CSN 503

ADVANCED COMPUTER NETWORKS

COURSE PROJECT – 2

A Report on

Design and implement a hybrid WebRTC signalling mechanism for unidirectional & bi-directional video conferencing

Group Members

Name	Enrollment no.
Nitin Gaurav Singh	15118049
Akhil KMV	15114038
Avinash Naik	15114040
Ankit Saybu	15114066

Aim of the paper

Create and implement a WebRTC hybrid signalling mechanism for video conferencing

What is WebRTC?

WebRTC stands for Web Real-Time Communication.

It is a technology that enables browser-to-browser communication.

WebRTC allows the transportation of audio, video and data .

How WebRTC works?

The three principal components of WebRTC are:

getUserMedia: allows a web browser to access the camera and microphone and to capture media

RTCPeerConnection: manages the peer-to-peer connection

RTCDataChannel: allows browsers to share arbitrary data

*** WebRTC does not specify any particular signalling mechanism or protocol between the client and the server. ***

But why need a signalling mechanism?

A signalling mechanism is the core of peer detection that coordinates the communication between users.

It starts exchanging media and supports the establishing communication among users.

Signalling connects the browser to a server and permits the participants to access this server.

So, a signalling mechanism has to be designed.

Which is nothing but the aim of this project!

Bottlenecks of the previously used methods

XMLHttpRequest (XHR/polling).
This leads to waste of bandwidth and delay, as long as the browser keeps polling for data regularly and the server continues responding even when no messages can be sent or received.

SIP (Session Initiation Protocol)
SIP has a high bandwidth consumption and delays and it also needs additional softwares for the servers and installation. So it was neither cost nor time effective.

Features of the Project

The project presents a novel signalling mechanism among different users, devices and networks to offer multi-party video conferencing.

In this project, WebNSM (WebRTC hybrid signalling mechanism) was created for video conferencing based on RTCPeerConnection (API) using socket.io mechanism to connect between each of the browsers.

Socket.io (API) offeres real-time bi-directional communication between a client and a server.

RTCPeerConnection (API) is an array of URL objects that send any ICE (Interactive Connectivity Establishment) candidates to the other peer, handles the video stream, and starts offer/answer negotiation process, etc.

There are three levels of users of this project:

Broadcast head : This is the room initiator. It can see the video of all the Broadcast entities in its room.

Broadcast entity : This user joins the room initiated by the Broadcast head. It can only see the video of the Broadcast head of that room.

Viewer : It can see the Broadcast head and all the Broadcast entities of a particular room but no one can see it.

And these topologies offer the following communications at the same time: (Hybrid nature)

one-to-one -> 1 Broadcast head to 1 Viewer (unidirectional)
1 Broadcast head to 1 Broadcast entity (bidirectional)

one-to-many -> 1 Broadcast head to many Viewers (unidirectional)
1 Broadcast head to many Broadcast entity (bidirectional)

many-to-one -> A Broadcast head and many Broadcast entities to 1 Viewer (unidirectional)
Many Broadcast entities to 1 Broadcast head (bidirectional)

many-to-many -> Many Broadcast entities to many Viewers (unidirectional)

Applications of this Project

WebNSM is useful to be used for various communications such as:

In Healthcare : many doctors can communicate with many technicians and patients

In e-learning : many teachers can communicate with many students and many students can in turn communicate with others

There are many more such communication applications because it gives a user a full flexibility to use appropriate topology according to its resources.

Method of Implementation

This implementation can be divided into following:

Setup a Browser Web Page

The main HTML (web page) of this experiment was programmed using JavaScript to set up many features, such as ***opening room, mute-audio/video, using full-screen and using volume slider***.

In the beginning, to open a room there must always be a room initiator.

The participants are free to select "Viewer" to watch and listen to the broadcaster head and entities or select "Broadcaster" to set up bidirectional video conferencing with the Broadcast head.

In this application, communication has one initiator and different peers as viewers and broadcasters.

When the room is opened, it will arbitrarily audio and video to present MediaStream, which can be obtained using navigator.getUserMedia() method to create a synchronised video and audio.

After getUserMedia, a web browser will request permission to access the camera and microphone to capture peer’s screen. A camera will start streaming when the permission is given. Now the application is ready for other peers to join the room.

On the other hand, when peers would like to be as viewers they do not need to invoke their camera and microphone, while they will only receive videos.

These steps of opening/joining the room applies to every peer, as well as stopping the streaming of their camera/microphone without influencing on the rest.

Implementing the WebNSM

This signalling must occur before a Peer-to-Peer (P2P) connection can be occurred. WebNSM was created using RTCPeerConnection API and socket.io (API) mechanism for an instant handshake.

