CS 8803 NM – Networking for Multimedia

Project 3

Video Conferencing

Due April 21, 2022 at 11:59pm

Submit on Canvas

Overall Objective

This project's goal is to 1) make you understand the workings of a simple two-user video conferencing application running entirely within a single device, and 2) show how you can deploy the signaling server "in the cloud" which allows the users to be anywhere.

READ EVERYTHING CAREFULLY – there are a lot of details and pointers.

Deliverables are underlined and colored red.

Part 1 – Deploying and analyzing a simple 2-peer video

To be ready for this part of the project you will need:

1- Download and install Node.js. You can download this from https://nodejs.org/en/. Installation instructions will depend on your operating system.
   - It will be a lot easier if you can put node.js in your path so you can just call it without specifying the full path – this will depend on your OS as well.
   - You can test if your node installation is working properly by following the simple example here https://www.w3schools.com/nodejs/nodejs_get_started.asp

2- You will also need Wireshark. You can get it from Wireshark · Go Deep.
   - If you've never used wireshark before here is a good hands-on introduction. http://www-net.cs.umass.edu/wireshark-labs/Wireshark_Intro_v8.1.docx

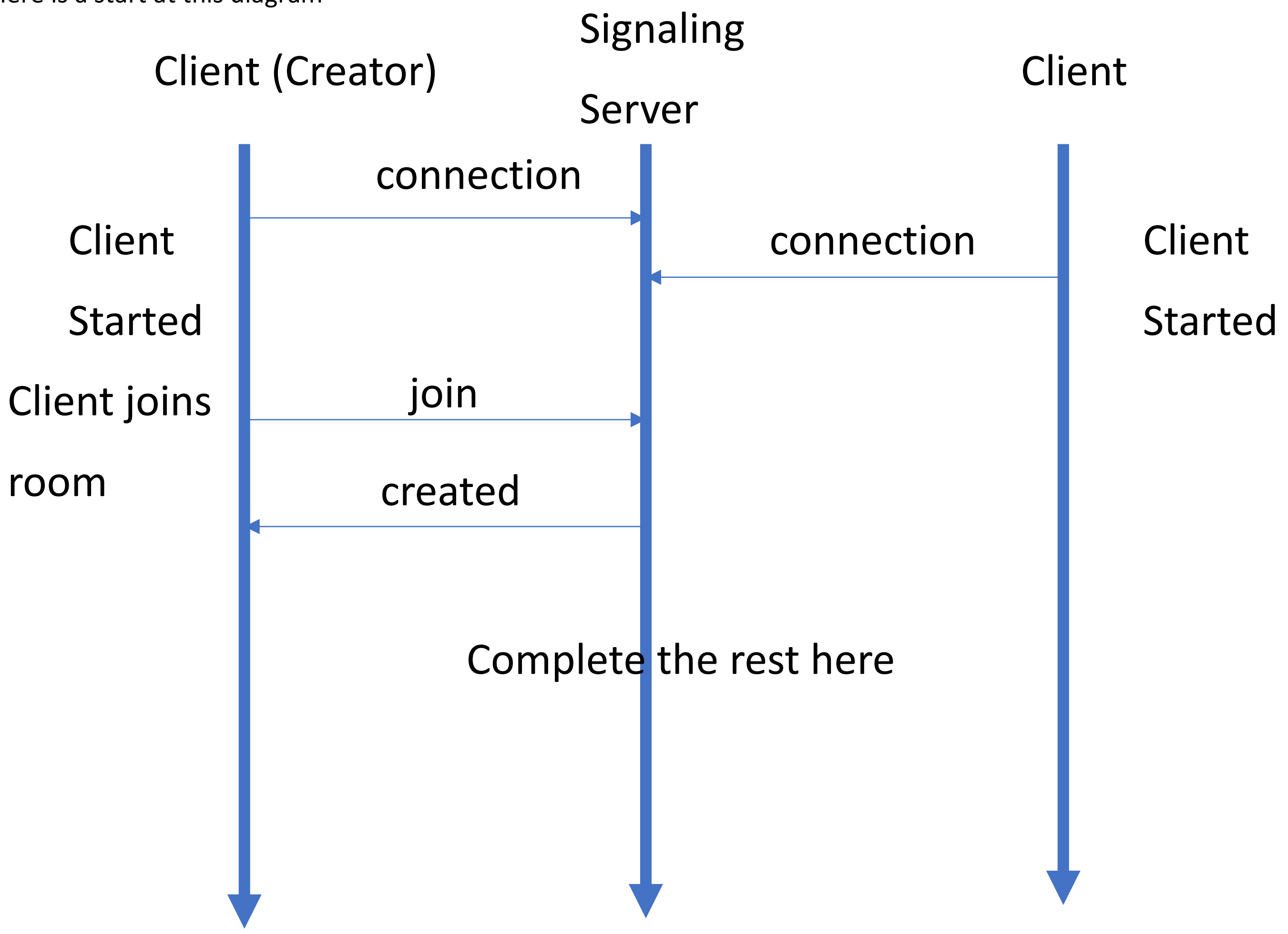
The main objective of this part of the project is to run the simple 2-peer video conferencing application here Video-Chat: A Video Chat Application made using WebRTC (github.com)

