Software Requirements Specification

for

p2pcast

Version 1.10

Prepared by

Group Name: NP Compete

Daniel Vicory	7024755	dvicory@gmail.com
Omar Masri	8646101	marismaro@gmail.com
Jerry Medina	4349882	jerryfoxyt@gmail.com
Nicole Theokari	5179528	nicole@theokari.com
Justin Liang	5217286	justinyliang@gmail.com

Instructor: Chandra Krintz

Course: CMPSC 189A

Lab Section: Wednesday 6:00-6:50PM

Teaching Assistant: Geoffrey Douglas

Date: 2/11/14

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.10	Daniel Vicory, Omar Masri, Jerry Medina, Nicole Theokari, Justin Liang	Formatting improvements, modifications to user interface, more possible use cases added	03/04/14

Version	Primary Author(s)	Description of Version	Date Completed
1.00	Daniel Vicory, Omar Masri, Jerry Medina, Nicole Theokari, Justin Liang	Initial Draft	02/11/14

1 Introduction

This section gives a scope description and overview of everything included in this SRS document. Also the purpose for this document is described and a list of abbreviations and definitions is provided.

1.1 Document Purpose

The purpose of this document is to present a detailed description of the peer-to-peer broadcast platform known as p2pcast. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, and the constraints under which the system must operate. This document is intended to be used as a reference for developing the initial version of the p2pcast platform for the development team.

1.2 Product Scope

p2pcast is a browser-based method of broadcasting video using peer-to-peer technologies. Its purpose is to allow users to stream video easily, quickly, and free of cost. There are two distinct components that make up the p2pcast platform. There is a component that runs in the user's web browser and another component that functions as a server to enable connectivity between users' web browsers.

Current services that provide a similar functionality include Skype and Twitch.tv. Skype uses an external program for its peer-to-peer streaming video. Twitch.tv is a streaming website which uses dedicated centralized servers to support its massive audience. p2pcast proposes to create a simpler de-centralized service to use used with no additional installments and with the propose of being multiplatform. Multiplatform will rapidly grow as support in existing browser and subversion for mobile continue to extend their support of WebRTC.

1.3 Intended Audience and Document Overview

This document is primarily intended for the development team, our mentors at Citrix Online, professors of CS189A and CS189B, and UCSB Capstone faculty. The SRS will discuss the details and implementation of the project. It is recommended for those without computer science experience to begin with section 1.4 for important acronyms and abbreviations that appear in this document.

1.4 Definitions, Acronyms and Abbreviations

Term	Definition

Web Application	The component that runs on end-user's web browsers, composed of HTML and JavaScript and is also the part that talks to other peers directly
Application Server	The component that runs on the host, used to index channels, facilitate peer connections, and serve the web application to web browsers
Broadcaster	A peer who originates video streams to users
Channel	A method of namespacing different broadcaster's video streams. Channels are created by a broadcaster which contains only their own video stream. Users can join a channel to view that broadcaster's video stream, in which they become a peer for that specific channel.
Citrix Online	The online services division of Citrix Systems, Inc.
End-user	A person who uses the p2pcast web application, whether to broadcast or view video streams
Forwarding/ Rebroadcasting	The process of one peer in a network transmitting data it is receiving from another peer in the network to at least one other peer
Google Chrome	A web browser developed by Google, the primary target of our p2pcast web application
Host	User that runs a p2pcast application server
ICE Framework	ICE is a framework used to connect peers. First tries UDP, then TCP with HTTP, then TCP with HTTPS, then TURN servers.
ICE Candidate	An ICE candidate is a network interface and port of a peer that is using the ICE framework

JavaScript	A dynamic computer programming language, the primary development of p2pcast is done in
NAT	A network protocol used in IPv4 networks that allows multiple devices to connect to a public network using the same public IPv4 address.
Node.js	A platform built on Chrome's JavaScript run-time, V8, for easily building fast, scalable network applications
p2pcast	A web application that allows for peer-to-peer video broadcasting
Peer	A browser that is made available to be connected to by other peers, can be a broadcaster or user
Peer-to-peer	A method of communication, where most data is transmitted between end-users instead of centralized servers
PeerConnection	An object from the RTCPeerConnection API
SDP	Session Description Protocol (SDP) is a format for describing streaming media parameters used in signaling
Signaling	A process to exchange control messages and coordinate communication between two peers
Socket.IO	A JavaScript library for real-time web applications
SRS	Software Requirements Specification
Stream	A sequence of data provided in a forward iterable-only manner