Therefore, WebNSM must be carried out before streaming can begin between peers. It relies on offer and answers negotiation process to describe the SDP (Session Description Protocol) of the session. The offerer is a peer who initiates the session to connect other peers. In contrast, the answerer is asked for connection from the offerer. The offerer is assumed to know the answerer's URL and then requests a connection through WebNSM. When the initiator opens the main room, WebNSM will be ready to support any offerer and detect a room presence. Thus, several functions and steps have been employed to create it.

First of all, it should transmit the data as a String and setup a default channel passed through constructor using "connection.channel = channel || RMCDefaultChannel". Additionally, it connects with a signalling channel when only the first participant is found using invoke "getUserMedia" then initRTCMultiSession function.

WebNSM was built to accomplish many characteristics, such as determining the room initiator "connection.initiator = true", allowing a single user to join a room "connection.join = joinSession", hearing new user with existing participants on New Participant (response), participants are shared with a single user or with all users, if the initiator disconnects sockets, participants should also disconnect, close the entire session, reject user-id, disconnect for all, open private socket that is used to receive offer-sdp "newPrivateSocket" and ask other users to create offer-sdp and function PeerConnection.

They also utilise RTC (Real Time Connection) to send data "connection.send = function(data, _channel)", initialize "RTCMultiSession" which is the backbone object. The custom devices are selected and screen_constraints, such as screen.width, screen.height. Participants also check if the screencapturing extension is installed.

When a stream is stopped, it must be removed from "attachStreams" array to allow re-capturing of the screen, if the muted stream is negotiated, audio/video are fired earlier than screen, stop local stream “if (response.stopped)”, stop remote stream, “if (response.promptStreamStop)”, create an offer SDP using "createOffer()" function, create answer SDP using "createAnswer()" function, createDescription() function, getBrowserInfo() function, construct a new RTCPeerConnection, trigger the stun server request, match just the IP address, remove duplicates, listen for candidate events and etc.

To establish a peer-to-peer connection, both clients need to create an RTCPeerConnection object. Then, each peer needs to obtain their Session Description, an object that indicates what kind of data they want to send to the other client through the connection and what they can do by built-in methods of the RTCPeerConnection object. Thus, the offerer will send the request to the answerer for the availability, including SDP offer to receive audio and video. The answerer (initiator/broadcaster) will receive the request and sends a confirmation of the availability as "room is active" with the SDP constraints to receive audio and video. The offerer gets the remote stream and creates an offer using "getLocalDescription" with RTCPeerConnection. Additionally, the offerer creates DataChannel method which is added to the RTCPeerConnection to create an "RTCDataChannel" object. When an "RTCDataChannel" on the offerer's side is generated, the offerer invokes "createOffer" of RTCPeerConnection, thereby enabling "createOffer" to return an offerer's SDP message. The offerer enables the SDP-offer message by setting various information and send them through WebNSM.

