# P2P Communication between NodeMCUs

Group Members:
ABHAY GOYAL (202211034)
HARSH VARSHNEY (202211001)

# Motivation

The problem of communication exists -

- In mountains, hilly areas (within an area and to other areas),
- Between the outside and inside areas of tunnels, and
- Remote, rural and forest areas.

So, we got a motivation to develop a system to solve this problem and improve connectivity in such areas.

# State of the Art Technology

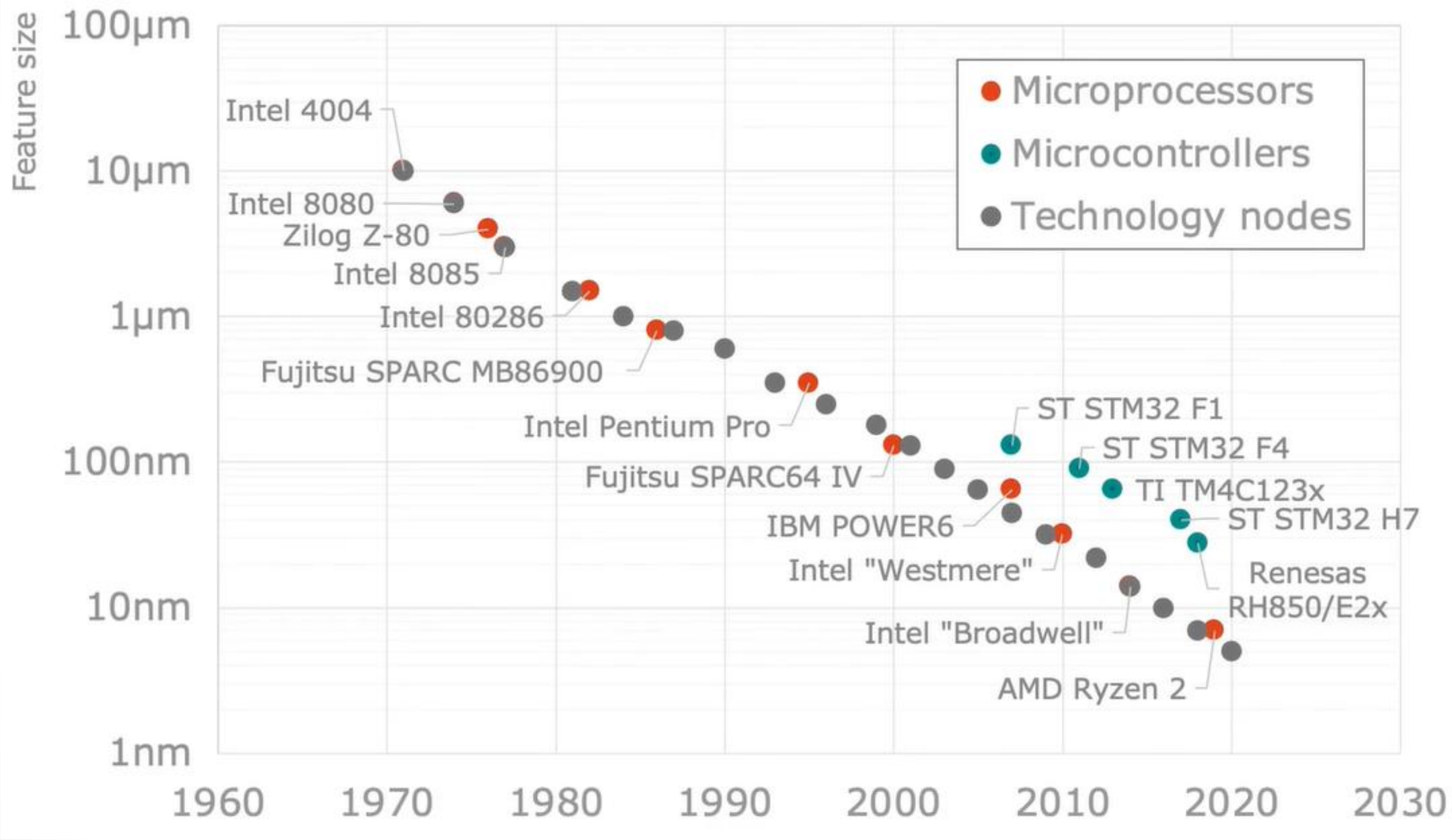


Image Source- https://bit.ly/3MhRN6W

# Project Objective

- 2-way P2P communication between 2 NodeMCUs.
- 2-way P2P communication between multiple NodeMCUs for increased connectivity and range.

