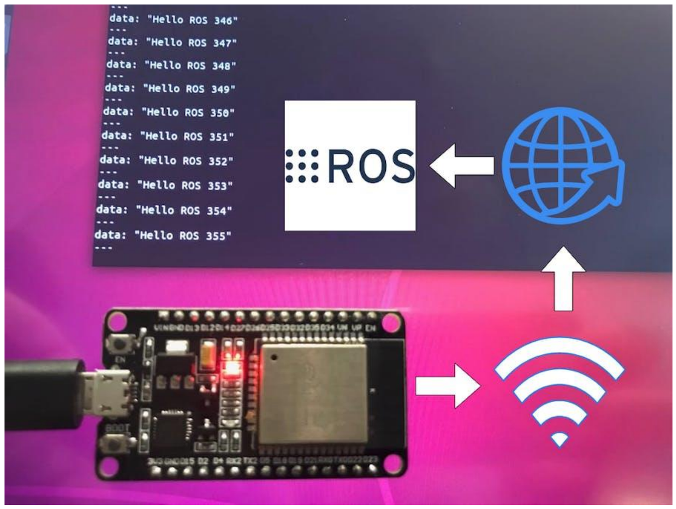
# ESP32 interface and connection

## 1. Our Goal?




In this project, we want to connect ESP32 module to our Jetson TX2 kit. The ESP32 is connected to a calling button. As a result, ESP32 sends a signal whenever the user pushes the button and on receiving this signal, the robot should start moving to a prespecified position.

With such settings, a wired serial connection is off the table, so how does the communication happen?

One of the available properties in the esp32 is its ability to connect to VPN client networks (e.g.: through home routers). Such feature enables us to connect between esp32 and any other device connected to the same router (other microcontrollers, laptops, phones, ...) and so, we can connect it to ROS/ROS2.

## 2. Methods to connect esp32 to ROS vs ROS2

In our project, we mainly work with ROS2. However, we can use ROS methods to send data to our device, and then bridge these data to be used in our ROS2 project.

### a. ROS using Husarnet

One way to connect ESP32 to ROS is using Husarnet client. Husarnet VPN Client can run not only on servers, laptops and mobile phones, but also on microcontrollers with very limited computing power and memory.

*In this part, I will show you how to use rosserial on ESP32 to connect it with the ROS powered system over the internet.*

- **Install Husarnet**

```
curl https://install.husarnet.com/install.sh | sudo bash
```

- **Setting up the environment**

Add these lines to .bashrc (or .zshrc if you use zsh) of the user who will use ROS:

```
source /opt/ros/noetic/setup.bash

export ROS_IPV6=on
export ROS_MASTER_URI=http://master:11311
```

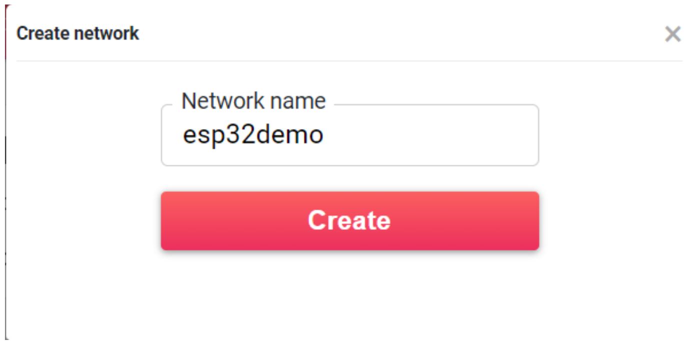
Sourcing the `/opt/ros/noetic/setup.bash` enables all ROS tools.

`ROS_IPV6` makes ROS enable IPv6 mode - Husarnet is a IPv6 network.
Setting `ROS_MASTER_URI` to `http://master:11311` ensures ROS will always connect to host called `master` - which extact machine it is depends on the setting on the Husarnet Dashboard.

You can also set `ROS_MASTER_URI` to other hostname - just be aware that Husarnet Dashboard ROS integration might not work as intended.

- **adding linux to Husarnet**

create a network in https://app.husarnet.com




get the join code from the add element button

- join to husarnet from linux machine now

```
husarnet join fc94:b01d:1803:8dd8:b293:5c7d:7639:932a/xxxxxxxxxxxxxxxxxxxxxx mydevice
```

for Arduino IDE setup, follow instructions at https://www.hackster.io/khasreto/run-rosserial-over-the-internet-with-esp32-0615f5 from Arduino IDE section

In "Customize Code" section, please note the following:

hostNameESP: is the name you want your esp32 to be called by in Husarnet

hostNameComputer: your computer name in husarent.

