
Connecting Clients Together

Connecting Clients Together

- Now that we have implemented our own signaling server, it is time to build an application to utilize its power.
- In this session, we are going to build a client application that allows two users on separate machines to connect and communicate in real time using WebRTC.
- By the end of this session, we will have a well-designed working example of what most WebRTC applications function like.

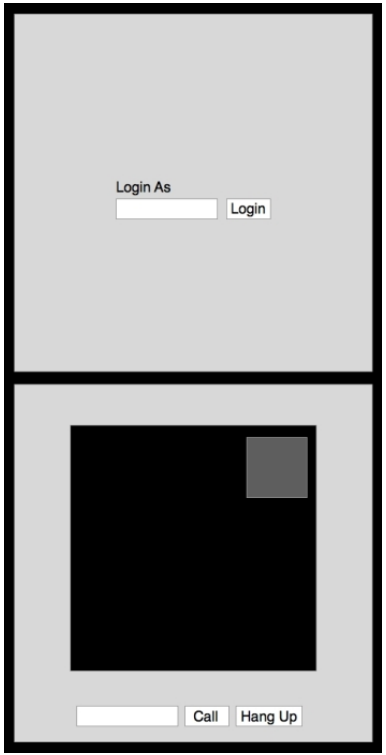
Connecting Clients Together

In this session, we will cover the following topics:

- Getting a connection to our server from a client
- Identifying users on each end of the connection
- Initiating a call between two remote users
- Hanging up a finished call

The client application

- The goal of the client application is to enable two users to connect and communicate with each other from different locations.
- This is often seen as the hello world of WebRTC applications, and many examples of this type of application can be seen around the Web and at WebRTC-based conferences and events.
- Chances are you have used something much similar before to what we will build in this session.



Setting up the page

- To start, we need to create a basic HTML page.
- The following is the boilerplate code used to give us something to work from.
- Copy this code into your own index.html document:

Refer to the FILE 5_1.txt

Getting a connection

- We can start by creating the client.js file that our HTML page includes.

- You can add the following connection code:

Refer to the FILE 5_2.txt

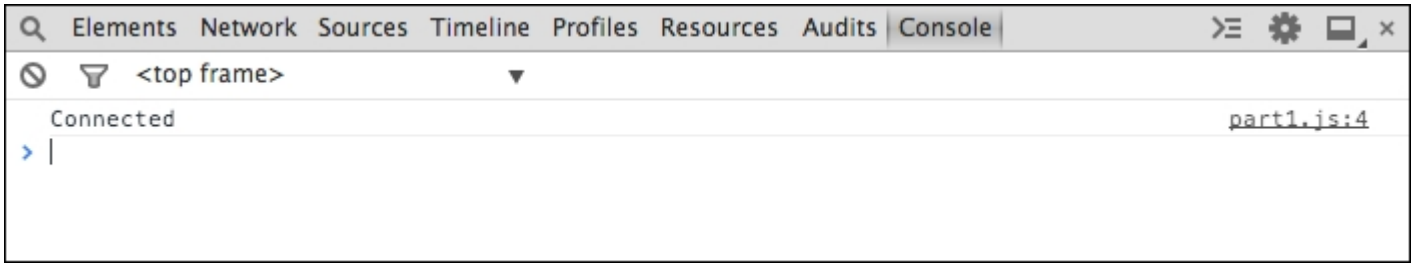
- The initial thing we do is set up the connection to our server. We do this by passing in the location of our server, including the ws:// protocol prefix on our URI

Getting a connection

- Next, we set up a series of event handlers.
- The main one to take note of is the onmessage handler, which is where we will get all of our WebRTC-based messages.
- The switch method calls different functions based on the message type, which we will fill out later in this session.

Getting a connection

- When you open this file now, you should see a simple connection message:



Logging in to the application

- To implement this, we need to add a bit of functionality to our application's script file.

- You can add the following to your JavaScript:

Refer to the FILE 5_3.txt

Logging in to the application

- Initially, we select some references to the elements on the page so that we can interact with the user and provide feedback in various ways.

- We then tell the callPage area to hide itself so that the user is just presented with the login flow.

- Then, we add a listener to the Login button so that when the user selects it, we send a message to the server telling it to log in.

Logging in to the application

Starting a peer connection

The startConnection function is the first part of any WebRTC connection. Since this entire process is not reliant on another person to connect to, we can set this step up ahead, before the user has actually tried to call anyone. The steps included are:

1. Obtaining a video stream from the camera.
2. Verifying that the user's browser supports WebRTC.
3. Creating the RTCPeerConnection object.