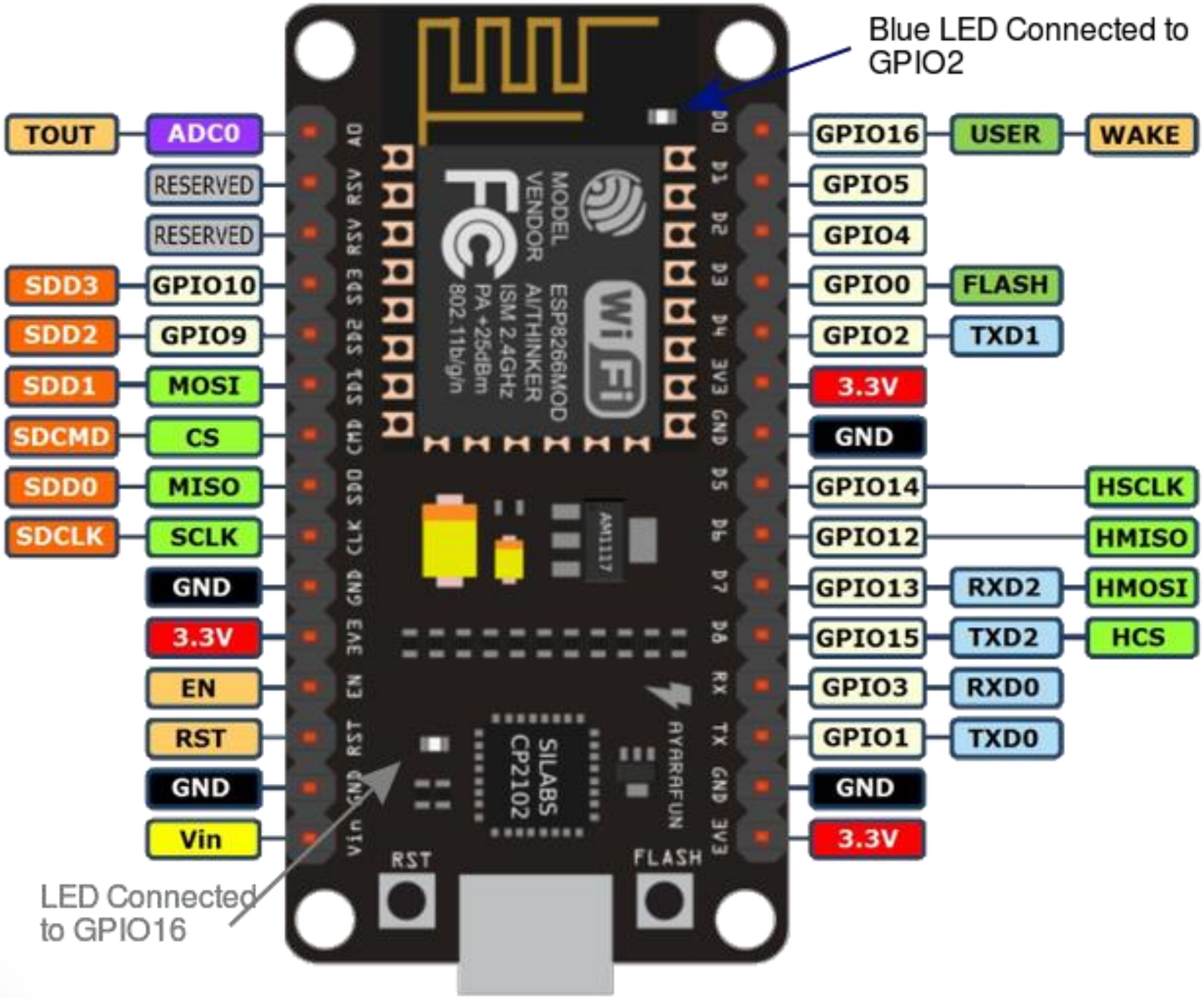
# Components Required

- Frontend (Hardware)
  - NodeMCU Modules
  - USB Cable
  - Connecting wires
  - Breadboard (in development phase)
  - Custom PCB (in implementation phase)
- Backend (Software)
  - Arduino IDE (For Arduino Programming)
  - Thonny IDE (For MicroPython Programming)

# NodeMCU

- An open-source firmware and development kit that helps us to prototype our IoT product within a few Lua script lines.
- The name "NodeMCU" combines “Node" (connecting point) and "MCU" (micro-controller unit).
- NodeMCU Development board features Wi-Fi capability, analog pins, digital pins, and serial communication protocols.
- It uses many open-source projects, such as lua-cjson, MicroPython, Arduino and SPIFFS.