STUN	Session Traversal Utilities for NAT (STUN) is a protocol that uses a third-party STUN server to allow peers to discover each other's IP address even if they are behind a NAT
TCP	This provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to a local area network, intranet or the public Internet.
TURN	Traversal Using Relays around NAT (TURN) is a protocol that uses a third-party TURN server to allow peers to receive or transmit data over TCP or UDP connections even if they are behind a NAT
UDP	A simple transmission model with a minimum of protocol mechanism.
Web Browser	A software application for retrieving, presenting and traversing information resources on the World Wide Web
Web Server	Computer software that deliver web pages to web browsers which may consist of static and dynamic (JavaScript) components
WebRTC	A W3C draft standard that enables Real-Time Communications (RTC) capabilities for web browsers via simple JavaScript APIs

1.5 Document Conventions

Bold text – Signifies important content or keywords
Italic text – Signifies comments

1.6 References and Acknowledgments

- WebRTC Specification: http://dev.w3.org/2011/webrtc/editor/webrtc.html
- Node.js Documentation: http://nodejs.org/api/
- Socket IO: http://socket.io/

2 Overall Description

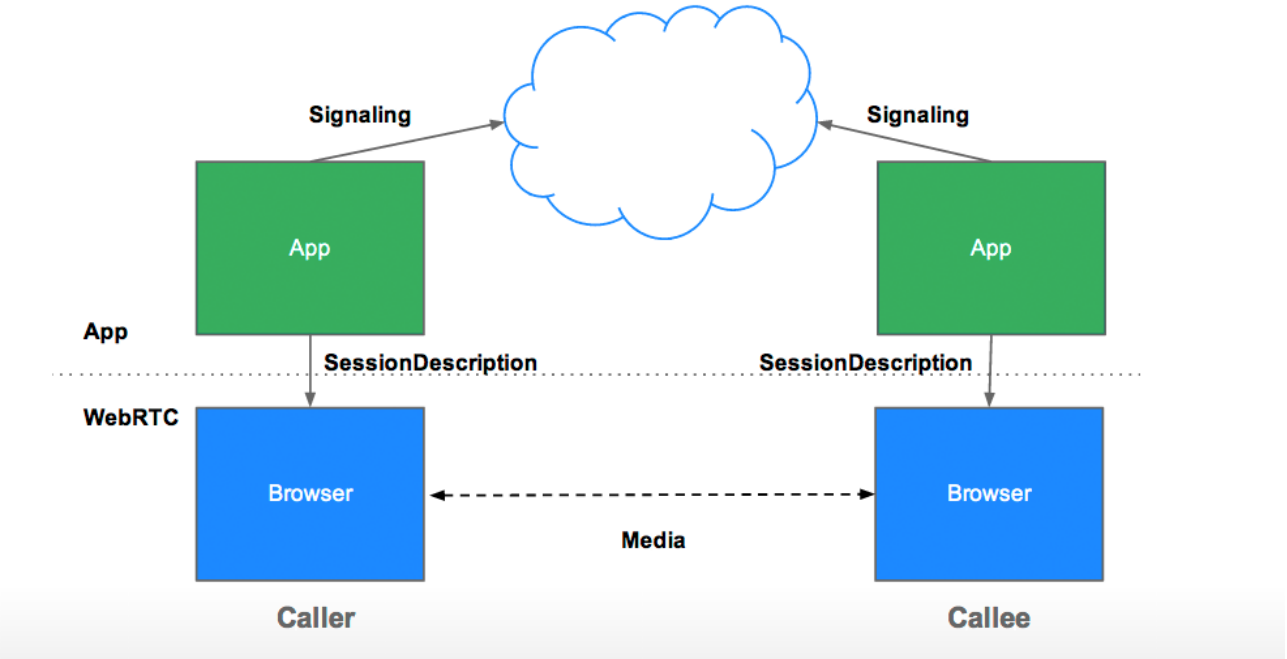
2.1 Product Perspective

There are currently no easy or decentralized way to do video broadcasting. Existing solutions to video broadcasting require Operating System centric programs or browser plugins. Additionally, they require the use of significant centralized server resources. This means that offering video broadcasting services is neither cheap nor convenient for users to decide to live stream a broadcast. Two popular approaches to live streaming video are Skype and twitch.tv. however, Skype is a program which a user and all viewers must install. Additionally it is not particularly suited for broadcast, though it is peer-to-peer. The second example Twitch.tv is centralized and requires their a third party program to stream. Both are proprietary formats and users are not in control of their own broadcast streams.

The uses of video broadcasting for the world are limitless. For example a user could want to share their professor's lecture. Another interesting use case would be sharing a local event, such as a concert, with multiple peers. Directly streaming to all the viewers would be impossible in these couple of scenarios. The average user would likely not have the CPU or bandwidth resources to support more than a few viewers. If a user wanted to support more than a few viewers then said user would have to make use of a service to rebroadcast their stream to all their viewers. Such a service will have restrictions: what a user is allowed to stream, how many viewers are allowed to view the stream, and it would have associated with the service.

