WebRTC based social video broadcasting

Shiva Ramaseshan	Krishnan Narayanan	Aman Sardana
sramases@asu.edu	kknaraya@asu.edu	asardan1@asu.edu

Abstract- WebRTC (Web Real-Time Communication) is an API definition drafted by the World Wide Web Consortium (W3C) that supports browser-to-browser applications for voice calling, video chat, and P2P file sharing without plugins[1]. Historically, RTC has been corporate and complex, requiring expensive audio and video technologies to be licensed or developed in house. Integrating RTC technology with existing content, data and services has been difficult and time consuming, particularly on the web.[2]

With WebRTC being developed as a web standard by relevant bodies such as IETF and W3C to ensure industry consensus, real time communication (RTC) on web applications should be as easy as entering text in a web page. It can be done without the need to download/update any plugins or extra piece of software. It would come with native support from the browser.

This project is about building a video broadcasting application using WebRTC on a social broadcasting web platform which will be used to launch the application. Our main task in this project would be the survey of this new technology WebRTC. Next, we need to think about how to do signaling to get the public IP address of the destination node so that a p2p connection could be setup. Finally, create a HTML5 responsive web application for this WebRTC app and use phonegap to convert this to mobile apps as well.

Keywords - WebRTC

INTRODUCTION

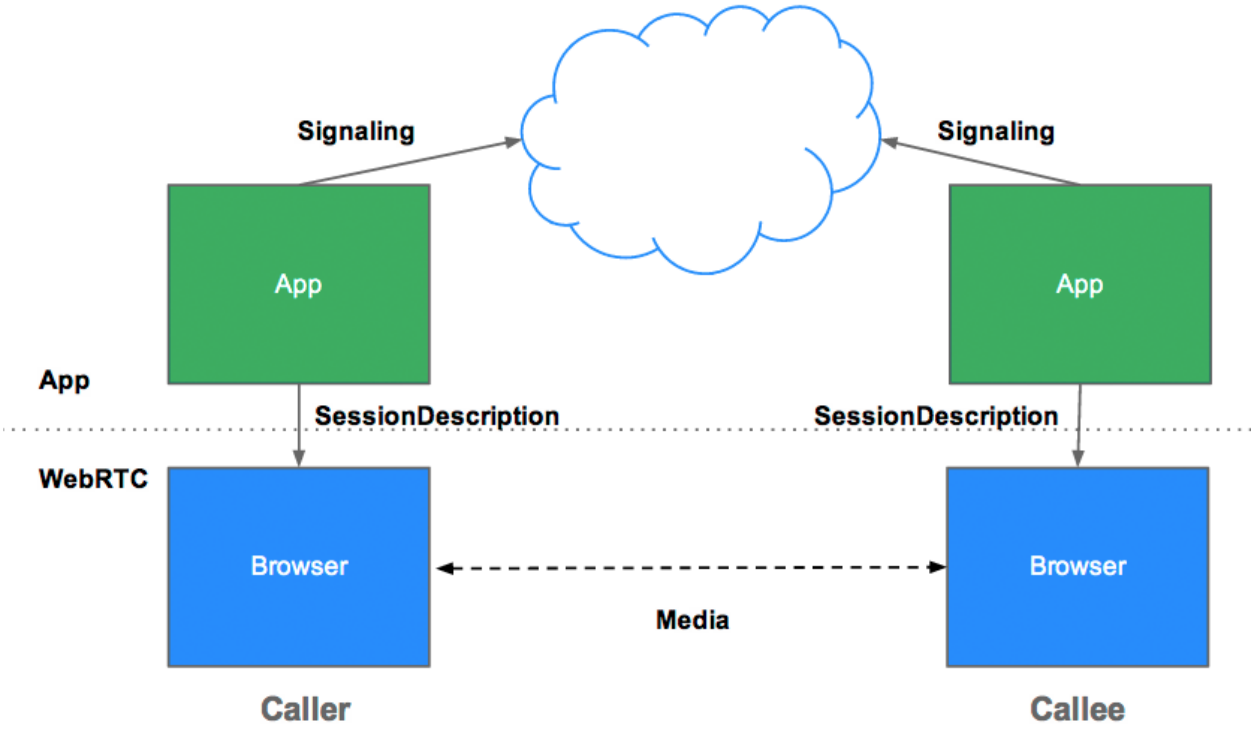
- **Problems to be addressed-** As WebRTC is just client side JavaScript API, it becomes a problem when the users are behind the NAT server. Then, it is not straightforward to get the public IP address of the destination node.
- **Why are these problems important?** - The above mentioned problem is important as most professional working environments would be behind a NAT server and not giving public IP. As WebRTC is just client side JavaScript API to be run on the browser, it needs public IP of the destination machine to establish a peer to peer connection.
- **Applied technologies & Solution--** WebRTC API, HTML5/CSS3, JS/jQuery, PHP, Node.js, ASU MobiCloud Environment, phonegap and MySQL. Our solution to the problem would be using STUN server to establish the connection. But the STUN server would work for about 80%-90% of the time. If the destination is behind the NAT server than we would need to employ TURN server to get the public IP address. The downside