# NodeMCU V3 Pin Diagram



Blue LED Connected to GPIO2

LED Connected to GPIO16

# One-way Communication Between Two NodeMCUs

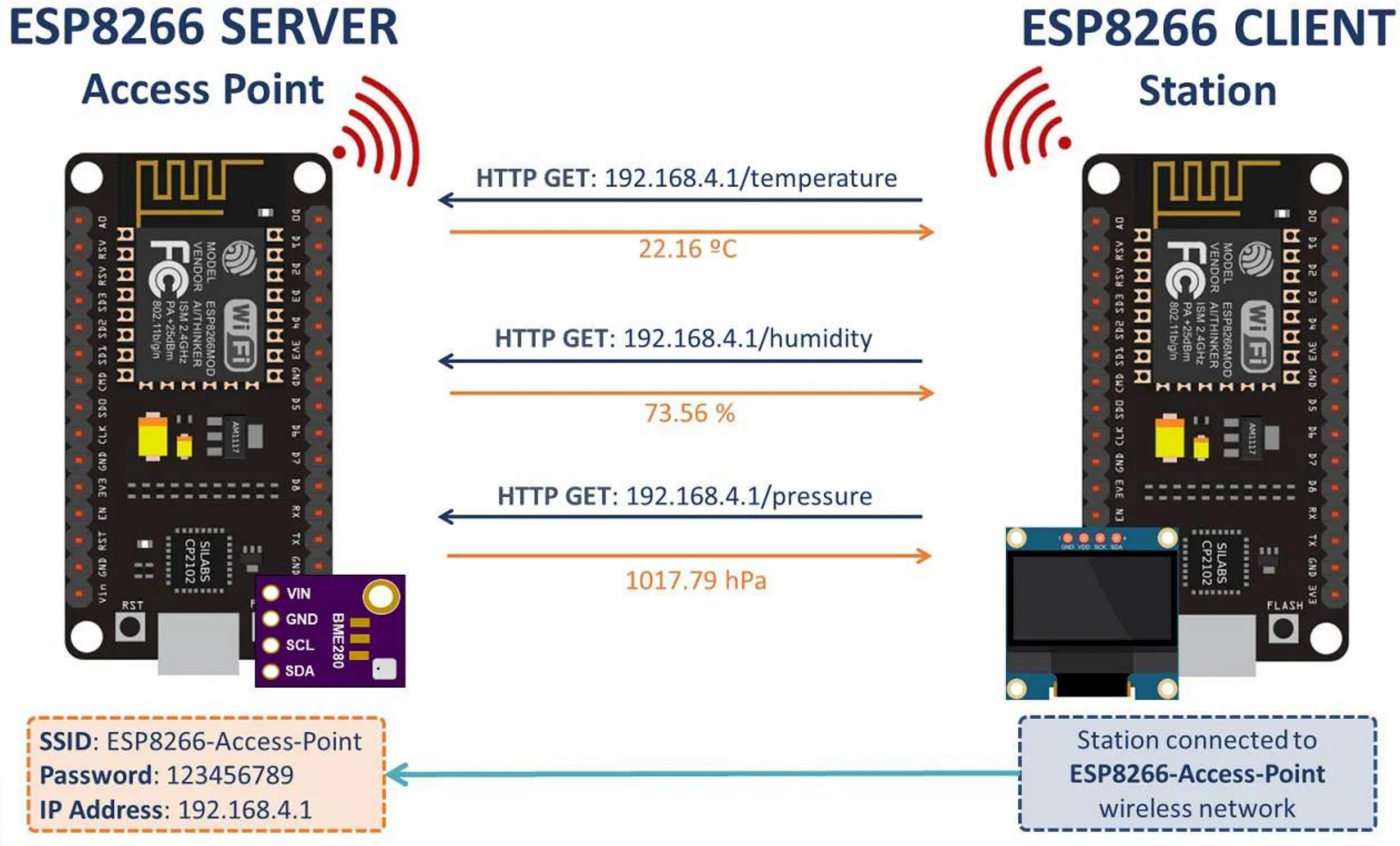
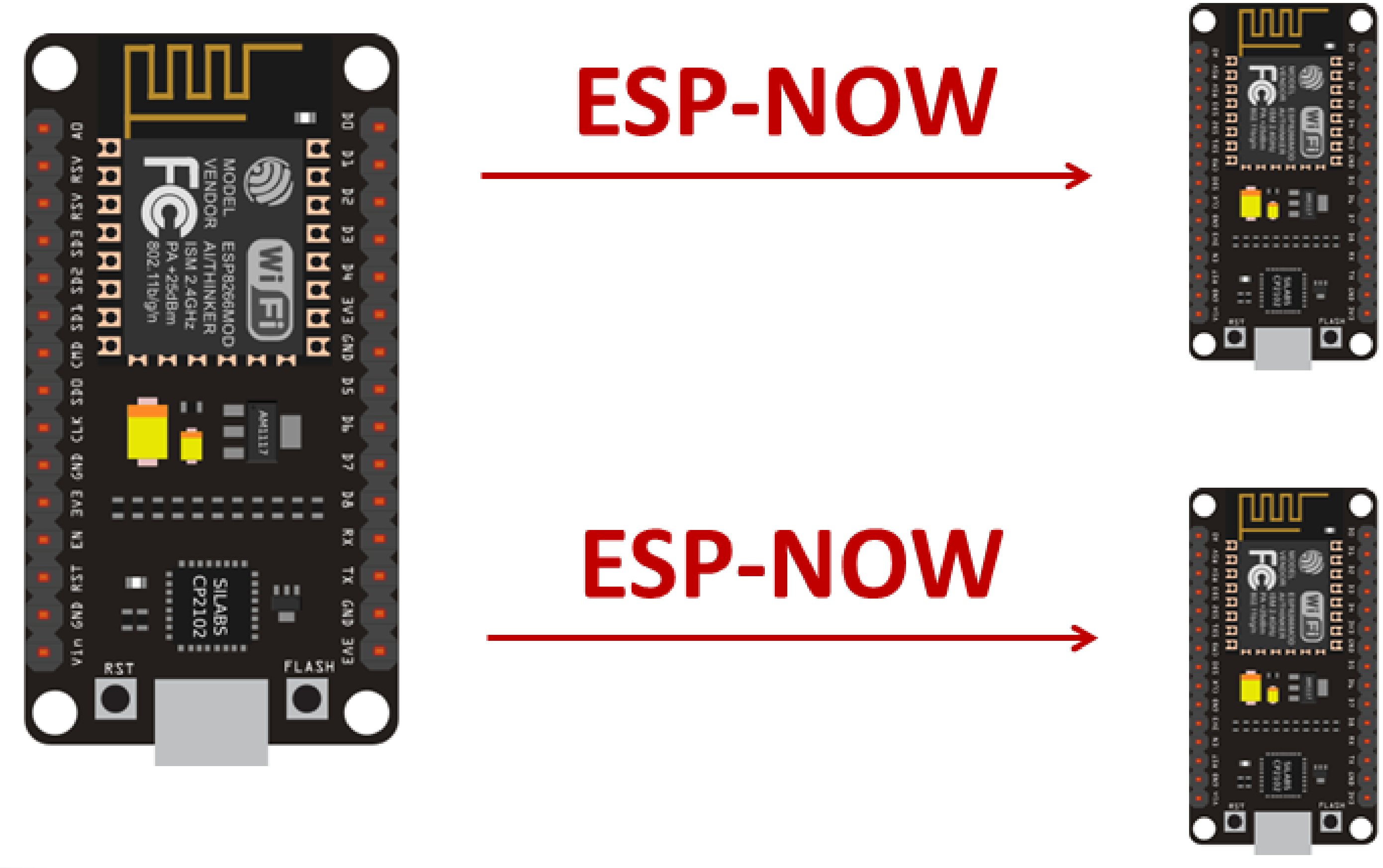