2.2 Product Functionality

- A broadcaster can go to go to a website and stream video to other users via a broadcaster-created channel
- A user can go to a channel on the website and watch a broadcaster's stream
- A user-friendly experience
- Each stream has its individual channel
- Each user will have a unique alias



2.3 Users and Characteristics

- Professors who wish to live broadcast their lecture to their students
  - Students can also rebroadcast to other students - perhaps a fellow classmate fell ill and was unable to attend
- Broadcasters can use this to share various entertainment content
  - Concerts
  - Shows
  - Live events
- Civilians can use this to stream a live news
  - Riots
  - Car accident
  - Political debates
- Online bloggers streaming a blog
- Scholars can use this to stream academia
  - Online tutorials
  - TED talks
  - Live demo

2.4 Operating Environment

- HTML5
- JavaScript/ECMAScript 5
- Node.js v0.10.25
- Windows 7+ and Mac OS X 10.8+
- Browser(s)
    - Google Chrome v32+
- WebRTC API
    - MediaStream API (getUserMedia)
    - RTCPeerConnection API
    - RTCDataChannel API
    - ICE Framework
- Servers
    - STUN
    - TURN
    - Any 3rd party server capable of running Node.js

2.5 Design and Implementation Constraints

The biggest constraint to the system and users will come across includes incompatibility of WebRTC with their preferred browser.

- The latest Chrome version the browser supports
- Bandwidth reduction/limitation on peers with limited internet capabilities
- A reconnection system as a safety net in the case of a sudden broadcaster disconnection
- TCP based vs UDP based
    - Reliability vs Speed and minimizing bandwidth usage
- Peers behind NATs
    - STUN/TURN servers needed
    - ICE framework

2.6 User Documentation

Our application will be primarily designed with user-friendliness in mind. The application will have a lightweight and simplistic GUI that will attempt to hide application and network complexities. In addition a video tutorial will be made to show basic functionality of p2pcast for incoming users.

2.7 Assumptions and Dependencies

- User has stable modern internet connection
- User has the latest version of Google Chrome
- User computer has enough power to rebroadcast
- Broadcaster has either an internal or external webcam
- The service is used preferably on a desktop or laptop
    - Currently mobile support needs to catch up

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The user interface will be designed with simplicity and functionality in mind. More features will be added later such as a channel list and descriptions for each channel.