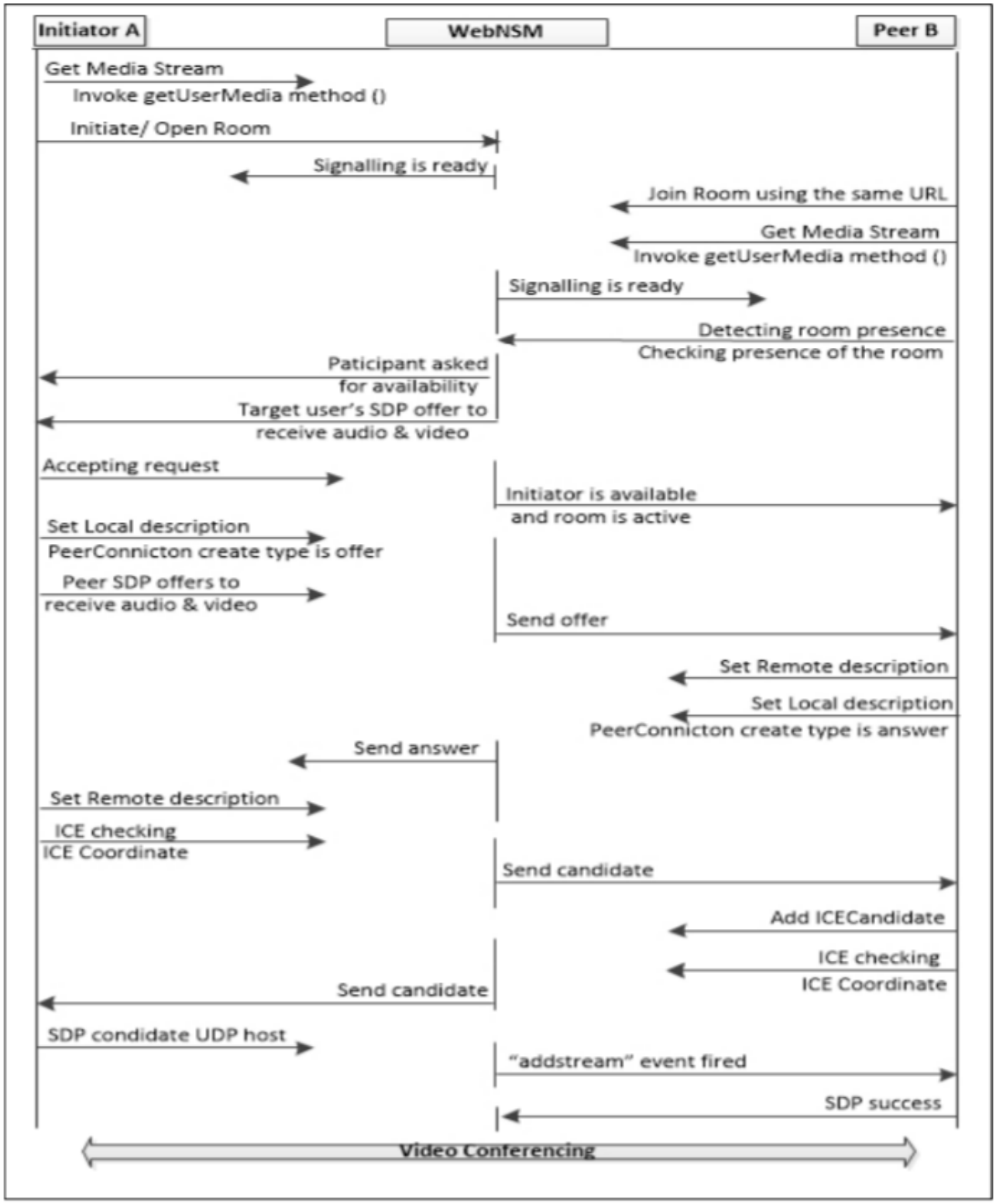
For instance, bandwidth information, using the period audio and video codecs, etc. Additionally, both the offerer and answerer change WebNSM state to "stable", to realise that there is no offer/answer exchange in progress. Once the "SDP-offer" message reaches the answerer through WebNSM, the answerer also initiates its RTCPeerConnection instance to accept the request. The answerer uses the "SDP-offer" into its RTCPeerConnection to create an "SDP-answer" and then forward it to the offrer.

Also, the two clients need to exchange information about communication methods that they can use to reach each other. These communication methods are known as ICE Candidates and they will be exchanged through the WebNSM. Now the answerer and offerer are able to respond and they both configure the Real Time Communication (RTC) packets transported.

After two peers exchange SDP-offer/answer and ICE candidates, they can create their session. The answerer and offerer "add SDP" to candidate UDP by the host IP for both of them. The other participants can join the session based on similar steps. According to a communication as viewers, when an initiator is active for streaming, a peer is able to accede the room as a viewer after detecting a room presence using WebNSM. WebNSM sends a notification to the initiator that " a participant has asked for availability and the target has no stream.

In other words, it is a unidirectional video conferencing from an initiator to a viewer. An initiator receives a request and sends a confirmation of the availability as "room is active" with the SDP constraints. Thus, an initiator has started broadcasting the audio and video to the viewer.

In contrast, if there are other broadcasters, a viewer will communicate all of them, so the viewer can communicate all broadcasters by receiving their audio and video at the same time.