Logging in to the application

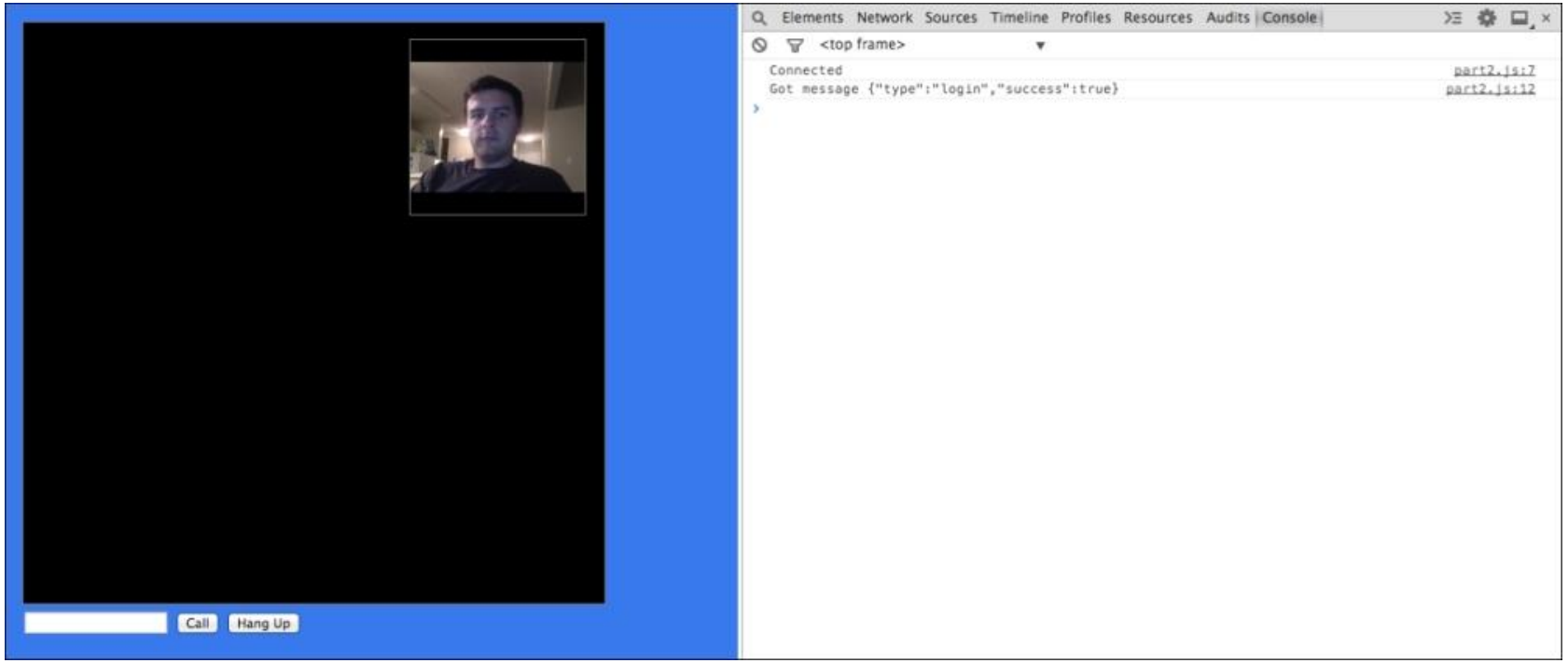
- This is implemented by the following JavaScript:

Refer to the FILE 5_4.txt

- This should all look rather familiar by now. Much of this code has been copied from the examples explained previously, Creating a Basic WebRTC Application.

- As always, check for the correct browser prefixes and handle any errors accordingly.

Logging in to the application



Initiating a call

- Now that we have set everything up properly, we are ready to initiate a call with a remote user.
- Sending the offer to another user starts all this.
- Once a user gets the offer, he/she will create a response and start trading ICE candidates, until he/she successfully connects.
- To accomplish this, we add the following code to our script:

Refer to the FILE 5_5.txt

Initiating a call

Inspecting the traffic

- Debugging a real-time application can be a tough process. With many things happening all at once, it is hard to build an entire picture of what is going on at any given moment.
- This is where using a modern browser with the WebSocket protocol really shines.
- Most browsers today will have some way to not only see the WebSocket connection to the server, but also inspect every packet that gets sent across the wire.