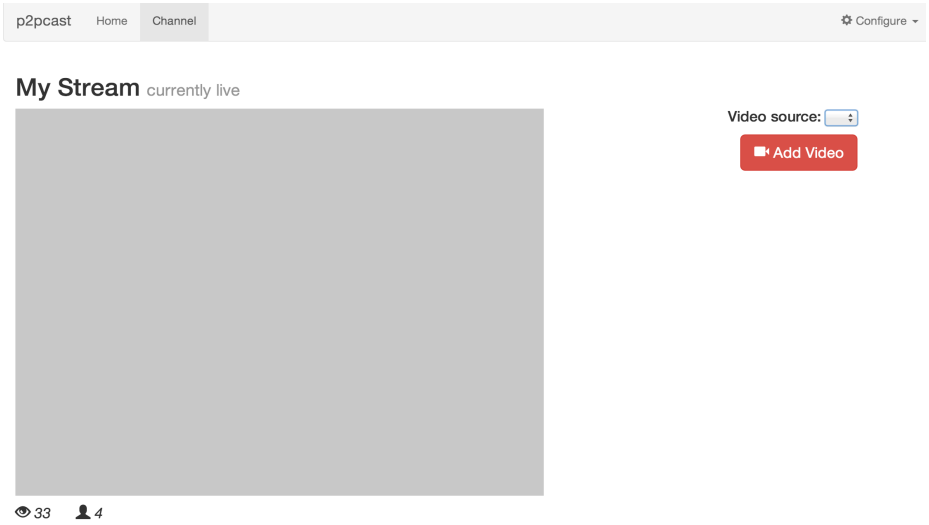


Figure 3.1.1: Mockup of the view page from the broadcast's point of view.

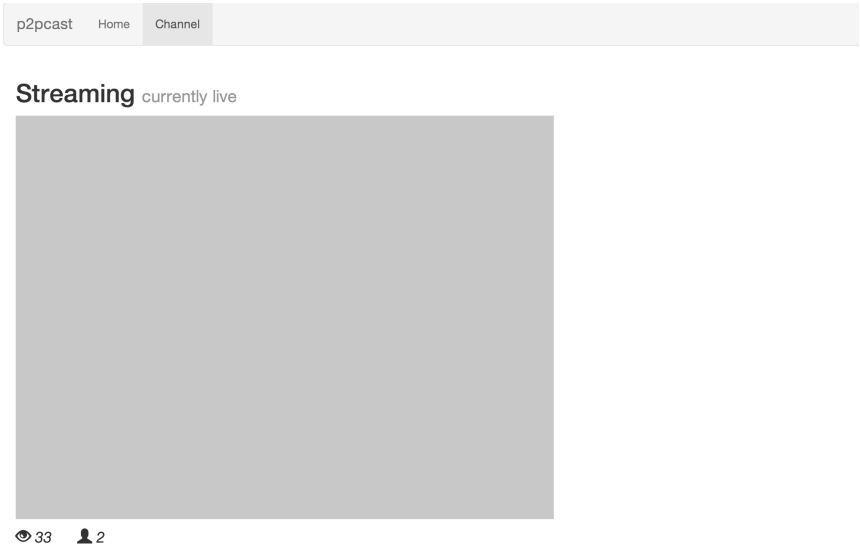


Figure 3.1.2: Mockup of the view page on the viewer's side with automatic connection to a broadcaster.

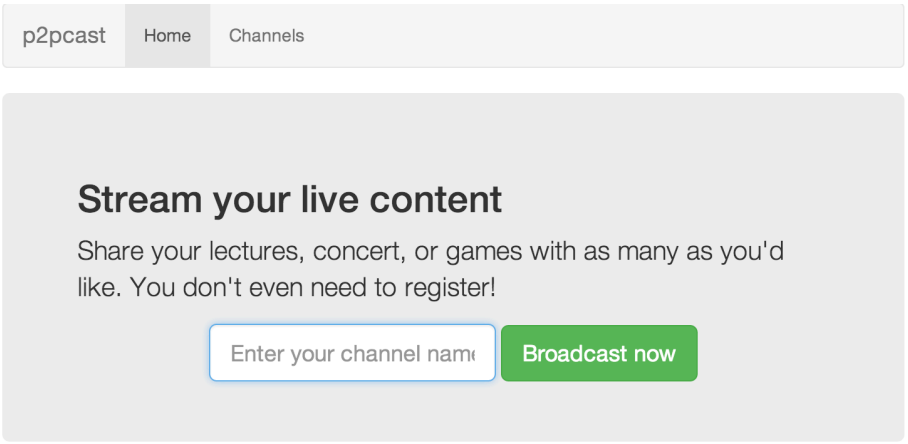


Figure 3.3.3: Mockup of homepage for p2pcast where user can select channel name and begin broadcasting.

