Lab 3: Client-Server & Pub-Sub Communication

NOTE: We will be installing essential software dependencies and packages so make sure you have a stable internet connection for your Jetson Nano before proceeding.

Objective: This lab will cover the setup and use of a communication interface between jetson nano devices and web servers.

Preface:
Client-server communication enables communication between the Jetson Nano and other devices or services, allowing the board to access and share information with them. Some uses for this structure include:

- remote control
- data sharing
- scalability
- inference tasks
- distributing workloads

Client-server communication enables building sophisticated applications that leverage the board's computing power and connectivity. Some of the concepts of this lab identify methods of communication between multiple edge devices.

In pub-sub communication for edge devices, publishers send messages without knowing the subscribers, and subscribers receive messages based on their interests/subscriptions, allowing for scalable and flexible data exchange in real-time scenarios. Some uses for this structure include:

- Real-time Updates
- Event-Driven systems
- Distributing workloads
- Data Aggregation

Materials Required:
- Jetson Nano Development Kit (basic setup complete)
- IMX 219 or USB Webcam
- ethernet cable/WIFI Module
- keyboard and mouse
- display with cables / secondary computer

Assignment Submission Instructions

You will need to turn in the following for full credit (10 points) on today's lab:

- screenshot of functional virtual server (1 pts)
- Part II: Pub Sub deliverables (4 pts)
- screenshot of webrtc stream (1 pts)
- all questions answered (3 pts)
- all the items above turned in on github/canvas, github preferred (1 pts)

Part I - Basic Web Server

Step 1: install relevant packages

- open terminal using sidebar or search
- update current package list

```
sudo apt update
```

- run the following commands:

```
sudo apt install nodejs
sudo apt install npm
```

Step 2: download the .zip file from drive link

- google drive link: https://drive.google.com/drive/folders/1GWN_onD-k3_4IfsEJPZt8kJHiP5x6rpC?usp=sharing
- unzip the js file
- place them in an easy to access file location

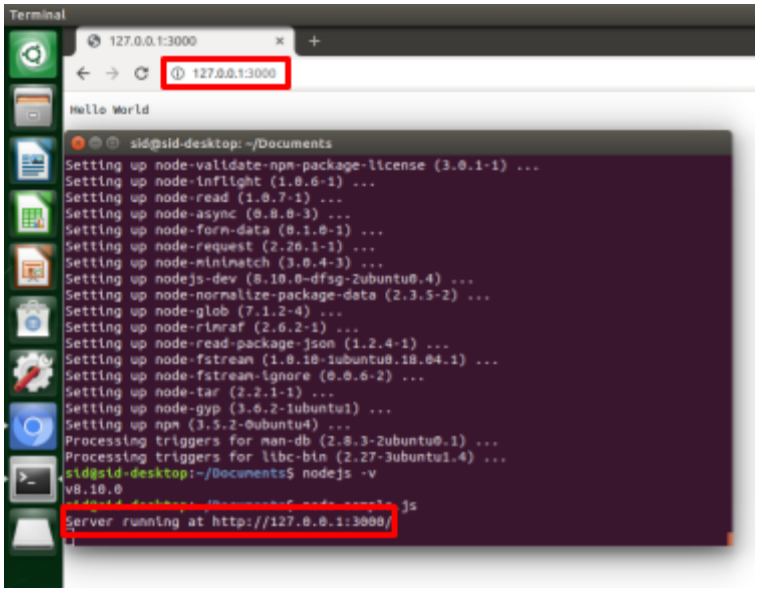
Step 3: run javascript code

- cd into the relevant directory
- use the following to compile and run the code

```
node sample.js
```

Step 4: go to virtual server

- grab the url output in console
- copy paste in any available browser



Based on your knowledge of the web server, answer the following:

Q2.1: Is the web server considered local or remote? What is the difference between each?

Q2.2: Why were we provided with a number prefix (127.0.0.1) in our URL?

Part II - Pub Sub

Setup:
Open Terminal >

```
sudo apt-get install python-dev python3.7-dev
python3.7 -m pip install --upgrade pip setuptools wheel
python3.7 -m pip install pyzmq
```

Read the usage of pyzmq
https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/basics.html
https://pyzmq.readthedocs.io/en/stable/api/zmq.html

Publish
Subscribe

Replace the ip address in subscribe code with your nano wlan/eth0 ip address, using 'ifconfig' command to get it.