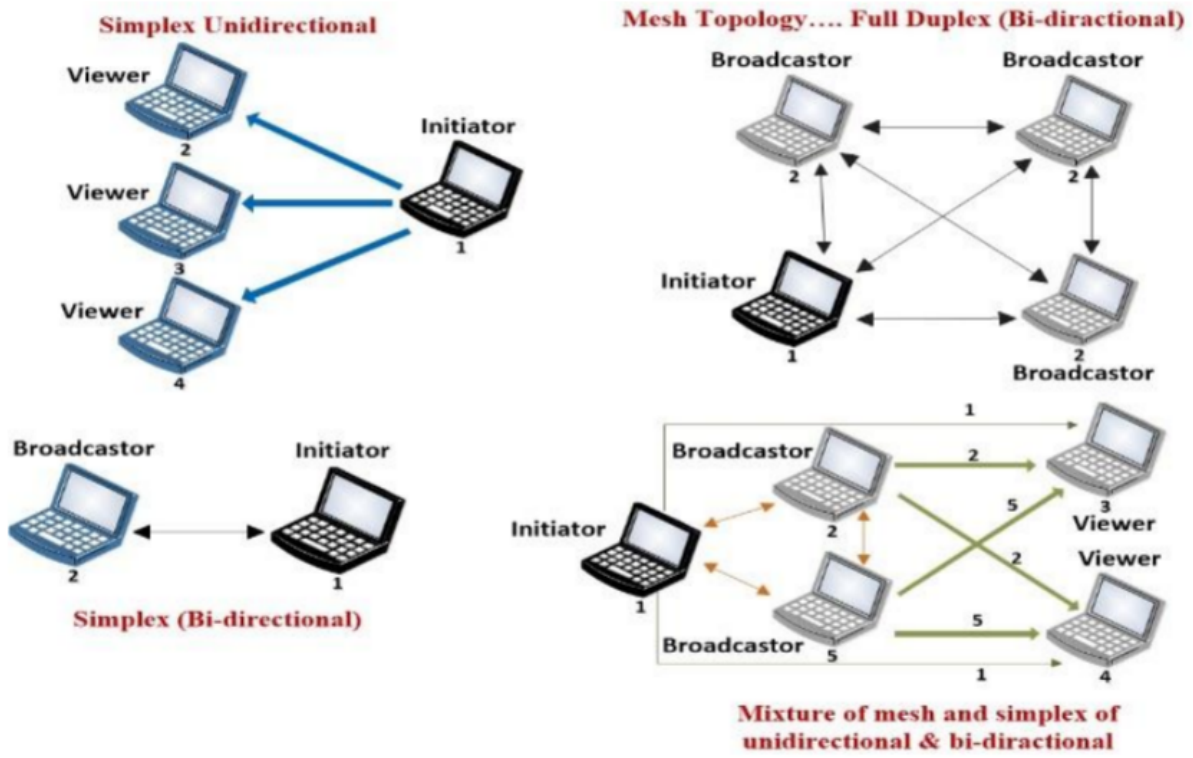


Presents the signalling between entities

Hybrid Topology

A host peer should initiate and start its browser to allow any user to participate in the session at any time without affecting the remaining participants, so using different systems allowing all peers to connect with each other as viewers and broadcasters to transmitted data from different devices simultaneously.

A hybrid uses different topologies and gives the users flexibility, reliability and multi-choice of communications such as initiator, broadcaster or viewer. Moreover, it allows several resources such as devices, networks and users to obtain video conferencing without any registration, downloading or installation and can be used in different applications. Using this scenario shows that it built a strong WebRTC application that works across multiple browsers, networks and topologies.



The architecture of the hybrid system

How to run the project

Install node.js on the system

sudo apt-get install nodejs

Also install npm which is a Node.js pachage manager

sudo apt-get install npm

Install RTCMultiConnection using npm

npm install rtcmulticonnection

This creates *node_modules/rtcmulticonnection* in the Home directory.

Then add the project files in the above folder.

To set up a server:

Edit the "socketURL" and "port" in the *config.json* file to the particular server ip address and server port number.

Also edit connection.socketURL value in the *index.html* file to the value same as above.

Then run the server on your machine by executing the following command in the same folder

node server --ssl

This makes the project up and running on the server.

So, to access the project just go to its url.

Now, if you want to be a broadcast initiator, enter a room number and click “Broadcast”.

And if anyone wants to join as a broadcast entity to a room, he can just enter that particular room number and click “Broadcast”.

And if anyone just wants to be a Viewer, he just needs to enter a particular room number and click “Viewer”.

Results and Conclusion

We successfully designed and implemented a hybrid WebRTC signalling mechanism for unidirectional & bi-directional video conferencing.

We implemented a signalling mechanism among different users, devices and networks to offer multi-party video conferencing using various topologies at the same time.

WebNSM can be considered as a novel signalling mechanism while it presents a flexible communication among users.

Moreover, this can be applied in different applications, such as get a group of people together on one call at the same time, conferencing among users, entertainment. e-Learning between teacher and students, m-Health among patients and doctor or specialist and technicians, etc.

In the future, there is an intention to expand this work over more scalable video conferencing using MATLAB simulator to discover the effectiveness of resources in WebRTC.

Our Remarks

We really enjoyed doing it.

At the same time, it was quite a challenge.

Finally, we are happy to have finished it while getting to learn a lot of things about the networking.