Image Source- https://bit.ly/3ywa1fp

# One-to-many Communication of NodeMCUs



ESP-NOW

ESP-NOW

MODEL ESP8266MOD
VENDOR AITHINKER
ISM 2.4GHz
PA +25dBm
802.11b/g/n
Wi Fi
AMS1117
SILABS CP2102
RST
FLASH
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3
A0 RSV RSV SD3 SD2 SD1 CMD SDO CLK GND 3V3 EN RST GND Vin

Image Source- https://bit.ly/3CLA6cM

# Many-to-one Communication of NodeMCUs

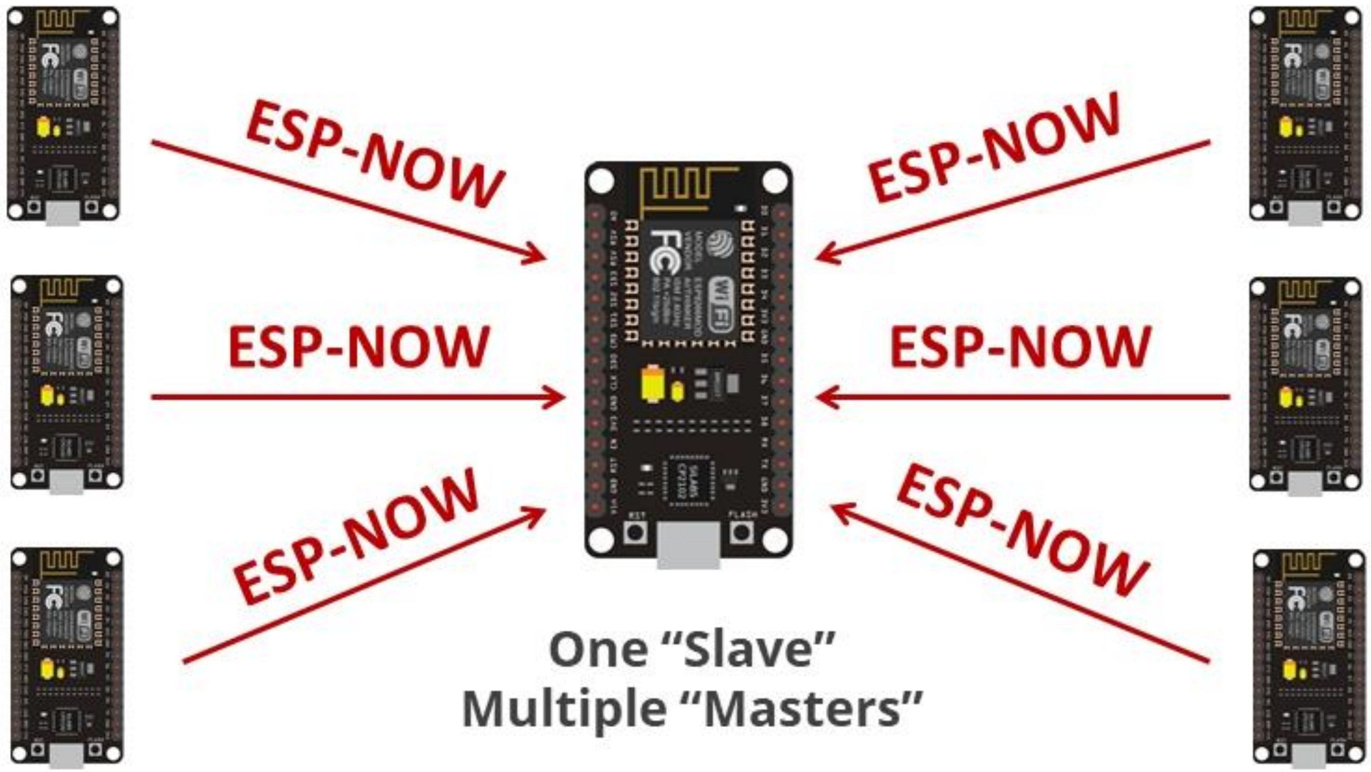


Image Source- https://bit.ly/3MiOOva

# Mesh Communication of NodeMCUs

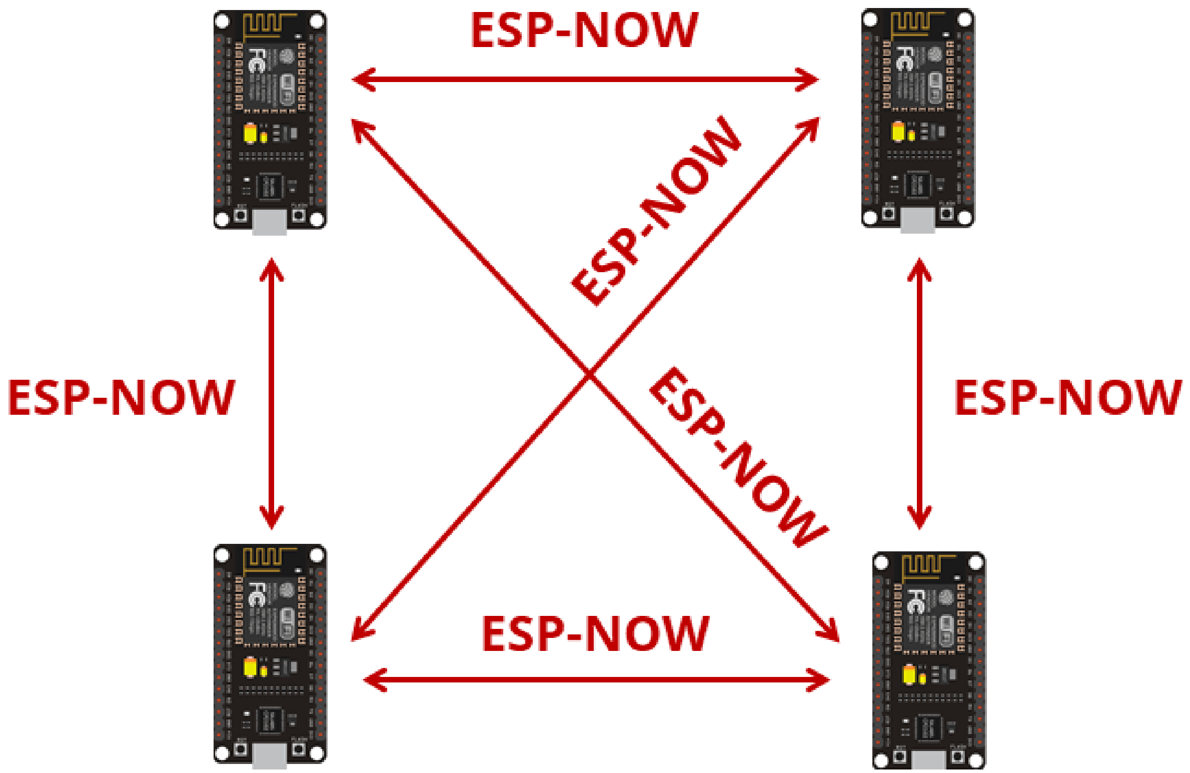
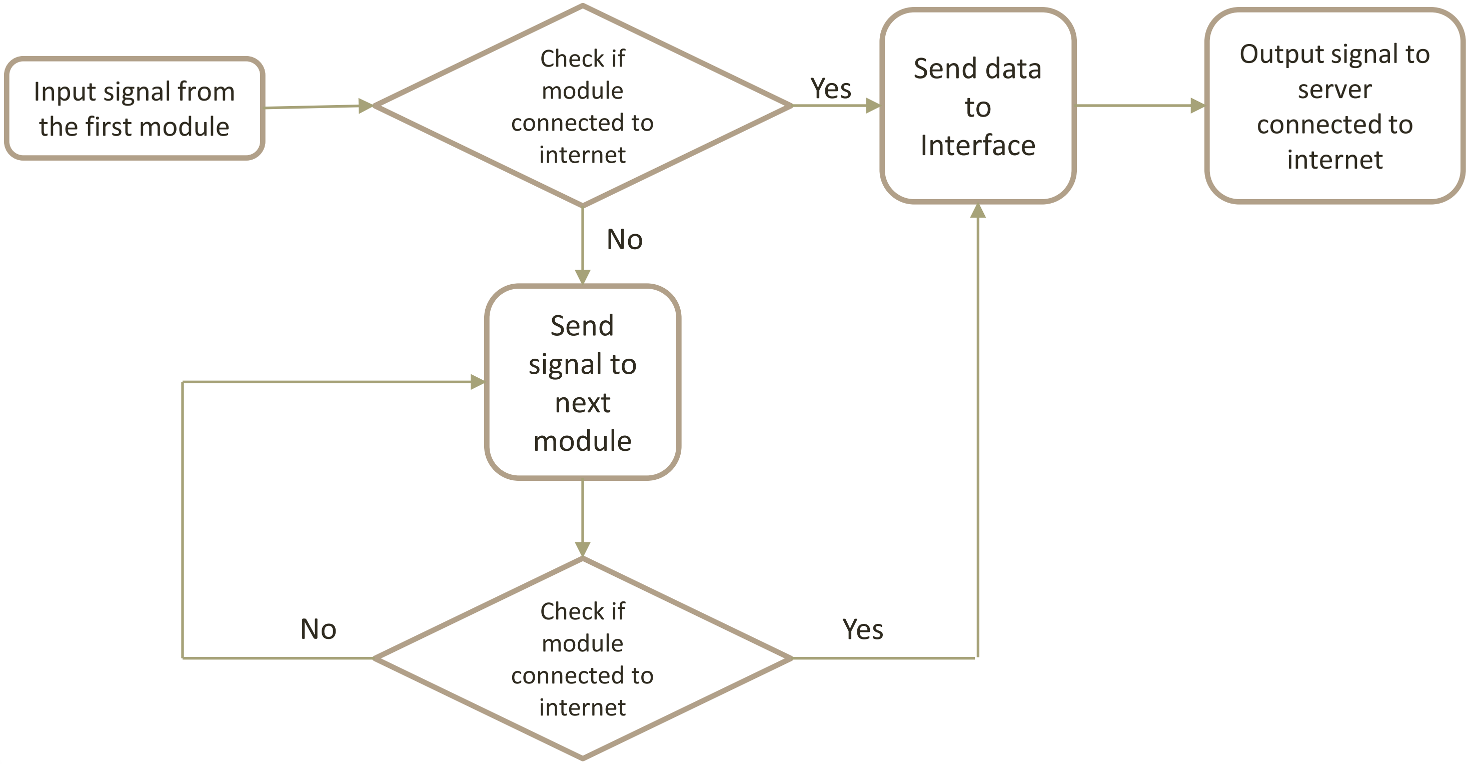