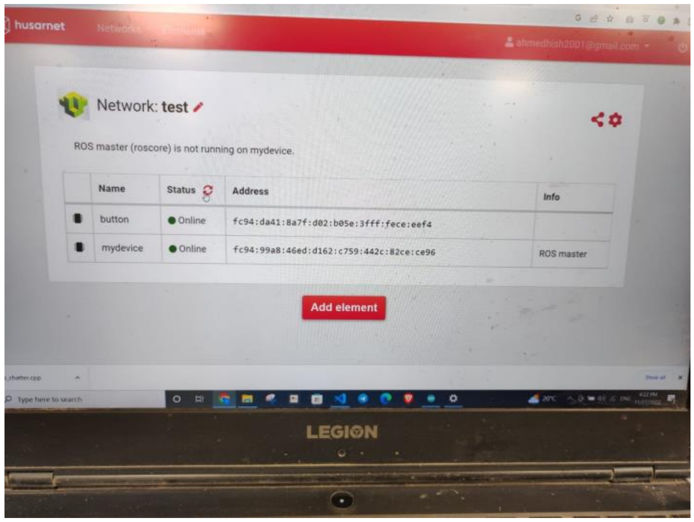
husarnetJoinCode: the code you got from https://app.husarnet.com

NUM_NETWORKS: set it to 1 if you are connected to only 1 router

ssidTab: the wifi name

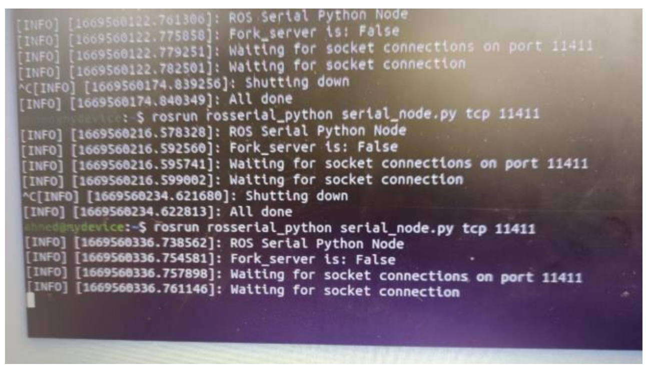
passwordTab: wifi password

## Results




After following the previous steps, you should see 2 devices connected to your VPN client. Which means they now can communicate with each other.

### Errors




If you have followed the instructions line-by-line, you should have encountered an error in “Set Up Rosserial” section. That’s not your fault, it’s because this special version of rosserial uses an outdated Boost. Such error can be fixed by removing your current boost version and install an older one, but I don’t recommend that you do that as many ROS and ROS2 packages depend on Boost. So such change will cause corruption to other packages.

We can still use Husarnet client to send data by following instructions in https://adkarigar004.medium.com/husarnet-and-esp32-in-action-5273f0a24785 , however, this method has proved to be inefficient with ROS.

## b. ROS2 with micro-ros


micro-ROS targets mid-range and medium-to-high performance 32-bits microcontrollers families. Up to now, the boards officially supported by the project were solely based on the STM32 series from ST, MCUs featuring ARM Cortex-M processors. On the other hand, the ESP32 is an ultra-low power consumption dual-core system with two Xtensa LX6 CPUs, exposing a large collection of peripherals.

### Step1. Micro-ROS Installation

```
# Source the ROS 2 installation
source /opt/ros/$ROS_DISTRO/setup.bash
# Create a workspace and download the micro-ROS tools
mkdir microros_ws cd microros_ws git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git src/micro_ros_setup
# Update dependencies using rosdep
sudo apt update && rosdep update rosdep install --from-path src --ignore-src -y # Install pip sudo apt-get install python3-pip
# Build micro-ROS tools and source them colcon build source install/local_setup.bash
```

### Step 2. Creating a new firmware workspace for ESP32

```
ros2 run micro_ros_setup create_firmware_ws.sh freertos esp32
```

## Step 3. Configuring created firmware

```
ros2 run micro_ros_setup configure_firmware.sh [PROJECT NAME] -t udp -i [LOCAL MACHINE IP ADDRESS] -p 8888
```

In our first trial, we will try int32_publisher project. Make sure to write int32_publisher instead [PROJECT NAME]