Step 1- The first step is to actually run this application – as we did in class. You will need to 1) download the code from github, 2) Unzip it into a folder, and 3) run it like we did in class.

- In a terminal within vscode, run index.js in node
- Point your browser to localhost:4000 in two separate tabs
- Join the same video session in both
- You should see yourself twice in each window
  (tip: make sure to mute your audio speakers – otherwise you will get some serious feedback!)

Step 2 – Analyze the code and draw a sequence diagram of the signaling that is taking place between the clients and the server. *Note that we expect you will do this only from an analysis of the code without actually running it.*

Here is a start at this diagram



Step 3 - Re-run the application from the beginning. This time while wireshark is running and *listening on the loopback interface*. Can you map the signals to specific packets?

A few tips here:

- Filter the Wireshark trace to only see packets going to and from and to port 4000 and remember the signaling is going over tcp. "tcp.port == 4000"
- Look for the actual signal words (e.g. "created") in the packet payloads. Why is it that you can read these words only in the data going from the server to the clients but not the other way round? Despite this can you guess from the timing which tcp packets are carrying which signals from the client to the server.

Number the signals in the diagram for Step 2 and then submit a screenshot of the filtered packet trace with each packet labelled with the signal number.

Step 4 – Staying with the same packet trace let's examine what happens after the signaling is complete.

The video between the clients is being carried over RTP. So now filter for udp.port == xxx, where xxx is the port number being used by one of the clients – you will need to figure this out from the trace. If you don't see RTP or RTCP but only UDP, you will need to follow the instructions here to see to see RTP/RTCP https://www.youtube.com/watch?v=z88F_-vIVs0

From the trace answer the following questions:

- What is the video bandwidth for one video stream? It is OK if you do a rough calculation. Note that the size of each RTP packet is given and wireshark gives you a timestamp.
- How often is RTCP being sent?
- Is the rule of RTCP taking less than 5% of the bandwidth being followed? Try to do a rough calculation to confirm this.

Step 5 – Let's now see the interaction between the clients and the STUN Server. For this you will need to re-run the server and clients. Collect a Wireshark packet trace but this time on the WiFi interface (or wired interface incase you are using a wired connection to the Internet). Let the videos run for a couple of minutes and then stop it. Examine the packet trace and filter it for "stun" protocol. Describe the interactions you see and explain what is going on in relation to what we discussed about ICE in class.

Part 2 – Using a cloud-based system

You may have observed that the system in Part 1 will only work in loop-back mode. This is obviously not very useful. In this part of the project you will deploy the system in the Heroku cloud platform. In this deployment the signaling server will run in the cloud. You will then be able to run the clients anywhere, including your phone!

Getting Ready:

In addition to what you used for Part 1 you will need the following:

1- A GitHub account is needed and the Git shell needs to be installed:
https://git-scm.com/downloads
The steps in this document use Git through the terminal instead of the GUI programs so it is best to install the command-line version.