Problem Statement:
Create a simple, real-time message board application that utilizes the Publisher-Subscriber (Pub-Sub) messaging pattern. This application will facilitate asynchronous communication among users subscribed to various topics.
Library: ZeroMQ

Requirements:

1. Create three predefined topics, for e.g. #general, #cs131, and #fun
    a. Users can publish messages to any of these topics

2. Publisher
    a. Write a publisher script and **run it on your Jetson device**
    b. Publishers must be able to select a topic and post messages to it.
    c. The message, once published, should be broadcasted to all subscribers of that topic.

3. Subscriber
    a. Write a subscriber script and test it on your device
    b. Subscriber must be able to subscribe to one or more predefined topics.
    c. Subscribers should receive real-time updates of any new messages posted to the topics they are subscribed to.
    d. **Testing on Your Device:**

        i. Test the subscriber script on your local device.
        ii. Run the publisher script simultaneously to ensure that messages on subscribed topics are correctly received in real-time.
    e. **Cross-Team Testing:**
        i. Share the subscriber script with a neighboring team.
        ii. They should be able to run your subscriber script, subscribe to any of your topics, and receive messages from your publisher script in real-time.
        iii. This step is crucial for verifying network communication and the proper functioning of the Pub-Sub mechanism across different devices.

4. User Interface
    a. Implement a simple command-line interface (CLI) for both publishers and subscribers.
    b. For publishers, prompt for topic selection and message input.
    c. For subscribers, prompt for topic subscription choices and display incoming messages.

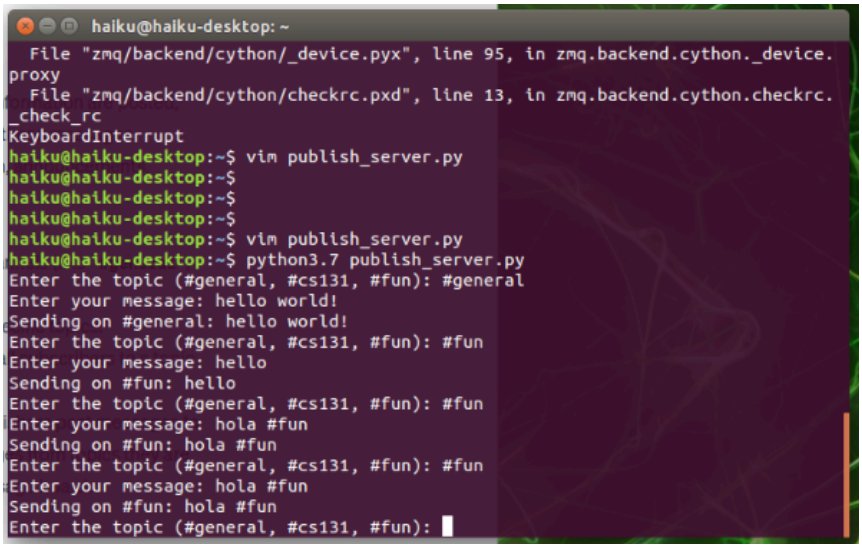
5. Error handling
    a. Include basic error handling, such as invalid topic selection.
    b. Validate user inputs where applicable.

Deliverables:

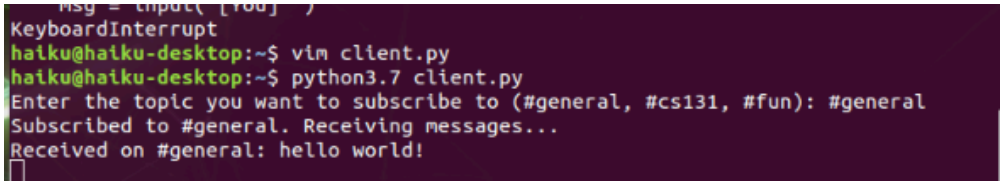
- Publisher and Subscriber code files
- Readme.txt file with all the commands used to run your code
- Screenshot of publisher while posting the messages from your Jetson
- Screenshot of the subscriber receiving the message from your neighbors Jetson device

