How to Guide: Live Streaming Video to NodeJS on Google Cloud

Team Members: Andy Cheng, Denzel Nasol, Harry Nguyen, Mathew Wong

Introduction

In order for us to host our node server and send streams that were created with our beaglebone green and a camera, we chose to stream our webpage to a google cloud platform while the machine was running. We can always check our camera using our phones, laptops or tablets similar to any security camera application that you may have seen before. We chose Google Cloud Platform so we can always have the webpage running, even if the beaglebone or camera is not turned on. Once it's turned on, it will automatically update the webpage with a real time video of what's occurring on the stream.

Hardware Required

- Beaglebone Green
- Logitech C920 HD Pro

Note that we will be going over the steps on how to set up live streaming from the beaglebone to Google Cloud. However, we will also be going over the steps to achieve this locally so if you are not planning to host your server on GCP just skip over the section on GCP.

Part 1: Setting up camera

The camera used for this project is the Logitech C920 Pro, however, previous teams have used other cameras such as the Logitech C270.

Steps to setup the camera:

1. Plug the camera into the USB port of the BeagleBone.
2. Check that the camera is successfully plugged in with the following command:
    a. `(bbg) $ lsub`
    b. You should be able to see the name of your device on-screen.
3. Install the following packages on your BeagleBone:
    a. `(bbg) $ sudo apt-get install libopencv-dev`
        i. Used to run Derek Molloy's capture.c program
    b. `(bbg) $ sudo apt-get install ffmpeg`
        i. Used to retrieve stream data from the capture program and send it to the Node server.

4. Download a copy of Derek Molloy's capture.c file from https://github.com/derekmolloy/boneCV
5. Replace lines 493-498 in the capture.c file with the following code (These next few steps are referenced from the following guide - https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/ensc351/links/files/2017-student-howtos/CapturingAndStreamingWebcamVideoOnBBG.pdf):

```
if (force_format) {
        if (force_format==2){
                fmt.fmt.pix.width       = 1280;
                fmt.fmt.pix.height      = 720;
                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
                fmt.fmt.pix.field       = V4L2_FIELD_NONE;
        }
```

This sets the data stream format as MJPEG, which is what our camera is configured to record at. This application has our camera stream at 720p but choose a stream resolution that best suits your project.

6. Through the BeagleBone, compile the capture.c file with the following command
    a. `(bbg) $ gcc capture.c -lv412 -o capture`
    b. This executable will allow you to turn on and begin streaming data from the webcam.
7. To use this executable, you can use ffmpeg to help process the video and send the data directly to your Node server with the following command:
    a. `(bbg) $ ./capture -F -o -c0|ffmpeg -vcodec mjpeg -i pipe:0 -f mjpeg udp://<ADDRESSHERE>:<PORT NUMBER HERE>`
    b. This command uses the capture executable along with ffmpeg to send stream data to the specified IP address and port number your Node server is running at.
8. Keep in mind while developing your application that not only can you run this command through your terminal, but it can also be run from inside your C code to activate your camera when needed.

Part 2: NodeJs server

Please have a look at this guide: Streaming Webcam Video to a Browser for more details on setting up the NodeJs server. In addition, we will also show you how you can create a recording of the live stream. We will be using the same library JSMpeg as the guide for playing/receiving the video.
We will be using socket.io instead of websockets. We will also be using express for the server framework.

Server-side NodeJs

You can choose if you want to use the code exactly or only take parts of the code (if you only take parts of the code it is not guaranteed to work). Create a folder called recordings which we will use to store the recordings. We will be creating a module called udp_server.js that manages the video streaming. In udp_server.js add the following

udp_server.js:

```
var dgram = require('dgram');
var fs = require('fs');
var SocketIOServer = require('socket.io').Server;
var io;
var udpServer;

var fileStream;

const STREAM_PORT = 8080;
const STREAM_IP_ADDRESS = 'localhost';

exports.listen = function(server) {
    io = new SocketIOServer(server);

    io.sockets.on('connection', (socket) => {
        console.log('A client has connected');

        // Handle disconnection of clients
        socket.on('disconnect', () => {
            console.log('Client disconnected');
        });
    });

    // Initialize socket to listen for the webcam streaming data
    udpServer = dgram.createSocket('udp4');
    udpServer.bind(STREAM_PORT);

    udpServer.on('message', (msg) => {
        if (!fileStream) {
            fileStream =
fs.createWriteStream(`recordings/new-recording.mp4`);
        } else {
            fileStream.write(msg);
```

```
        }
        io.emit('stream', msg);
    });

    udpServer.on('close', () => {
        if (fileStream) {
            fileStream.end();
        }
    });

    udpServer.on('error', (err) => {
        if (fileStream) {
            fileStream.end();
        }
        console.log(`server error: ${err.stack}`);
    });
}
```

