

REFERENCES

- [1]. LinkedTV: <http://www.linkedtv.eu/>
- [2]. Microsoft Kinect: <http://www.microsoft.com/en-us/kinectforwindows/>
- [3]. Kinesis.IO : <http://kinesis.io>
- [4]. Node.js: <http://nodejs.org/>
- [5]. Websocket: <http://www.websocket.org/>
- [6]. Kinect on Wiki: <http://en.wikipedia.org/wiki/Kinect>
- [7]. Developing Kinect for Windows: <http://www.microsoft.com/en-us/kinectforwindows/develop/new.aspx>.
- [8]. Developer Download: <http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx>
- [9]. Kinect SDK Architecture: <http://msdn.microsoft.com/en-us/library/jj131023.aspx>
- [10]. MSDN- Skeletal Tracking: <http://msdn.microsoft.com/en-us/library/hh973074.aspx>
- [11]. HTML5 Web Sockets: *A Quantum Leap in Scalability for the Web* <http://www.websocket.org/quantum.html>
- [12]. BinaryJS: <http://binaryjs.com/>
- [13]. OpenKinect: http://openkinect.org/wiki/Main_Page
- [14]. Kinect hand process: <https://github.com/bmwesting/Kinect-Hand-Processing>
- [15]. Kinect Gesture Library: <https://github.com/eawerbaneth/Kinect-Gesture-Library>
- [16]. Air Kinect Gesture: <https://github.com/tonybeltramelli/Air-Kinect-Gesture-Lib>
- [17]. Qt Air Cursor: https://github.com/nemein/Qt_AirCursor
- [18]. Xkin: <https://github.com/fpeder/XKin>
- [19]. Kinesis Demos : <http://kinesis.io/demos>