Internet Technology for Mobile Computing Final Report SpaceCall: A Real Time Audio Call Spaceify Application

Dmytro Arbuzin 415181
Erik Berdonces Bonelo 463935
Binghui Liu 465580

1. Aims and objectives

The goal of the project was to develop a conference-voice system for indoor spaces to allow users interact with each other using smartphones as microphones.

The core requirements for the systems were:

- cross-platform compatibility;
- user friendliness to allow easy one-time usage without installing additional applications;
- ability to run under spaceify environment;

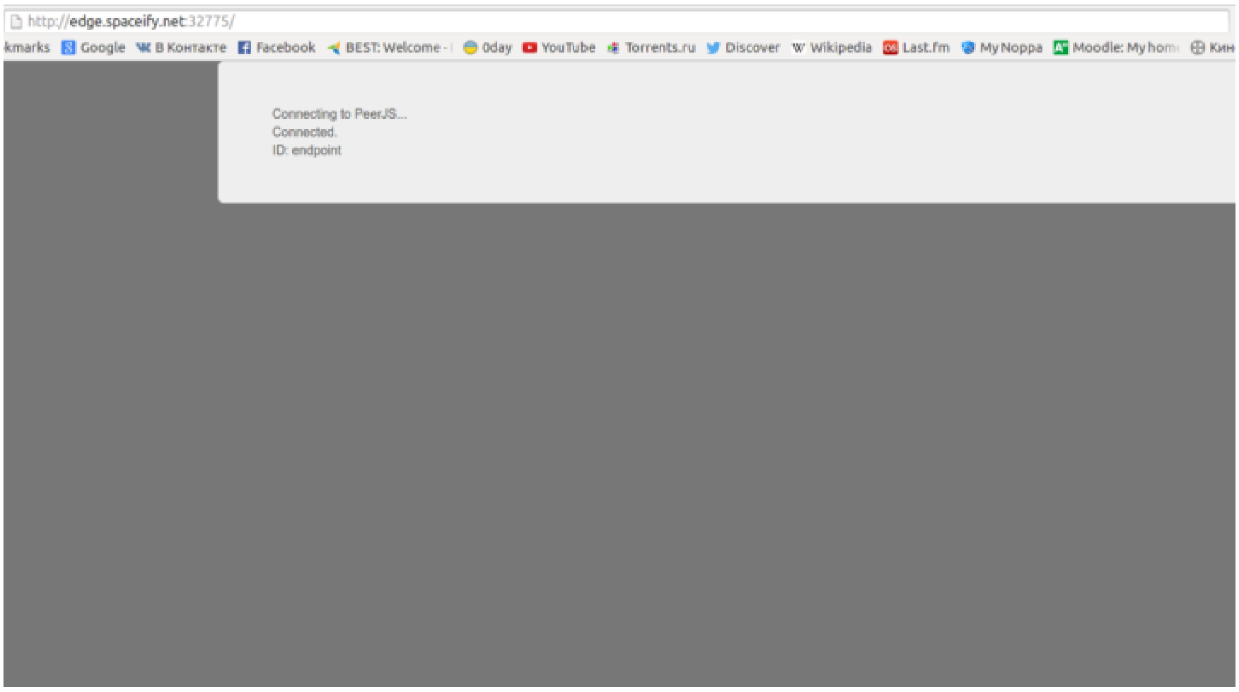
2. Work Practice

Our initial goals is to build a system that should consist three parts: one server, one middleware and one client. The server was planned to be based on Asterisk / VoIP Server that processes audio clips and handle the connections from the clients. However after doing some research on Asterisk, we found it quite heavy loaded, especially the installation requires a computer than “EVERYTHING ON THIS COMPUTER WILL BE DELETED AND REPLACED WITH THE AsteriskNOW DISTRO.”, which we are not able to

provide. So we made researches on other possible frameworks or services, and eventually switched our solution to PeerJS 0.3.5 (http://peerjs.com), a light weight WebRTC framework.
WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs (http://www.webrtc.org) supported by various modern mobile browsers. It supports text, audio and video. PeerJS wraps the browser's WebRTC implementation to provide an easier-to-use API.
PeerJS consists two parts: a server and a client. Whenever a client is connect to the server, it will be assigned with a unique ID, and then two clients can connect peer-to-peer identifying by this unique ID.
The server is deployed locally inside the virtual machine of Spaceify, and its basic function is only to assign an ID for any incoming client connections. Note even though an ID, ought to be a unique random number, we have manually set the ID for the loudspeaker as "Endpoint".