3.1.2 Hardware Interfaces

P2pcast will be using the WebRTC API, specifically MediaStream(), to receive data from a user's webcam and/or microphone. MediaStream() abstracts the webcam and/or microphone complexities from the user. Hardware interfaces include desktops, laptops, and mobile devices, which will be abstracted by the user's browser and operating system.

3.1.3 Software Interfaces

WebRTC JavaScript framework, tools for UI (bootstrap), websockets, and other special tools used for databases.

3.1.4 Communications Interfaces

Communication interfaces will be abstracted by the WebRTC API, specifically RTCPeerConnection(). With RTCPeerConnection() a Session Description Protocol (SDP) hides many of the network and media information from the application. This allows communication between two peers that can be designed with a high level approach without worrying about the lower level implementation. RTCPeerConnection() also has encryption for the data channels and signaling mechanism. WebRTC uses DTLS and SRTP for its encryption protocols.

P2Pcast also focuses on the ICE framework for connecting two peers directly. The ICE framework will first try to connect the peers using UDP with the most minimal latency. If UDP fails the ICE framework will try TCP with HTTP, and later TCP with HTTPS. Finally when a direct connection

does not work, ICE will try to use TURN servers to connect peers by adopting a method that requires a third party.

3.2 Functional and Behavior Requirements

3.2.1 Use Case View

- As a broadcaster, I want to go to a website and stream my video to other viewers.
- As a viewer, I want to go to a website and watch other people's live streams.
- As a user, I want a user-friendly experience.
- As a user, I should know whether my web browser is supported.
- As a broadcaster, I want to be able to add filters to my broadcast.
- As a broadcaster, I want to be able to stream what's coming from my webcam.
- As a broadcaster, I want to be able to stream what's on my screen.
- As a broadcaster, I want to be able to create a "Channel" to stream to.
- As a viewer, I want to be able to select a channel and see the stream from the channel.
- As a viewer, I want to be able to disconnect from a channel, and select another channel to view whenever I want.
- As a user, I want to be able to communicate with other users in the channel in real-time.
- As a broadcaster, I want to be able to stream to a lot of people without using a lot of resources.
- As a broadcaster, I want to be able to stream to a few viewers who will rebroadcast my stream to other viewers.
- As a viewer, I want to be able to view channels without constraining my bandwidth as a peer.
- As a viewer, I want to be able to rebroadcast a stream effectively with little to no technical knowledge.
- As a broadcaster, I want the service to be able to intelligently choose which peers will be able to rebroadcast the stream most effectively.
- As a user, I want my streams to be interrupted as little as possible when a peer encounters issues.

3.2.2 Potential Use Case View

- As a broadcaster, I want to be able to select multiple cameras to broadcast from simultaneously.
- As a user, I want to be able to use p2pcast with the Mozilla Firefox browser.
- As a user, I want to be able to register and browse through a list of channels that other users are broadcasting on.
- As a user, I would like the option of using a mobile device to view or broadcast content
- As a user, I would like the option of having live chat between broadcasters or viewers

- As a broadcaster, I would like the option of sharing my computer screen as a video source to viewers while simultaneously broadcasting from a webcam.

4 Other Non-functional Requirements

4.1 Performance Requirements

1. Creating a new channel as a broadcaster should happen immediately, taking no more than 5 seconds.
2. Joining an existing channel should happen quickly, a live video stream shall begin within 10 seconds.
3. Sudden and unexpected peer disconnections should trigger a new connection to another peer and begin streaming again within 30 seconds.
4. Planned peer disconnections (ex: user leaving channel page and other peer notified) shall be handled gracefully and minimize stream disruptions and begin streaming again within 10 seconds.
5. Peers shall be able to handle rebroadcasting to at least 3 other peers, using recent Apple MacBook Pros for baseline performance of this function which is mainly bottlenecked by browser implementations of WebRTC and video decoding/transcoding.