with the TURN server is its cost, but there is a free and open-source TURN server available from google, which we plan to use.

- **Expected Outcome -** A social broadcasting web application with user profile management similar to Google™ Hangout, but the upside would be it would work without the need of any plugins. Android users would be able to download an app.
- **Project Management Plans -** We'll be driving the project into multiple phases, depending on the work. First phase would be the setup of user management in the site. The next would be setting up video conferencing using webRTC. Secondly, we would take care of signalling using Node.js. Thirdly, testing the project for bugs and focus on load testing as to figure out how many people are able to join the conference. All the work would be divided amongst all the group members such that each would have the chance to work on every technology used in the project and the load is balanced as well.

SYSTEM MODELS

A. System Model



The architecture described above is called JSEP, JavaScript Session Establishment Protocol. Once the signaling process has completed successfully, data can be streamed directly peer to peer, between the caller and callee—or if that fails, via an intermediary relay server. Streaming is the job of RTCPeerConnection[2].

B. Software and Languages

- webRTC API
- HTML5/CSS3
- Twitter Bootstrap
- JavaScript
- PHP
- Node.js / node packages express.js. socket.io
- MySQL database
- Chrome with ripple emulator extension

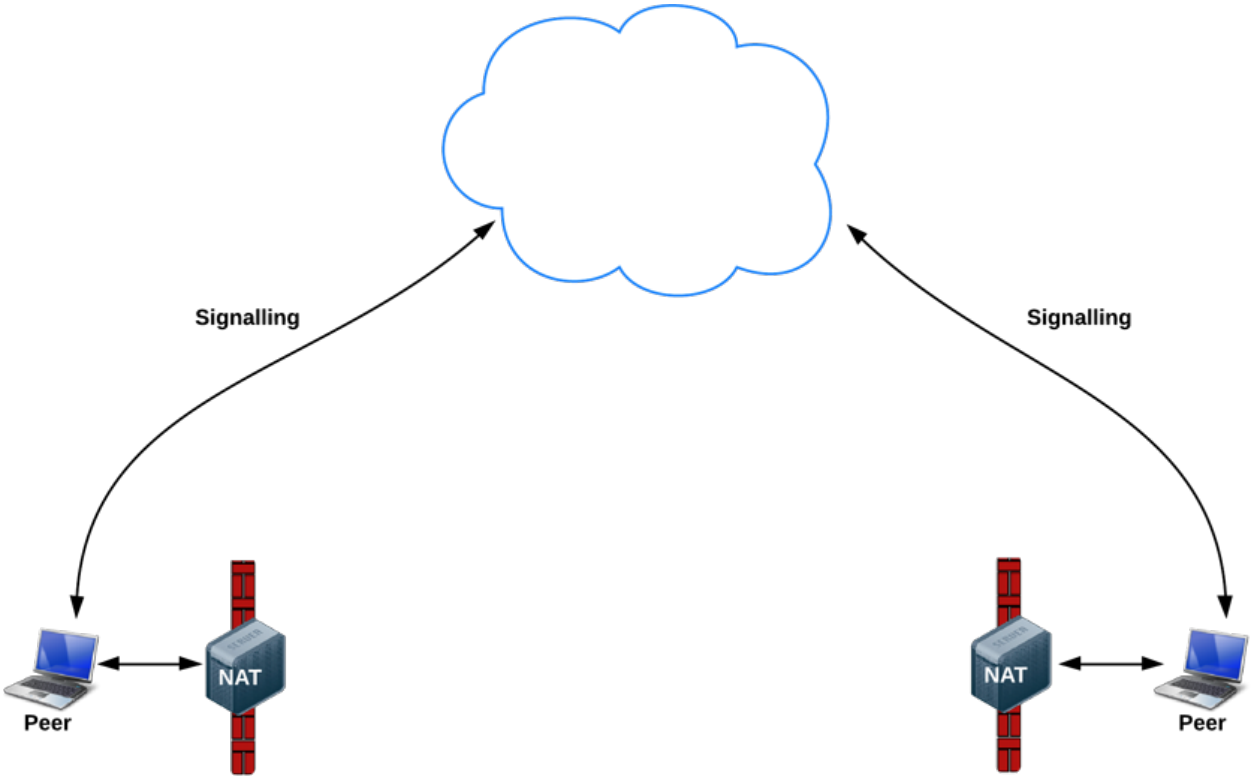
PROJECT DESCRIPTION

A. Project overview

The objective of this project is to create a social broadcasting application which allow users to broadcast & record the event which can be viewed by any user on the internet. The social platform will let the user choose the categories they are interested in and they will get notified about all the upcoming events of the selected categories. We can also provide an option for the user to sign in through facebook or google plus and have them send out invites to their friends.