Image Source- https://bit.ly/3CkdRJJ

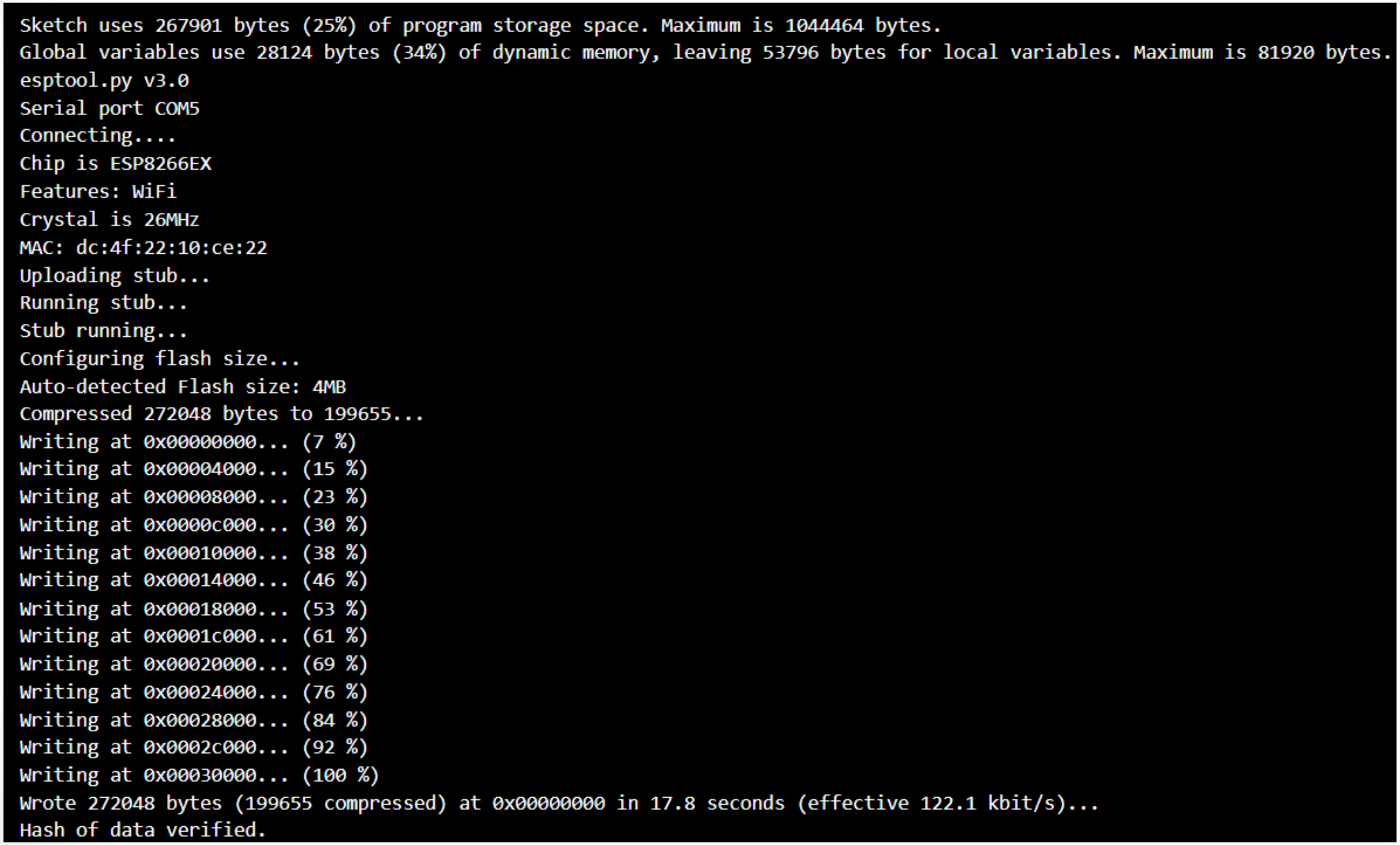
# Flow Chart


Input signal from the first module
Check if module connected to internet
Yes
Send data to Interface
Output signal to server connected to internet
No
Send signal to next module
No
Check if module connected to internet
Yes

# Test Code - 1

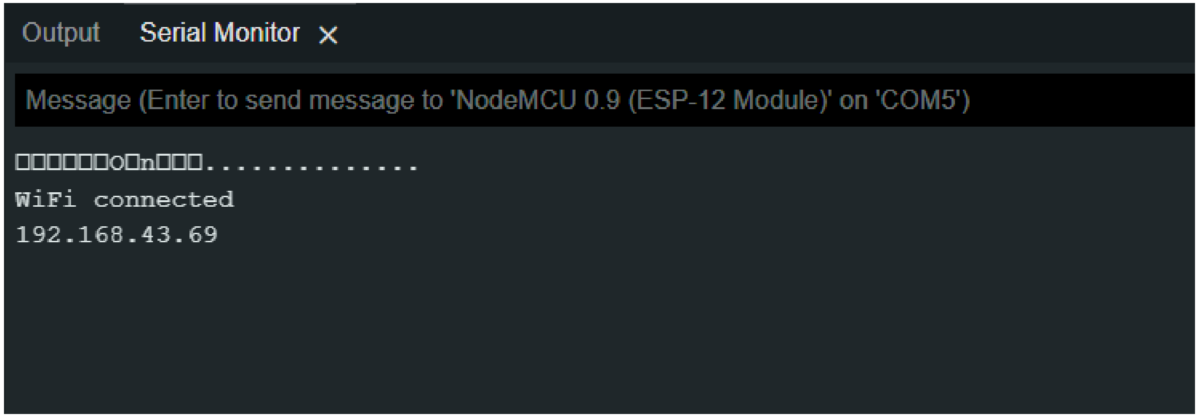
```
//Arduino test code for printing IP assigned by Router.

#include "ESP8266WiFi.h"

// WiFi parameters to be configured
const char* ssid = "HARRY-Redmiy2"; // Write here your router's username
const char* password = "9837080356"; // Write here your router's passward

void setup(void)
{
  Serial.begin(9600);
  // Connect to WiFi
  WiFi.begin(ssid, password);

  // while wifi not connected yet, print '.'
  // then after it connected, get out of the loop
  while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
  }
  //print a new line, then print WiFi connected and the IP address
  Serial.println("");
  Serial.println("WiFi connected");
  // Print the IP address
  Serial.println(WiFi.localIP());

}
void loop() {
  // Nothing
}
```