Note that we are using port 8080 for streaming otherwise set the STREAM_PORT = <your-streaming-port>. Finally add the following code to index.js:

index.js

```
var fs   = require('fs');
var path = require('path');
var express = require('express');

var sioserver = require('./udp_server');

const WEBSOCKET_PORT = 8088;
const WEBSOCKET_IP_ADDRESS = 'localhost';

const app = express();

app.use(express.json());
app.use(express.urlencoded({extended:false}));
app.use(express.static('public'));

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, '/public/index.html'));
});

app.get('/recordings', function (req, res) {
```

```
    const dirPath = './recordings';

    fs.readdir(dirPath, function (err, files) {
        if (err) {
            console.log(err);
            return res.status(500).send('Error reading directory');
        }
        res.json(files);
    });
});

const server = app.listen(WEBSOCKET_PORT, () => {
    console.log(`server listening on port ${WEBSOCKET_PORT} with address`,
server.address());
})

sioserver.listen(server);
```

Please see the guide on how to set up the client side.

Part 3: Hosting your NodeJs server on GCP

This section assumes that you already have a GCP account with billing enabled. This costs around $25/month depending on your VM specs.

Setup and Configure Linux VM

1. Login to your google cloud account and click on ‘Console’

2. Click on Compute Engine > VM Instances.

3. Click on CREATE INSTANCE at the very top of the page.

4. Here is our suggested VM configuration. Note that this exact configuration should be around $25.46/month, but this may change based on Google’s pricing. Feel free to lower the specs if you want to save some credits but we suggest using the default configuration due to performance reasons.
    a. Machine Configuration
        i. Series: E2
        ii. Machine-type: e2-medium (2vCPU, 4GB memory)

5. Under the Boot Disk section choose Ubuntu 18.04 LTS with Standard persistent disk and give it a size of 10GB which should be enough for storing live recordings on the serve

6. Under the Firewall section check off Allow HTTP Traffic

7. Click Create

8. Go back to your VM Instances dashboard and click on 'Set up Firewall Rules'

9. Click on CREATE FIREWALL RULE
    a. Direction: Ingress
    b. Action on match: Allow
    c. Targets: All instances in the network
    d. Check off UDP under the protocols and ports section
    e. Leave everything else as the default
10. Now go back to your VM Instances dashboard and connect to your VM through the browser's SSH shell.

11. Install and setup nodejs, npm, nginx and pm2. Here are some tutorials:
    a. https://linuxize.com/post/how-to-install-node-js-on-ubuntu-20-04/
    b. https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/
    c. Installing pm2: sudo apt install pm2
        i. Pm2 was used to run our server constantly as long as the vm instance was still running, you could choose to start your server with node server.js instead, but we used pm2 as once you close your instance terminal, the server won't be active anymore.

12. Copy the nginx config file to the nginx configuration path. Default should be /etc/nginx/sites-available

13. Next you would type in sudo nano default and inside the default file, you can add the following code for nginx
    a. For server_name, this will be your external ip address so it will change based on your gcp. The ports indicate what port you are reverse proxying, and in this case, the reverse proxy is combining both the /client and /server together.

default.conf

```
upstream node_server {
      server 127.0.0.1:8088;
}
server {
      listen 8080 udp;
```

```
        proxy_pass node_server;
}
server {
        listen 80;
        server_name <your-ip-address>;
        location /client {
                proxy_pass http://localhost:8088/;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }
}
```

14. Once that's added to nginx, you must restart nginx with the following command “sudo systemctl restart nginx”
    a. If systemctl is not found, you may have to install it. To install systemctl you can use the following commands
        i. `sudo apt-get update`
        ii. `sudo apt-get install systemctl`
15. Copy the nodejs scripts from the nodejs section in the guide to your VM. Remember to do a npm install.
16. On another tab navigate to http://<external-ip-address>. The external ip address of your VM should be displayed under External IP in the VM Instances dashboard
17. (Optional) If you want you can also set up a domain and certificate authority so your webpage can be accessible through DNS instead of the IP address. This will not be covered in the guide but here are some tutorials to get you on the right track:
    a. https://cloud.google.com/dns/docs/tutorials/create-domain-tutorial
    b. https://cloud.google.com/certificate-authority-service/docs/creating-certificate-authorities
18. Start the camera on the beaglebone and ta-da, your beaglebone is live streaming to the Cloud!

Troubleshooting

Please see the other guides before moving on with this one. First you should make sure that you can stream the data to localhost before you move on to the GCP section.
If you changed any of the port numbers then make sure your server is listening on the correct port/ip address. Some library functions might be outdated by the time this guide is released. Check out the documentation:
Express: https://expressjs.com/
JSMpeg: https://github.com/phoboslab/jsmpeg
Socket.io: https://socket.io/
FFMPEG: https://ffmpeg.org/documentation.html
NGINX: https://docs.nginx.com/

References

Previous student guides
Capturing and Streaming Webcam Video with the BeagleBone Green
Streaming Webcam Video to a Browser

BoneCV by Derek Molloy for the camera
GitHub - derekmolloy/boneCV: Beaglebone Webcam and OpenCV Examples Repository