The project is divided into six main tasks which includes Survey, Environment setup, Requirement Analysis, System design, Implementation, Testing.

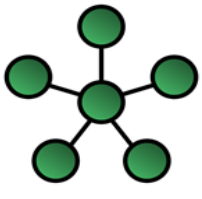
Survey
In this phase a detailed research on WebRTC, Signaling server, STUN server, TURN server is done. What are all the API's provided by WebRTC, what all we can use for signaling (signaling methods and protocols are not specified by WebRTC standards[2]), what all if any free STUN and TURN servers we can use for our project. Also find out about which network topology to use star vs mesh. Each has its merits and cons.



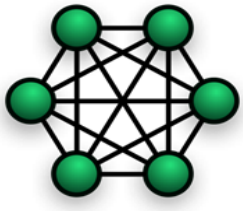
The real world

As we can see in the diagram above, In reality most devices live behind one or more layers of NAT, some have anti-virus software that blocks certain ports and protocols, and many are behind proxies and corporate firewalls. WebRTC apps can use the ICE framework to overcome the complexities of real-world networking. ICE tries to find the best path to connect peers. It tries all possibilities in parallel and chooses the most efficient option that works. ICE first tries to make a connection using the host address obtained from a device's operating system and network card; if that fails (which it will for devices behind NATs) ICE obtains an external address using a STUN server, and if that fails, traffic is routed via a TURN relay server.[2]

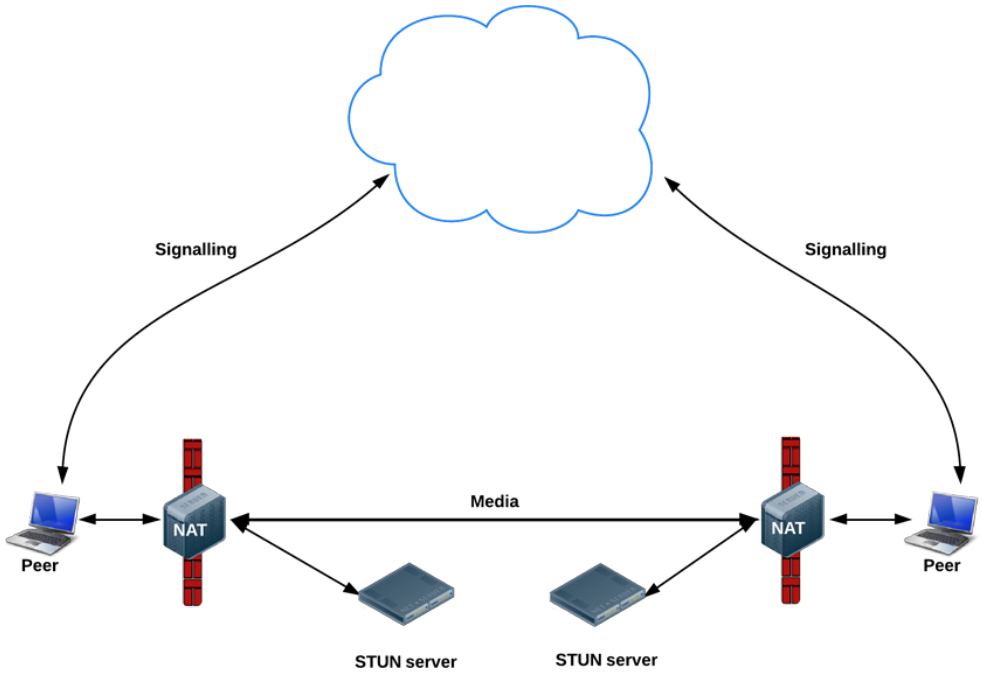
We can implement using either star or mesh topology. Using star we do not have to worry about overloading network bandwidth but everybody can't each other like in mesh.