# Test Code - 1 Output



```
Sketch uses 267901 bytes (25%) of program storage space. Maximum is 1044464 bytes.
Global variables use 28124 bytes (34%) of dynamic memory, leaving 53796 bytes for local variables. Maximum is 81920 bytes.
esptool.py v3.0
Serial port COM5
Connecting....
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: dc:4f:22:10:ce:22
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 272048 bytes to 199655...
Writing at 0x00000000... (7 %)
Writing at 0x00004000... (15 %)
Writing at 0x00008000... (23 %)
Writing at 0x0000c000... (30 %)
Writing at 0x00010000... (38 %)
Writing at 0x00014000... (46 %)
Writing at 0x00018000... (53 %)
Writing at 0x0001c000... (61 %)
Writing at 0x00020000... (69 %)
Writing at 0x00024000... (76 %)
Writing at 0x00028000... (84 %)
Writing at 0x0002c000... (92 %)
Writing at 0x00030000... (100 %)
Wrote 272048 bytes (199655 compressed) at 0x00000000 in 17.8 seconds (effective 122.1 kbit/s)...
Hash of data verified.
```

# Test Code – 1 Output (Cont…)



Message (Enter to send message to 'NodeMCU 0.9 (ESP-12 Module)' on 'COM5')

```
□□□□□□o□n□□□..............
WiFi connected
192.168.43.69
```

# Test Code - 2 Algorithm

```
# Test code using MicroPython on NodeMCU for GPIO Interface


Activate.WLAN()

Enable.ESPNOW()

Enable.ESPNOW.P2P()

peer  = b'\xa0\x20\xa6\x14\x68\xc6’

ESPNOW.add_peer(peer)

def SEND():
    ESPNOW.send(peer, data)
     print("Data Sent successfully.")

def RECV():
    while True:
            host, msg = ESPNOW.recv()
                  print("Data Received Successfully")
                   break
```

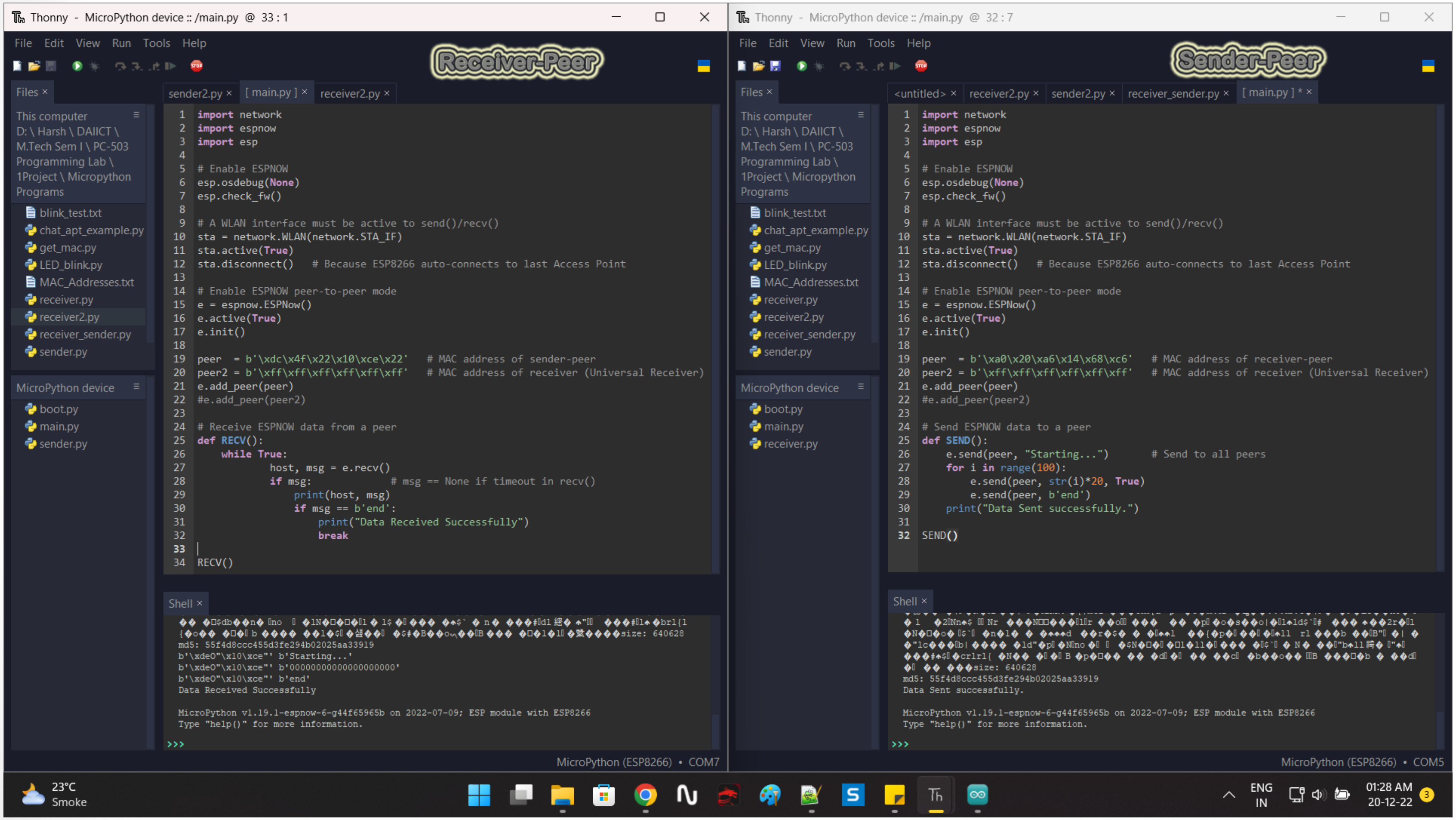
# Observations

- Writing code in MicroPython requires less memory as compared to Arduino IDE code.
- MicroPython provides a vast collection of libraries as compared to Arduino IDE.
- MicroPython code requires interpreter while Arduino IDE code is directly compiled and uploaded.
- MicroPython code can even be run using REPL without uploading it on device.
- Serial Communication through MicroPython is relatively easy as compared to Arduino IDE.