Your application could look like this

Publisher

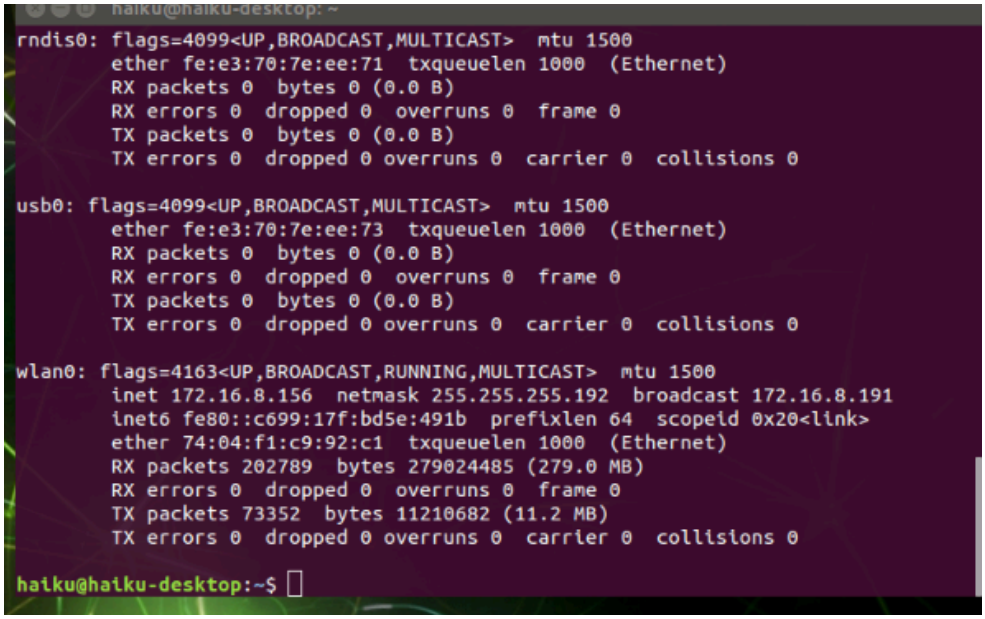


Subscriber



NOTE:
For other Jetson devices to connect and receive messages, ensure they are on the same network as your device. Find your Jetson's IP address using ifconfig, wlan is for Wi-Fi) or eth is for Ethernet, and look for the 'inet addr'. In the subscriber script on other devices, replace localhost with your Jetson's IP address in the socket.connect line.

```
ifconfig
```