To get the `LOCAL MACHINE IP ADDRESS` , use the following command

```
ifconfig
```

```
      ether 3c:7c:3f:19:41:a9  txqueuelen 1000  (Ethernet)
      RX packets 0  bytes 0 (0.0 B)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 0  bytes 0 (0.0 B)
      TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

o: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
      inet6 ::1  prefixlen 128  scopeid 0x10<host>
      loop  txqueuelen 1000  (Local Loopback)
      RX packets 2688  bytes 242390 (242.3 KB)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 2688  bytes 242390 (242.3 KB)
      TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 192.168.1.68  netmask 255.255.255.0  broadcast 192.168.1.255
      inet6 fe80::f2d2:d4c5:5fc2:2dbb  prefixlen 64  scopeid 0x20<link>
      inet6 2405:9800:bc30:de27:6b14:828f:aed9:e274  prefixlen 64  scopeid 0x0<global>
      inet6 2405:9800:bc30:de27:d159:56da:1463:9ab9  prefixlen 64  scopeid 0x0<global>
      ether 80:30:49:3e:8c:a9  txqueuelen 1000  (Ethernet)
      RX packets 221364  bytes 248233101 (248.2 MB)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 64697  bytes 11992336 (11.9 MB)
      TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The copy the highlighted part and paste it instead `[LOCAL MACHINE IP ADDRESS]`

## Then use the following command to set your wifi credentials for ESP32 to connect to your network

```
ros2 run micro_ros_setup build_firmware.sh menuconfig
```

## Step 4. Build firmware

```
ros2 run micro_ros_setup build_firmware.sh
```

Step 5. Flash firmware

Plug your ESP32 cable to the computer and run command

```
ros2 run micro_ros_setup flash_firmware.sh
```

## Step 6. Creating the micro-ROS agent

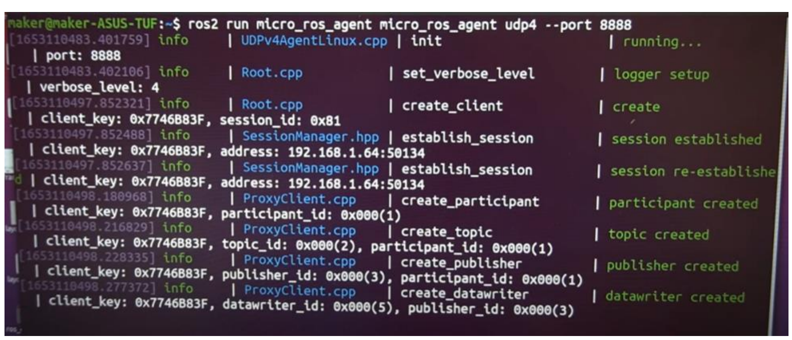
```
# Download micro-ROS-Agent packages
ros2 run micro_ros_setup create_agent_ws.sh
# Build step
ros2 run micro_ros_setup build_agent.sh source install/local_setup.bash
```

Step 7. Running the micro-ROS