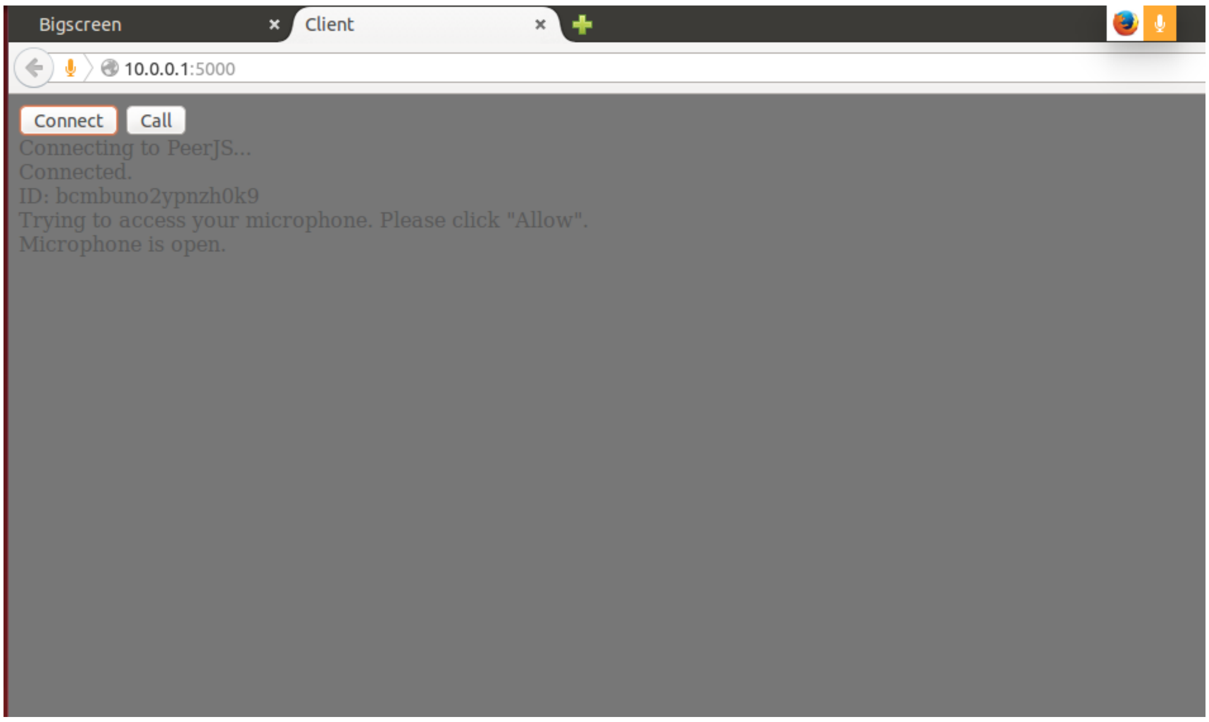
3. Implementation

Our project is a web application hosted in the Spaceify environment. The goal is to allow users interact with each other using their smartphones as microphones. The end side of this application is a speaker, which receives and plays all the incoming audio data from the other clients it has been connected to. The user side is a client application, with two buttons, CONNECT and CALL. "CONNECT" will connect the user to the endpoint, which is the speaker, then "CALL" will perform calling the speaker, and transfer any audio incoming collected by the microphone to the speaker via WebRTC in real time, so the speaker will be able to broadcast any user's audio/voice in a collaborative way. Since the whole system is planted in the Spaceify environment, all the users are connecting through a local network.


http://edge.spaceify.net:32775/
kmarks Google В Контакте Facebook BEST: Welcome 0day YouTube Torrents.ru Discover Wikipedia Last.fm My Noppa Moodle: My hom

```
Connecting to PeerJS...
Connected.
ID: endpoint
```

Fig. 3.1 - Screenshot from the speaker projected to BigScreen app



```
Connecting to PeerJS...
Connected.
ID: bcmbuno2ypnzh0k9
Trying to access your microphone. Please click "Allow".
Microphone is open.
```

Fig. 3.2 - Screenshot from the client connected to the speaker locally

For the endpoint (the speaker), it will conduct the following workflow:

- Connect to the PeerJS server, to register itself as “endpoint”
- Wait for incoming calls from other peers
- Get the ID of the peer that makes the call, maintain a peer list
- Get the incoming call’s stream
- Inject a <audio autoplay> HTML5 component into the webpage, using it to play the stream

The workflow of a client is somehow symmetric to the speaker, but instead of waiting for an incoming call, it will start an outgoing call, which is calling the destination ID (in our case is endpoint) with an audio stream collected from the microphone.