2- A free account on Heroku. "Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to **deploy, manage, and scale modern apps**." Information about the Heroku free tier is here https://devcenter.heroku.com/articles/free-dyno-hours. The free tier of Heroku gives 550 "dyno hours" for free each month. One dyno hour is essentially running one instance of a project for one hour. Since we only need a single instance for this project, this will be more than enough. If the deployed Heroku web app is not used for 30 minutes, then it will go to sleep, and stop consuming the free hours.

Required Accounts and Software. A Heroku account can be made here: https://signup.heroku.com/ and the Heroku software can be downloaded here: https://devcenter.heroku.com/articles/heroku-cli#install-with-an-installer

It is best to use the same email that was used for the GitHub account.

Setting up Project on Heroku

After the Git and Heroku accounts are ready and the software is installed, then the Video-Chat project can be set up on Heroku using the following steps.

1. Some changes need to be made to the Video-Chat project to make it compatible with Heroku. First, open the file Video-Chat/index.js. Modify line 7 to read:

```
let server = app.listen(process.env.PORT || 4000, function () {
```

This is necessary as Heroku will assign a random port for the NodeJS to listen on which "overrides" the port 4000 originally used here.

2. Second, open the file Video-Chat/public/chat.js. Modify line 1 to read:

   ```
   let socket = io.connect("https://" + window.location.hostname);
   ```

   This will ensure that the client makes the WebSocket connection to the correct URL. The Heroku URL is random, so this makes it easier to deploy the project.

3. Use the command line (or Git Terminal program, if on Windows), navigate to the Video-Chat directory

4. In the command line, type `git init`

5. Create a new file called "Procfile" (with no file extension) in the Video-Chat directory. The contents of this file should be exactly:

   ```
   web: node index.js
   ```

6. Run `git add *`, this will ensure that Git does not miss any files.

7. Run `git commit -m "initial"`, which readies the files for upload to Heroku. (note: you may find you will need to run the step below first)

8. Set up Git to use your account email and password

   ```
   git config user.name githubusername
   git config user.email github@email
   ```

9. Run `heroku  create`. It may be necessary to do this from the Command Prompt in Windows instead of Git Terminal. This will print the URL which the video call can be accessed at in later parts.

10. Run `git push heroku master`
    - Note, that it may be necessary to replace "master" with "main" if this command does not work.

    After running this command, there will be some output describing the progress of uploading to the Heroku repository. The git push command will also do most of the set up for the Heroku application.

11. Run `heroku ps:scale web=1.` This ensures only one instance of the NodeJS project is running.

12. Run `heroku logs --tail`. If everything is running correctly, it should show something like the following at the output:

```
2022-03-09T23:34:59.531844+00:00 heroku[web.1]: State changed from crashed to starting

2022-03-09T23:35:01.401383+00:00 heroku[web.1]: Starting process with command `node index.js`
2022-03-09T23:35:02.721795+00:00 app[web.1]: Server is running

2022-03-09T23:35:03.278426+00:00 heroku[web.1]: State changed from starting to up
```

13. Run `heroku open`. This will open the webpage in the default browser. It is best if the webpage is opened in Google Chrome. Note the URL that shows up in your browse. This will be the URL you would use for running the conferencing app from anywhere. Report back this URL in your submission.

    a. Enter a room name (can be anything) and click Join. The camera footage should show up if it works.

    b. Open the Heroku webpage in another tab and enter the same room name and join. If it is working correctly, then two copies of the camera should show up on both tabs. One copy may be smaller than the other to start with.

    c. Close the client tabs and try running the clients again but this time from different systems (two laptops or one laptop and one phone). Always using the URL that showed up after step 13. above.

Tip: You can find this URL by logging into your heroku account and looking at the project that is running Node.js

IMPORTANT: Note that the only deliverable for Part 2 is the Herokuapp link. Please keep the project running in Heroku as we may check that your link is working by trying to connect to it.