4.2 Safety and Security Requirements

Safety

- Use of the broadcast service should be done ethically and morally
- Broadcasters shall know when their camera is broadcasting live
- Assume any broadcast may potentially be recorded by any viewer

Security

- Broadcasters should assume their video stream is not private to anyone that has the channel name
- Communications between peers is encrypted with DTLS-SRTP as defined in WebRTC
- Naïve denial of service attacks should not be possible
    - For instance, peers could degrade stream quality or take on peers and then later purposefully drop their connections to them
    - Some denial of service attacks can be avoided by avoiding new peers or those that drop packets
    - Peers rebroadcasting a different stream than what is from the broadcaster will not be addressed initially

4.3 Software Quality Attributes

4.3.1 Adaptability

- Peers will attempt to modify bandwidth requirements in the face of changing network conditions, or for low-bandwidth connections such as mobile
- Channel concept will be an overlay over p2pcast's peer-to-peer network allowing for the core signaling functions to be separate

4.3.2 Availability

- Any user may access the service to broadcast or view broadcasts via the website
- Server-side software will not have any unusual requirements allowing for deployment in many environments

4.3.3 Correctness

- We will be developing in test-driven environment
- Testing and error checking will be done periodically to ensure correct functionality between big changes in our code

4.3.4 Flexibility

- The service will work over various operating systems such as Linux, Window, and Mac OS X, with the only requirement being the latest version of Google Chrome
- The user experience will be identical across operating systems

4.3.5 Interoperability

- Users will be able to communicate with each other even if their systems consist of different platforms, as long as they are running the latest version of Google Chrome
- System differences between the two peers should be hidden from the user

4.3.6 Maintenance

- End-user shall not need to take any unordinary measures to use p2pcast
- The host shall be competent in the administration of common web application software and techniques
- The host must make sure the p2pcast web application is accessible to end-users
- The end-user will not need to download or install any software to make use of p2pcast

4.3.7 Portability

- The service should be able to run on any latest version of Google Chrome browser with no additional plugins or software on a desktop or laptop.
- (talk about client-side and server-side portability)

4.3.8 Reliability

- The p2pcast server will attempt to make sure a peer always has a connection to another peer in the network
- Lower quality connections will be attempted to be detected and resolved automatically with a different peer connection
- Navigation between different channels should be a normal operation and not cause undue interruption to peers you are rebroadcasting to

4.3.9 Reusability

- The WebRTC API was designed in mind to be re-usable, which is the core of this application

4.3.10 Robustness

- Disconnected connection between two peers will be remedied with minimal down time
- Software will try to reconnect the disconnected peer with another healthy

4.3.11 Testability

- The WebRTC API handles many of the network and application complexities
- Bugs and other network issues will be noticeable and caught right away if video connection disconnects or video quality deteriorates
- Entry and exit points to our application will be clearly defined and easy to navigate to

4.3.12 Usability

- The service should be easy enough to learn quickly.
- The user interface should be intuitive with minimal user set up.

Appendix A – Data Dictionary

<Data dictionary is used to track all the different variables, states and functional requirements that you described in your document. Make sure to include the complete list of all constants, state variables (and their possible states), inputs and outputs in a table. In the table, include the description of these items as well as all related operations and requirements.>

Appendix B - Mentor Meetings

1/21/14

- Use Pivotal Tracker and set it up.
- For next week, get the example on multiple computers.
- Next time, go over stories and teach us how to be a product owner.
- Looking at the http://cs.ucsb.edu/~dvicory/cs189a/relay_video_track.html code it seems to be working correctly. The next step is to not try it locally and to try it on different browsers.
- Could build a node.js server or get it from Ashish.
- A lot of stories can be turned into epics. Also allowed to have technical user stories.
- Focus on the first what is going to come from the webcam and don't include the screen sharing in the SRS.
- Logins for security purposes? Or a little band trying to perform would not want that. Identification agnostic.
- What is on each page? Title, channel name, description, number of viewers.
- Concentrate on the basic webpage that can broadcast to other people.
- Build up the layers, connections, then channels, then logins
- Firebase can be used. Focus on media rebroadcasting.
- Everyone can build a tree, but the problem comes when someone drops out. This will be our main focus.
- Story: No specific user gets overloaded.
- We have a lot of technical stories but we can definitely breakdown the user stories once we start using them.
- Use twitch.tv as an example.
- Spike is a time bound experiment.