# Results

- 1 – way communication between 2 NodeMCUs successfully established using both Arduino and MicroPython.
- 2 – way communication between 2 NodeMCUs successfully established using both Arduino and MicroPython.
- Multi – way communication algorithm designed and analyzed for MicroPython.

# Result (Cont…)



Receiver-Peer:

```python
import network
import espnow
import esp

# Enable ESPNOW
esp.osdebug(None)
esp.check_fw()

# A WLAN interface must be active to send()/recv()
sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.disconnect()   # Because ESP8266 auto-connects to last Access Point

# Enable ESPNOW peer-to-peer mode
e = espnow.ESPNow()
e.active(True)
e.init()

peer  = b'\xdc\x4f\x22\x10\xce\x22'   # MAC address of sender-peer
peer2 = b'\xff\xff\xff\xff\xff\xff'   # MAC address of receiver (Universal Receiver)
e.add_peer(peer)
#e.add_peer(peer2)

# Receive ESPNOW data from a peer
def RECV():
    while True:
            host, msg = e.recv()
            if msg:             # msg == None if timeout in recv()
                print(host, msg)
                if msg == b'end':
                    print("Data Received Successfully")
                    break

RECV()
```

```
�� �□$db��n� no  �1N�□�□�1 � 1$ � ��� ��$` � n� ���#dl � �"  ���#1�brl{l
{�o�� �□� b ���� ��1�$ ���� �$#�B��o��B ��� �□�1�1 �����size: 640628
md5: 55f4d8ccc455d3fe294b02025aa33919
b'\xdeO"\x10\xce"' b'Starting...'
b'\xdeO"\x10\xce"' b'00000000000000000000'
b'\xdeO"\x10\xce"' b'end'
Data Received Successfully

MicroPython v1.19.1-espnow-6-g44f65965b on 2022-07-09; ESP module with ESP8266
Type "help()" for more information.

>>>
```

Sender-Peer:

```python
import network
import espnow
import esp

# Enable ESPNOW
esp.osdebug(None)
esp.check_fw()

# A WLAN interface must be active to send()/recv()
sta = network.WLAN(network.STA_IF)
sta.active(True)
sta.disconnect()   # Because ESP8266 auto-connects to last Access Point

# Enable ESPNOW peer-to-peer mode
e = espnow.ESPNow()
e.active(True)
e.init()

peer  = b'\xa0\x20\xa6\x14\x68\xc6'   # MAC address of receiver-peer
peer2 = b'\xff\xff\xff\xff\xff\xff'   # MAC address of receiver (Universal Receiver)
e.add_peer(peer)
#e.add_peer(peer2)

# Send ESPNOW data to a peer
def SEND():
    e.send(peer, "Starting...")       # Send to all peers
    for i in range(100):
        e.send(peer, str(i)*20, True)
        e.send(peer, b'end')
    print("Data Sent successfully.")

SEND()
```

```
� 1  �2Nn$  Nr ���N□□���1r ��o ���  �� �p �o�s��o|�1ld$`#  ��� ���2r�1
�N�□�o� $` �n�1� � ����d ��r�$� � �1  ��{�p� �� �11 rl ���b ��B" �| �
�"lc���b| ���� �1d"�p �Nno � �$N�□� �□1�11� ��� �$` � N� ��"b11 � "
���#$ �crlrl{ �N�� � � B �p�□�� �� �d � �� ��c �b��o�� B ���□�b � ��d
� �� ���size: 640628
md5: 55f4d8ccc455d3fe294b02025aa33919
Data Sent successfully.

MicroPython v1.19.1-espnow-6-g44f65965b on 2022-07-09; ESP module with ESP8266
Type "help()" for more information.

>>>
```

2-way communication using MicroPython

# Discussion

- In 1-way communication, one device works as host and other as client. So, a device can either receive or send data.
- In 2-way communication, either device can send or receive data.
- ESP – NOW protocol is used to implement 2 – way communication.
- ESP – NOW protocol can be also used for Multi – way communication or Many – to – Many communication.

# Conclusion

- By using ESP – NOW protocol, 2 – way communication was successfully established. There will be communication between two and more devices at subsequent positions without the internet.
- It will help solve the connection dropout problem in tunnels or affected areas.

# Future Work

- We can go for more communication modes to make communication more effective and create a network of devices.
- A local network of devices can be used in communication for emergency, relief and for disaster management purposes.

# References

- Parihar, Yogendra Singh. (2019). Internet of Things and NodeMCU A review of use of NodeMCU ESP8266 in IoT products. 6. 1085. © 2019 JETIR June 2019, Volume 6, Issue 6
- P2P (Bilateral) Communication Between NodeMCU Esp8266 Boards Using Arduino IDE (doi:10.24193/subbphys.2020.08)

# Useful Resources

- https://docs.micropython.org/en/latest/esp8266/tutorial/index.html
- https://github.com/espressif/esp-now
- https://github.com/glenn20/micropython-espnow-images
- https://github.com/glenn20/micropython/tree/espnow-g20
- https://github.com/micropython/micropython/tree/master/ports/esp8266
- https://github.com/techiesms/ESPNOW-One-To-Many-Communication-codes