```
# Run a micro-ROS agent
ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
```

If you have come this far, that means you now have a working connection between ros2 and esp32. You can check that by resetting esp32 (by bushing reset button)

You will find some new lines have been added to your current running terminal as follow:




And by running ros2 topic list

You will find a new topic created (make sure the esp32 is power)

By running ros2 topic echo [TOPIC_NAME], you should see some data sending from esp32 to ROS2

## *What’s next?*

Now that we saw data being transmitted from esp32 to ROS2 as follows:


We should ask ourselves: “what if I don’t want to send incrementing numbers, what if I want to send actual useful data?”

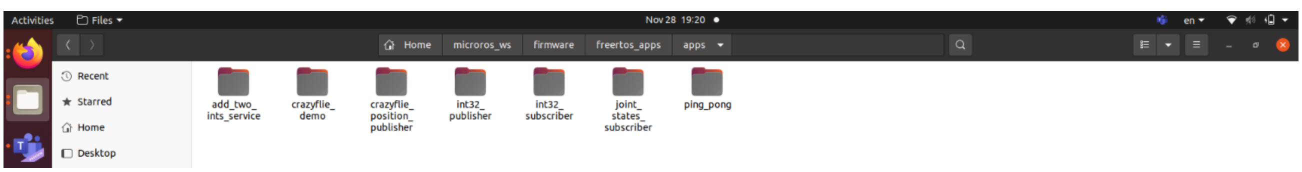
The answer to that question is simple, then we will have to create our own project (app).

To do so, micro-ros documentation provides tutorials to write your own project in

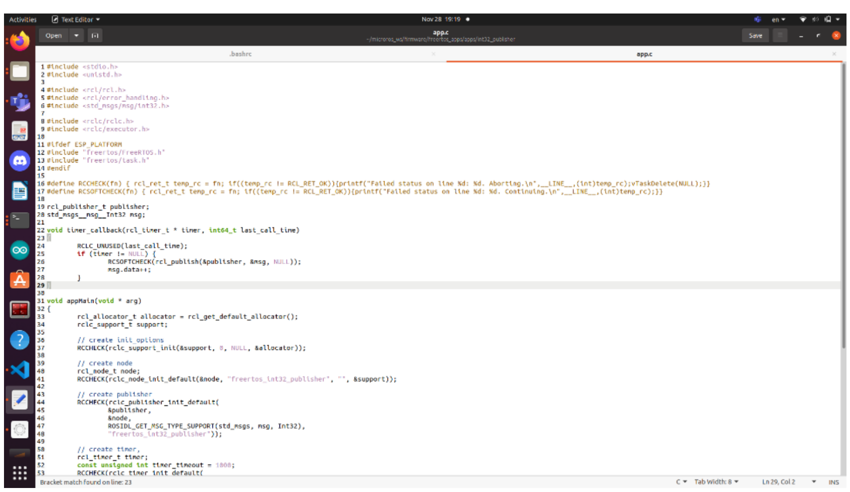
[https://micro.ros.org/docs/tutorials/core/overview/](https://micro.ros.org/docs/tutorials/core/overview/)

let’s take a closer look to the current available apps, for example: int32_publisher

first, we navigate to the apps folder, we will see the following



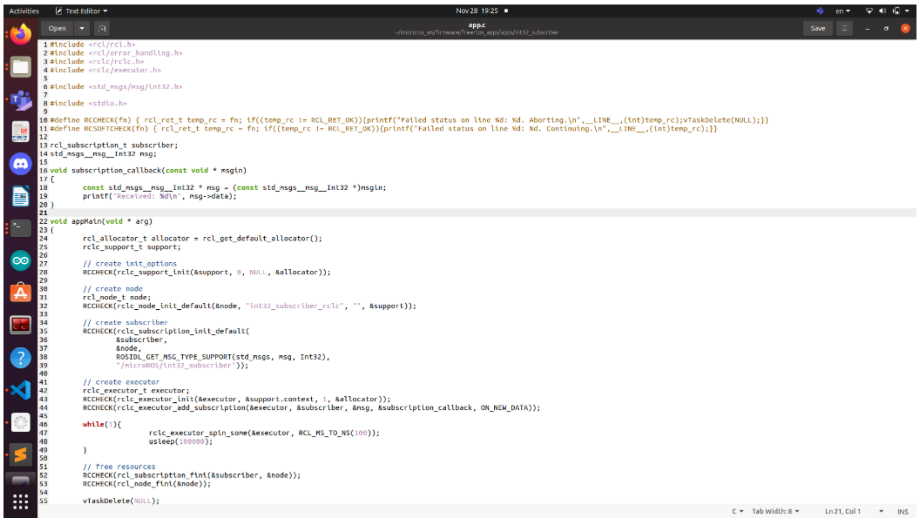

By taking a closer look at int32_publisher app, we find the following



We notice that at line 26, a message containing the required data is published ever specific period.

So for instance, if we want to publish (1) when the button is pressed, we need to add another if condition and rewrite msg.data to be msg.data=1

The same goes for the subscriber demo int32_subscriber



We can change the action taken when receiving data in line 19, for example

if(msg->data == 1)

Gpio_LED_ON();

Else

Gpio_LED_OFF();

These are basic examples for using publisher and subscriber using freeRTOS, we can create more complicated projects by following tutorials in micro-ros documentations.

### *Disadvantages*

As micro-ros is considered one of the most straightforward methods, it creates a challenge by being supported only on low-powered microcontroller and not supported on mobiles phones.

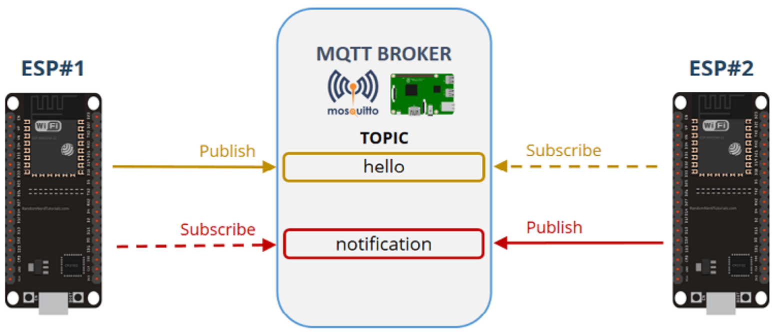
This will make HRI (Human-Robot interface) much more difficult. So, as effective as this method can be. We still need to look for another method.

## c. ROS2 with MQTT client (future)

The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers.

### *Overview*




### *Prerequisites*

- MicroPython firmware

To program the ESP32 and ESP8266 with MicroPython, we use uPyCraft IDE as a programming environment. Follow the next tutorials to install uPyCraft IDE and flash MicroPython firmware on your board:

Install uPyCraft IDE: Windows PC, MacOS X, or Linux Ubuntu

Flash/Upload MicroPython Firmware to ESP32 and ESP8266

- MQTT Broker

### *Resources*

https://randomnerdtutorials.com/micropython-mqtt-esp32-esp8266/

https://ashut0sh.me/posts/mqtt-implementation-in-python/

https://www.ibm.com/docs/en/ibm-mq/7.5?topic=clients-getting-started-mqtt-client-java-android

(last one is an example for HRI)