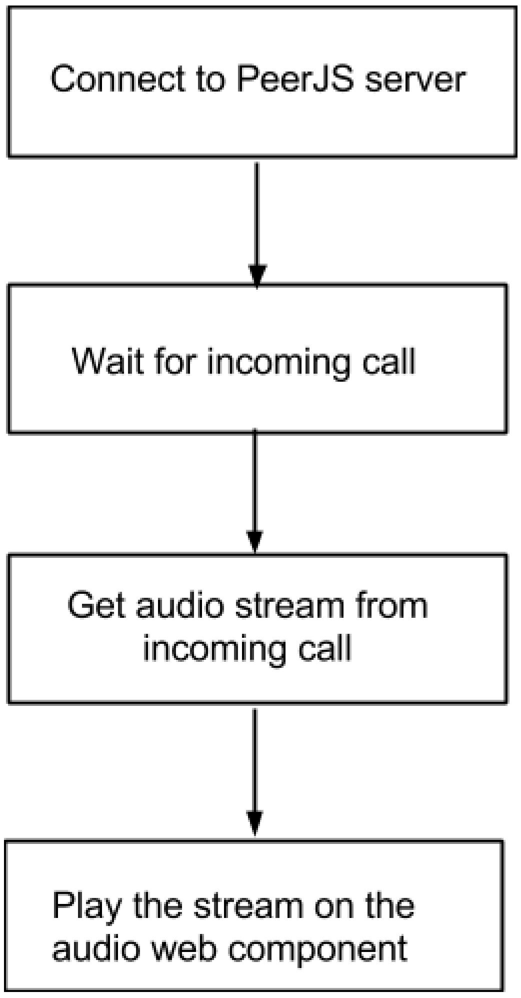
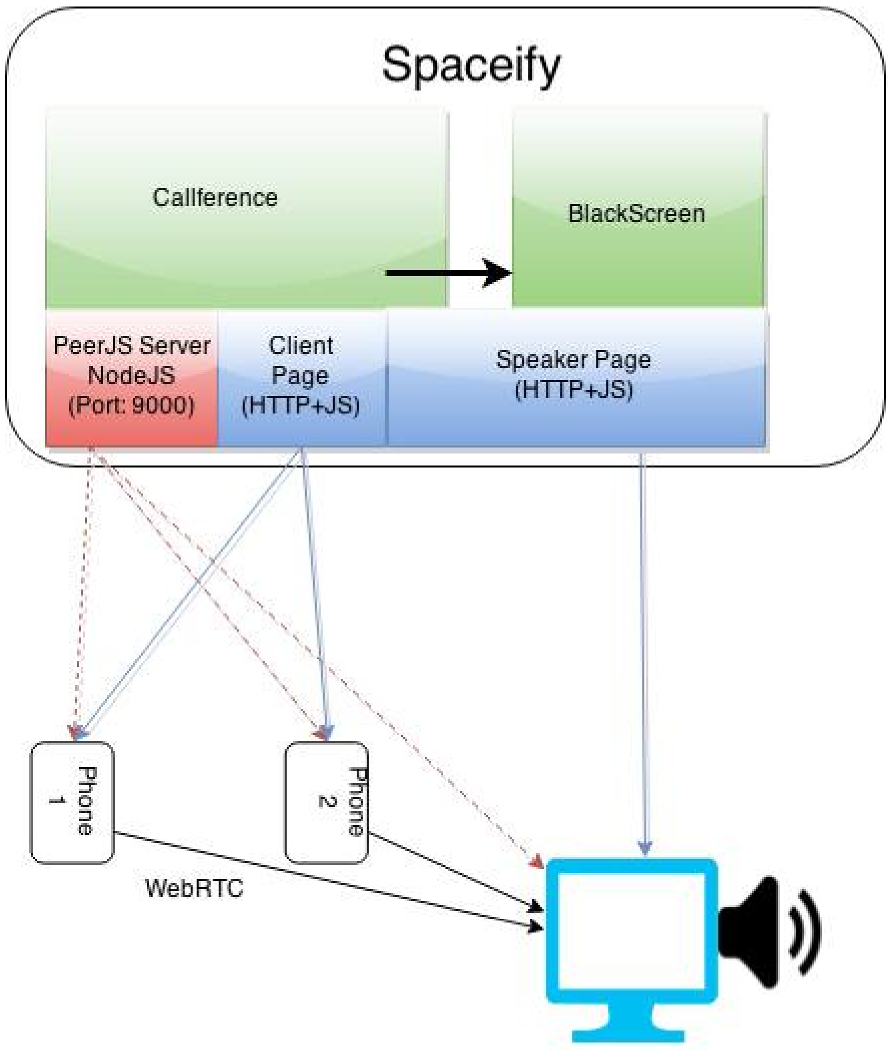

Connect to PeerJS server
Wait for incoming call
Get audio stream from incoming call
Play the stream on the audio web component

Fig 3.3 - Speaker Workflow

4. System architecture



5. Spaceify environment and installation

To integrate our system system to spaceify environment we have used the provided sandbox application urlviewer (https://github.com/ptesavol/urlviewer) and changed the tile interface to a button which opens the hardcoded link of locally running speaker (localhost:5000/speaker.html).

Installation:

Inside spaceify VM do next:

```
cd $HOME
git clone -b spaceify --single-branch https://github.com/Sturgelose/CallFerence
cd CallFerence
sudo spm install /home/spaceify/CallFerence/volume/
sudo service spaceify restart
```

Usage:

Unfortunately spaceify doesn’t support starting multiple servers from the manifest file, so you need to start servers manually:

```
cd /home/spaceify/Callference/volume/application/
sudo peerjs --port 9000 --key peerjs & node server.js &
Go to edge.spaceify.net with Firefox, and open a BigScreen
Click “Start CallFerence” button at edge.spaceify.net
Clients can connect directly to localhost:5000 port, in our case it is 10.0.0.1:5000
```

6. Limitations

WebRTC is a new technology and it’s not supported by major mobile platforms, currently it’s working properly only on Android with Google Chrome browser. iOS and Windows Phone 8 won’t work.