Initiating a call

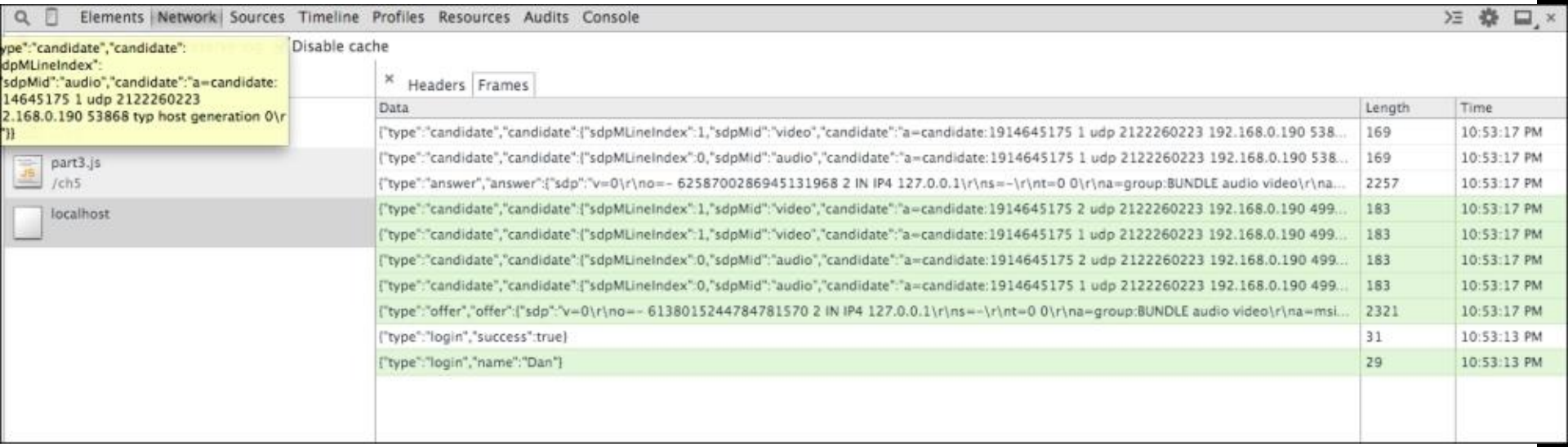
- In my example, I use Chrome to inspect the traffic. Opening up the debugging tools by navigating to View | Developer | Developer Tools, will give me access to an array of tools for debugging web applications.

- Opening the Network tab will then show all of the network traffic that has been made by the page.

- If you do not see any network traffic, refresh the page with Developer Tools open.

Initiating a call

It shows every packet sent in human-readable format for easy debugging.



Initiating a call

- There are also many other ways to get this information from the computer.
- Using the console output on both the server and client are great ways to get small pieces of information.
- You can also look into using network proxies and packet interceptors to intercept the packets being sent from the browser.

Hanging up a call

- The last feature we will implement is the ability to hang up an in-progress call.

- This will notify the other user of our intention to close the call and stop transmitting information. It will take just a few additional lines to our JavaScript:

Refer to the FILE 5_6.txt

Hanging up a call

1. First off, we need to notify our server that we are no longer communicating.

2. Secondly, we need to tell RTCPeerConnection to close, and this will stop transmitting our stream data to the other user.

3. Finally, we tell the connection to set up again. This instantiates our connections to the open state so that we can accept new calls.

A complete WebRTC client

- The following is the entire JavaScript code used in our client application.

- This includes all the code to hook up the UI, connect to the signaling server, and initiate a WebRTC connection with another user:

Refer to the FILE 5_7.txt

Improving the application

- What we have built over the course of this section is an adequate place to jump off into bigger and better things.
- It provides a baseline of features that almost every peer-to-peer communication application needs.
- From here, it is a matter of adding on common web application features to enhance the experience.

Improving the application

- On top of this, the application will need to be foolproof to ensure the best possible experience.
- User input should be checked at each part of the way by both the client and server.
- Also, there are several places where the WebRTC connection can fail, such as not supporting the technology, not being able to traverse firewalls, and not having enough bandwidth to stream a video call.

Summary

- After completing this session, you should take a step back and congratulate yourself for making it this far.
- Over the course of this, we have brought the entire first half of the course into perspective with a full-fledged WebRTC application.
- With how complex peer-to-peer connections can be, it is amazing that we have been able to make one successfully in just five short course.
- You can now put down that chat client and use your own hand-built solution for communicating with people all over the world!

Summary

- If there was any point where you can take a break and put down this course, now would be the time.
- This application is a jumping off point to start prototyping your own WebRTC application and adding new innovative features.
- After reading this far, it is also a good idea to research other WebRTC applications on the Web and the approach they took when developing.