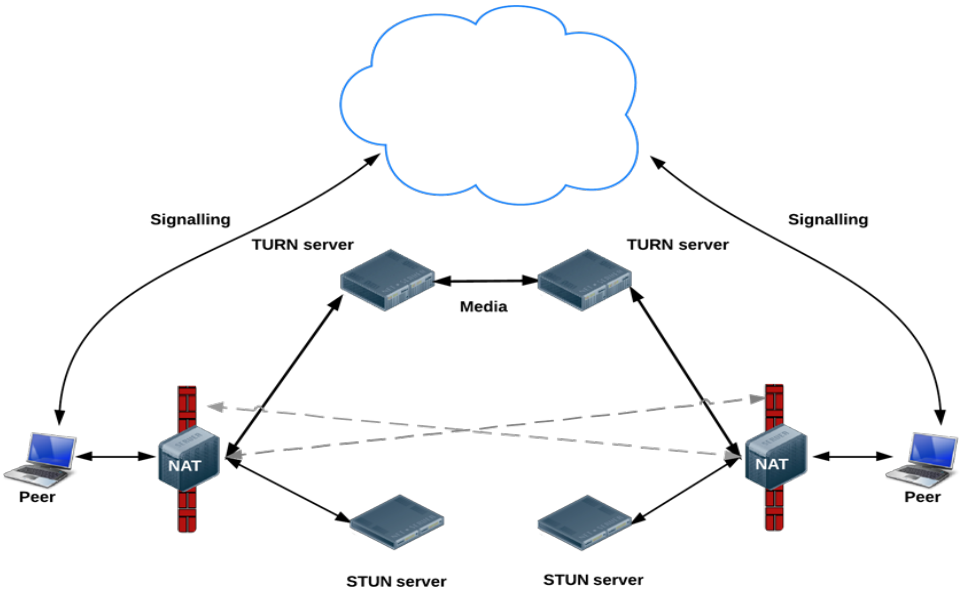
Star Toplogy



Mesh Toplogy



Using STUN servers to get public IP:port addresses[4]



The full Monty: STUN, TURN and signaling[4]

We built our signalling server using node.js with socket.io.js or else we can use ready made signalling servers like webRTC.io, easyRTC, Singalmaster.

Environment Setup
We request access to servers in the cloud environment. We would be in need of a couple of virtual machines. These would serve to be the basic necessities to start the project. Once this is completed, we move on to downloading and installing softwares on the virtual machines and the client side nodes. We would be making use of Node js, a browser that supports WebRTC preferably Google Chrome. Also this version of the browser should be compatible for HTML5 and CSS3.

Requirement Analysis
In this phase we would analyse the user needs and requirements for the project. Create a detailed functional requirements document.

System Design
In this phase a system is designed based on the proposed model which includes creating UML diagram which will help in identifying how the system will run and how entities interact with each other.

Implementation

- Task 1: HTML5 responsive web application

Make a responsive website using HTML5 and twitter bootstrap. The website should have login page and should show dashboard to the logged in users.

- Task 2: Setting up user management on the website

We come up with database design for the website. Write backend code in a scripting language like PHP to connect the website front end to the database.

- Task 3: Setting up the signaling server

Using node.js set up the signaling server or use ready made signalling servers which would handle the establishment of the p2p connection.

- Task 4: Setting up the STUN/TURN server

We plan on using the Google's free and open source STUN/TURN server and configuring that to our project setup.

Testing and Re-engineering
In this phase a set of test cases will be created to measure the efficiency and performance of the system. After running the test cases re-engineering will be done on the processes which needs improvement. This is an important phase in the project and will be repeated until all the project requirements are met.

B. Project Task Allocation
This is a 3 member group project. We are not planning on having any team lead sort of thing and everybody would be having equal responsibility. As, we are using GIT on the mobicloud for

the SCM, that would help us collaborate on the project as well. Each of us would pick up tasks and complete their part and push to GIT.