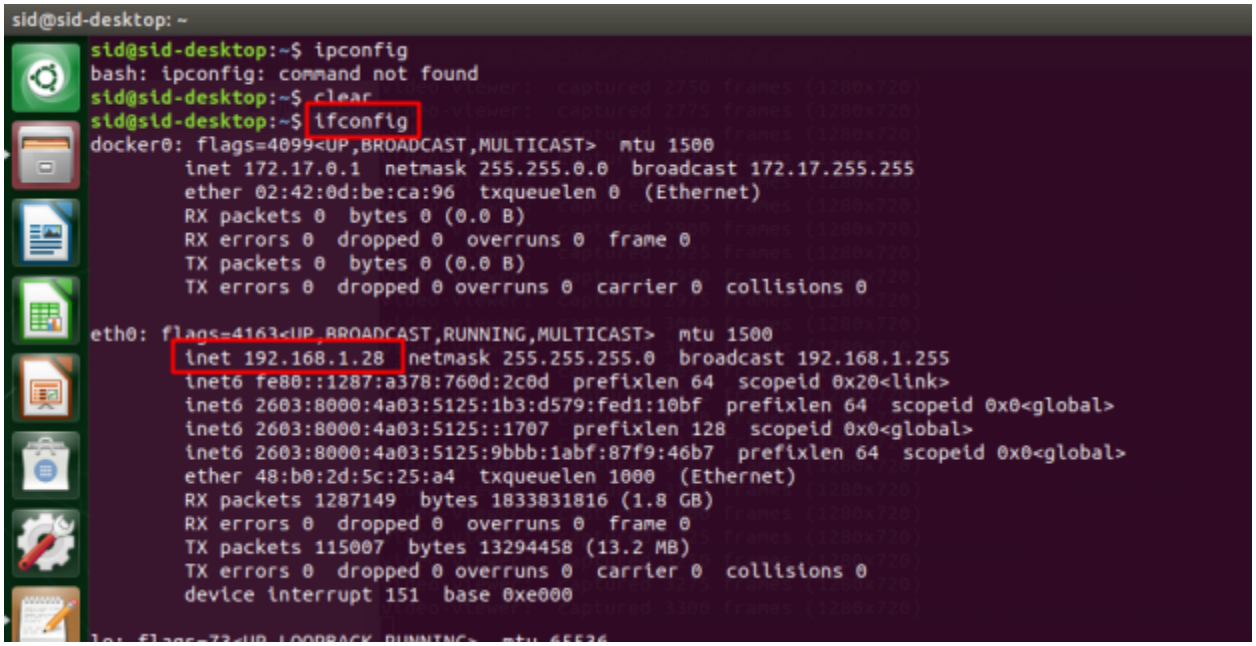
Part III - WebRTC Server

Step 1: identify your nano IP address

- connect your nano to an ethernet cable or WIFI module
- launch the terminal
- run the command:

```
ifconfig
```

- If using Ethernet - under the eth0 port, identify your IP address
- If using WIFI module - under the wlan port, identify your IP address



Step 2: enable https/ssl

- go to the *jetson-inference* container
- run the docker container using

```
docker/run.sh
```

- go into the following directory: ./jetson-inference/data
- generate a self-signed SSL certificate and key using the following commands:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256
-days 365 -nodes -subj '/CN=localhost'

#set the $SSL_KEY and $SSL_CERT environment variables in docker
export SSL_KEY=/jetson-inference/data/key.pem
export SSL_CERT=/jetson-inference/data/cert.pem
```

Step 3: sending webrtc streams

- make sure you are running docker/run.sh
- run the following command (must be in root):