1/28/14

- Diagram Design: Web Sequence Diagrams
- UI Mockups: balsamiq, write HTML and then just take snapshots
- Use cases are very broad:
    - Play or pause the broadcast
    - Filters over the video
    - Leaving the broadcast
- User stories are broken up use cases or epics:
    - Can be things that address errors, i.e. what if a node leaves?
    - What if you are in the wrong browser?
    - What if there is no network connection?
- Ask Chandra:
    - Where do the user stories go in the SRS?

  - Picture markups?
  - Clarification on what the “complete slice is”

2/4/14

- Ignore the lag, don't need to worry about the real time.
- SRS Comments:
- 2.4
  - Server side
  - What is node.js running on?
  - Platforms
  - To deploy it what do I need? Hardware platform and what needs to be installed
- 2.5
  - Browser crashing
  - Dropping out not cleanly
  - Peers dropping out
- 2.6
  - Do a video tutorial
  - No manual
- 2.7
  - Computer is powerful enough to broadcast
  - Microphone and webcam is only required for the broadcaster
- 3.1.2
  - How will you deploy as a company?
  - Browser APIs
- 3.1.3
  - node.js
  - Database, tools, libraries
  - Bootstrap
  - Browser to server and vice versa
  - Too much? It should go in the design doc
- 3.1.4
  - HTTP
- 3.2
  - server and client (things they send to each other)
  - User and client interactions
- 4.1
  - Add hard numbers, 30 seconds for example
  - Only the first one is a performance requirement
- 4.2
  - ddos attacks

  - People injecting themselves
  - Hijacking the stream, sending frames back to the broadcaster but probably don't worry about it
  - Full control of nodes that join through the server
  - Broadcaster should know that the camera is on and broadcasting (status "We are on the air")
  - What do we want to do?
- 4.3
  - Maintainability, look in blue
  - Portability, client and server side
  - Reliability, server really reliable, client not so much
  - Address each of them

2/11/14

- Get interaction design where the user can touch all of the functionality without making it look nice. From this you will see what objects you are using then do the formal design.
- Focus on the core server and worry about the database later.
- Someone should see if audio can be rebroadcasts
- Things to finish for the next sprint:
  - User can join without specifying a peer
  - Finish the design specification
- Web Stats API
- Balanced binary tree initially for p2pcast

2/18/14

- Sequence diagrams that connect the state diagrams
- It does not need to be broadcaster in sequence diagram
- Notify disconnection to disconnect()
- Set up WebRTC stuff or put a box around it (Need to make modifications to state-diagram and abstract away WebRTC details from sequence diagrams?)
- Worst case scenario closing window (connection) vs. the one where we wait for permissions from the server
- Don't need to add tree specifications, don't commit to a tree algorithm

- Need a sequence for a creation of a channel, release of a channel, broadcaster disconnecting, pause/play

- Components are the ones from the picture

2/25/14

- Add ice candidates to Connection Handshake and take out final answer from initial peer

- Look into what happens when disconnection occurs. How long will it take for the server to reissue a new connection.

- Screen sharing and video discussion. Example of this is webinar

- url channel name suggestions - easy to read URL for users

Appendix C - Group Notes

Tuesday, Jan 14, 2014

- Before going to the Citrix office we all briefly watched the video on WebRTC from Google.
    - https://www.youtube.com/watch?v=p2HzZkd2A40
- We went to the Citrix office for the first time to meet and discuss project with our mentors.
- We also discussed the project in more depth so we can complete the vision statement.
- We made a plan for next week, which includes making user stories.

Wed, Jan 15, 2014

- We gathered together to work on the vision statement after the discussion section
- We also met with Geoff and scheduled a time to meet with him each week.