C. Deliverables

a. A customized web site for users to sign in (can sign in through Facebook or Google plus)
b. The users after signing in should be able to watch any video being broadcasted as well as broadcast the videos
c. The users before signing in has the privilege of watching pre recorded videos.
d. The chatroom can be interactive where the user can chat or raise a question and the broadcaster can respond to the user.
e. The signed in user has the option of sending out invites to his/her friends from either facebook or google plus.
f. The signed in user will have his/her own dashboard and profile page where the interests of the user will be listed and they would get notifications depending on those interests.

Software Packages, Tools and languages used for this project

- HTML5/CSS3
- Twitter Bootstrap
- JavaScript
- PHP
- Node.js / node packages express.js. socket.io
- MySQL database

Algorithms

- For signalling server using node.js and socket.io.js
- For recording the video and storing it.
- For breaking the storage of the video according to time ex 10min or 15 min video.
- To notify the user about new video being broadcasted according to his or her interest.

D. Project Timeline

Phases	Start Date	End Date	Duration (in days)
Prework	9/2	9/5	3
Development	9/5	10/1	25
Testing	10/1	10/6	5

Risk Management of the Project

- Nodes which are behind a NAT server will find it difficult to establish the connection. To avoid this, we would configure TURN server. If lot of traffic starts going to the TURN server then that might crash (Load Balancing is out of scope of the project). It would be a third party server that we would be using, so not having much control over that.
- WebRTC doesn't enforce a maximum limit on the number of connections which we can establish. So, as a result we cannot accurately predict the number of connections before which the system might start to hang. It totally depends on the bandwidth of the users in that session.
- WebRTC is a new technology and as it is with any new technology it is not that widely supported as of August 2014. It is supported on Chrome 23, Firefox 22 and Opera 18 browsers for desktop but not Safari and IE.
- As of August 2014, it is not yet fully supported across all mobile devices. The mobile browsers Chrome 28, Firefox 24 and Opera Mobile 12 supports webRTC for all android. There isn't much support for other mobile platforms.
- As of August 2014, it is not yet a complete nor stable, and as such is not yet suitable for commercial implementation.[3]
- Signaling mechanisms aren't defined by WebRTC standards, so it is up to us to make the signaling secure. If the attacker manages to hijack signaling, they can stop sessions, redirect connections and record, alter or inject content[2].

CONCLUSION

Summary - To conclude, this project is about user being able to do interactive video conferencing and being able to broadcast it to the public. There would be a recording available of the session so that the anybody can view it again later. The video conferencing would be achieved using webRTC and the web app hosted on ASU MobiCloud environment. We would have to use Node.js to do the signalling. Presently, it would be a star topology where the broadcaster sends video signal to everybody, so that everybody could see him, but rest not everybody can see everybody else. So this would be simpler in complexity than Google Hangouts but the main advantage of this project is that it would be plugin free. WebRTC is built right into the browser natively. Although till now Chrome, firefox and opera are the browsers to support this.

Future Extension - Each user (video broadcaster) can start having subscribers. The system would work much like the YouTube subscription where the subscriber gets notified about the new broadcast from the user. Users would be able to sign in with Facebook and Google+ accounts. When they do that they would be asked permission to send their friend's subscription request. Implementation of mesh topology where everybody can see and hear each other The

one who is currently speaking would come in the center of the screen and all others would be lined up at the bottom, much like Google Hangouts.

ACKNOWLEDGMENT

We would like to thank Professor. Dijiang Huang for the wonderful base idea about the project. We would also like to thank our mentor Yuli Deng for the inputs on the project proposal.

REFERENCES

[1] WebRTC (Wikipedia) Retrieved from http://en.wikipedia.org/wiki/WebRTC

[2] Getting Started with WebRTC - HTML5 Rocks (HTML5 Rocks) Retrieved from http://www.html5rocks.com/en/tutorials/webrtc/basics/

[3] WebRTC 1.0: Real-time Communication Between Browsers (WebRTC 1.0: Real-time Communication Between Browsers)
Retrieved from http://dev.w3.org/2011/webrtc/editor/webrtc.html

[4] WebRTC in the real world: STUN, TURN and signaling - HTML5 Rocks (HTML5 Rocks)
Retrieved from:-
http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/#after-signaling-using-ice-to-cope-with-nats-and-firewalls

[5] (Wikipedia) Retrieved from:-
http://en.wikipedia.org/wiki/Network_topology#Star

[6] (Wikipedia) Retrieved from:-
http://en.wikipedia.org/wiki/Network_topology#Mesh