```
# stream V4L2 camera to browsers via WebRTC
# follow webrtc://<INTERFACE>:<PORT>/<STREAM-NAME>
# Replace /dev/video0 with csi://0 for IMX 219 camera
detectnet /dev/video0 webrtc://@:8554/output
```

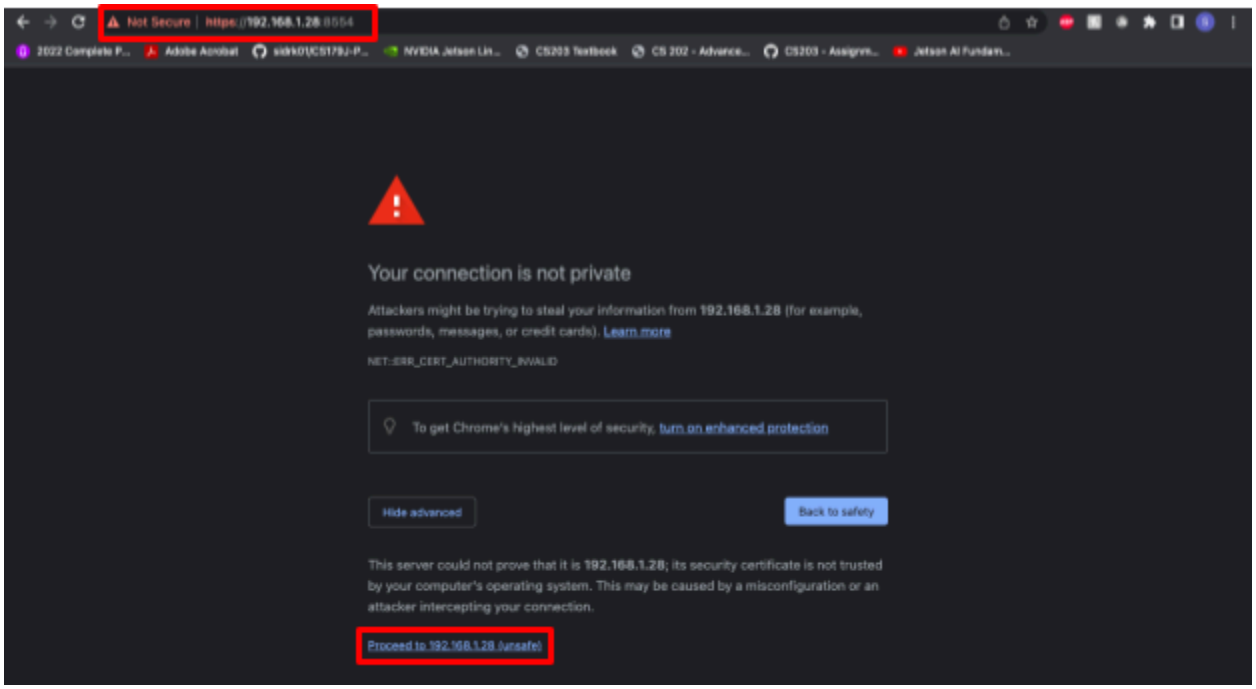
Step 4: view your webrtc stream

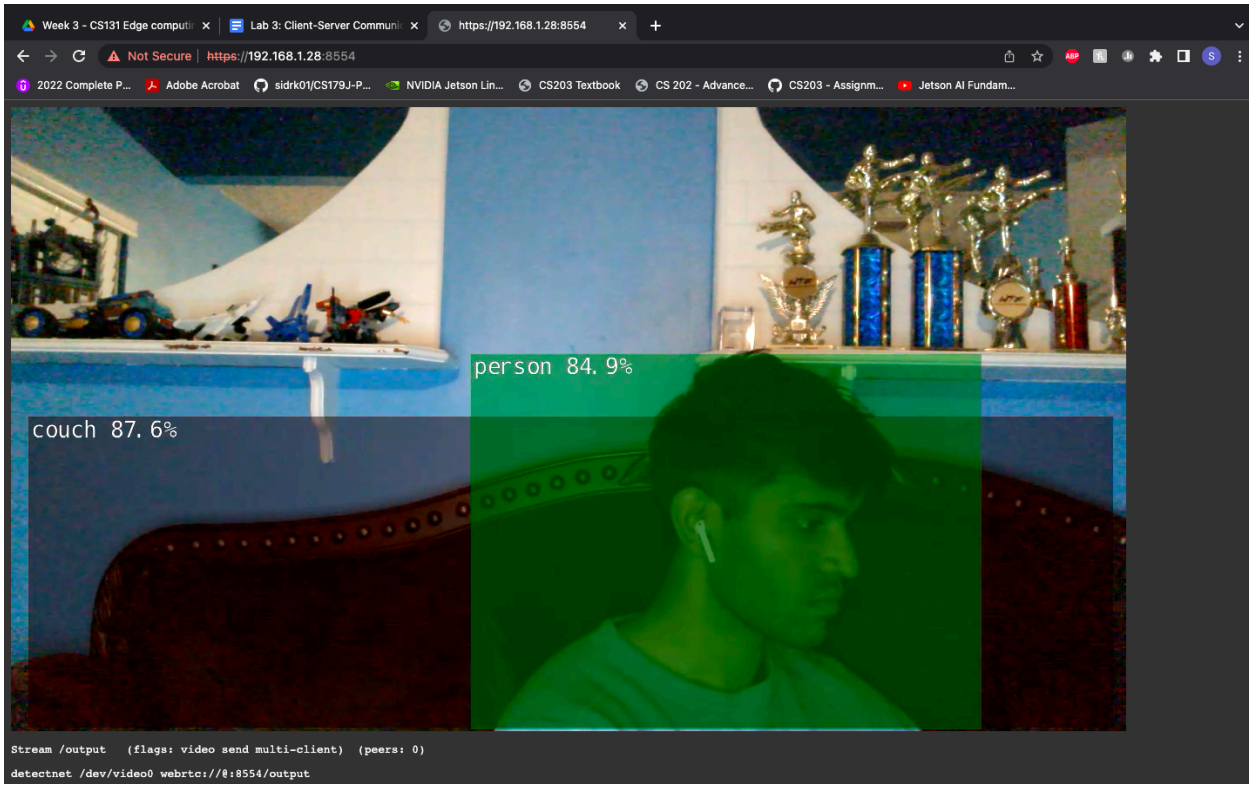
NOTE: the interface should be viewable to all devices connected to your network

- start a chrome browser on a secondary computer
- navigate to the following URL:

```
https://<JETSON-IP>:8554
```

- click on *proceed to URL* when prompted

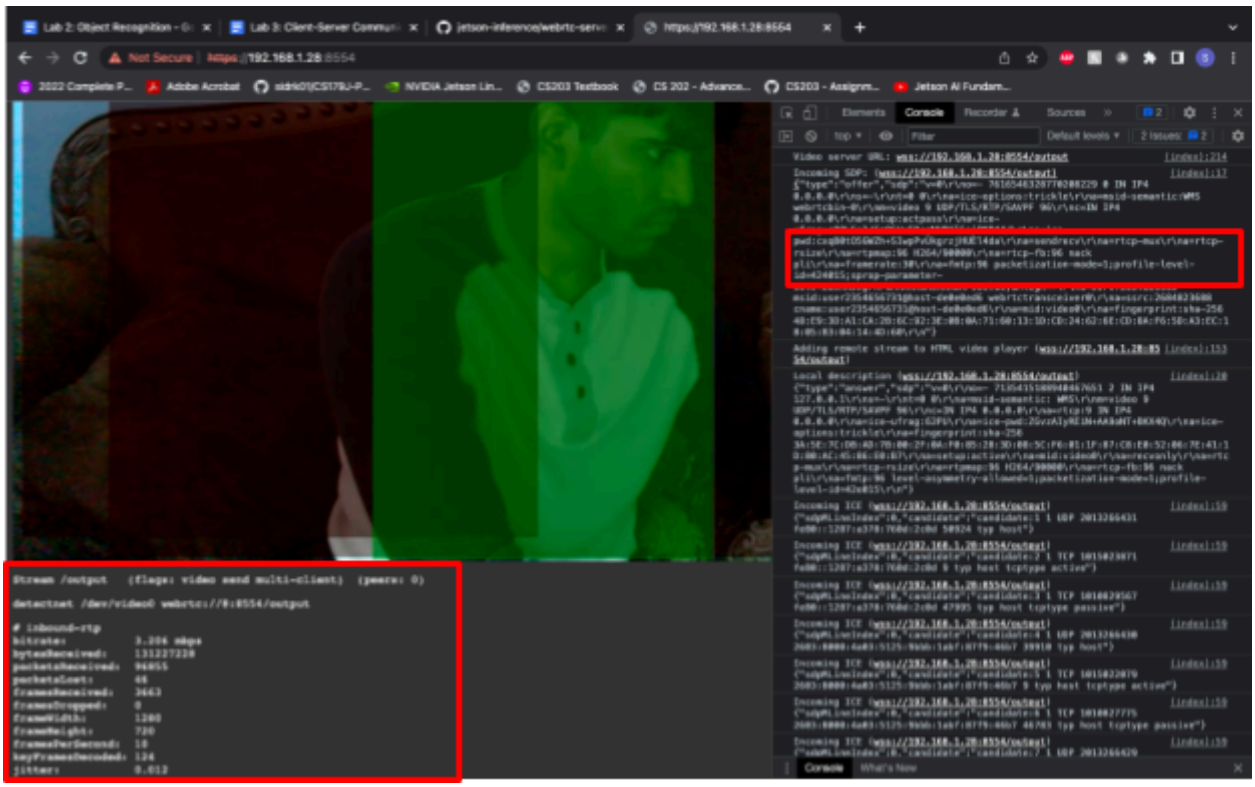




DetectNet sent to chrome browser through webRTC for playback

Step 5: connection statistics and console debug

- scroll to the bottom of the page
- right click on the page and select inspect
- navigate to the console tab



Use the information provided by the statistics to answer the following questions:

Q3.1: What does bitrate represent in terms of video-streaming? How is it relevant here?

Q3.2: How many packets were received and dropped? What does this statistic represent?

Q3.3: Were you able to receive a stable framerate of 30 fps? Describe any possible latency that could affect this parameter.

Resources

- NodeJS Tutorial
- WebRTC Server Repository
- Jetson-Inference Repository
- Lab 2: Object Recognition