Thurs, Jan 16, 2014

- We reviewed WebRTC
    - http://www.webrtc.org/
- Also javascript
- Set up node js for every member

Fri, Jan 17, 2014

- Set up our GitHub repository

Sun, Jan 19, 2014

- The group came up with different use cases

- Tested code to make sure video forwarding works because it is pretty new in Chrome
    - http://cs.ucsb.edu/~dvicory/cs189a/relay_video_track.html

Mon, Jan 20, 2014

- Set up node.js on our computers
- Continued looking at code from yesterday
    - http://cs.ucsb.edu/~dvicory/cs189a/relay_video_track.html
- Looked at this WebRTC review
    - http://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-simple

Tues, Jan 21, 2014

- Met with Geoff for our weekly meeting
- Met with our mentors for our weekly meeting

Wed, Jan 22, 2014

- Daniel: Setup private projects on Pivotal Tracker by emailing Ross

Thurs, Jan 23, 2014

- Got prototype server code from Ashish and began to examine it

Fri, Jan 24, 2014

- Started group programming and understanding existing codebase

Sun, Jan 26, 2014

- Finished peer-to-peer prototype with simple Node.js server

Mon, Jan 27, 2014

- Update tasks on Pivotal Tracker

Tues, Jan 28, 2014

- Met with Citrix and decided that for the next meeting we should try to have the SRS done and also working on updating Pivotal Tracker with tasks

Wed, Jan 29, 2014

- Sprint planning

Thurs, Jan 30, 2014

- Uploaded the SRS to the Skydrive so that we can edit individually

Fri, Jan 31, 2014

- Worked as a group on the SRS

Sat, Feb 1, 2014
- Began extensive work on wireframing, beginning with the broadcaster's view

Mon, Feb 3, 2014
- Met as a group to work on the SRS

Tues, Feb 4, 2014
- Met with our mentors and showed them what progress we had made on the SRS

Wed, Feb 5, 2014
- Continued to work on the SRS

Thurs, Feb 6, 2014
- We did not meet because everyone had midterms and projects the following week

Fri, Feb 7, 2014
- Finished up wireframes and added them to the SRS

Mon, Feb 10, 2014
- Completed our SRS and prepared for the demo tomorrow

Tues, Feb 11, 2014
- Look for a deployment service (heroku, joyent) that has standardized database support (postgres, mongoDB, mysql.. etc)
- Work on documentation (find a framework first)
- Look into bandwidth issue
- Design decisions (data structure for peers)

Thursday, Feb 13, 2014
- Team met up at Omar's place to discuss and finalize plans for sprint 2
- Decided on Heroku as our team's server provider

Friday, Feb 14, 2014
- Valentine's Day
- Switch to Heroku for deployment

Saturday, Feb 15, 2014
- Began work on Design Specification

Sunday, Feb 16, 2014
- Worked on Design Specification

- Got Vizio Office installed for everyone's pc - free from ECE department

Midterm Week

Monday, Feb 17, 2014

- Worked in Visio for UML and state diagram for Design Specification

Tuesday, Feb 18, 2014

- Met with mentors and showed them our work-in-progress design documents

Wednesday, Feb 19, 2014

- Group bonding exercise

Thursday, Feb 20, 2014

- Met up to discuss weekend schedule/plan

Friday, Feb 21, 2014

- Outlined Class UML 1-4

Saturday, Feb 22, 2014

- CS 170

Sunday, Feb 23, 2014

- Developed more UML diagrams for the Design Spec - ready to double check design with mentors and professors.
- Brief update to pivotal tracker tasks

Monday, Feb 24, 2014

- mock-up integration with prototype
- worked on code for allowing end-users to join without specifying a peer name

Tuesday, Feb 25, 2014

- Merged branches into Master on git to prepare for demo
- Fixed diff conflicts from merge
- Updated video source html for broadcast and peer
- Worked on possible feature for upcoming demo

Wednesday, Feb 26, 2014

- We all went to the awesome section
- Prep for demo after the section
- Debugged demo

Thursday, Feb 27, 2014

- Quick preparation for Final Demo in respect to everyone's schedules

- Final Demo planning; features & priorities

Sprint 3

Friday, Feb 28, 2014

- Rain Rain go away ~
- Cancelled in person meet up for facebook planning/discussion

Saturday, March 1st 2014

- Worked on prototype tree structure for channel to handle disconnections better.
- Design Spec addition

Sunday, March 2nd 2014

- Received flowchart from Citrix's UX Hang Yu
- Finish UML Diagrams
- Wrap up the Design Spec
- Finished up the Class UML with Visio

Monday, March 3rd 2014

- Finishing touches to the Design Spec
- Reviewed Citrix's UX Input and prepared for next meeting