# **[Demo 8: How to use TCP/IP with Arduino ESP32](http://www.iotsharing.com/2017/05/tcp-udp-ip-with-esp32.html)**

**[1. Introduction](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)**

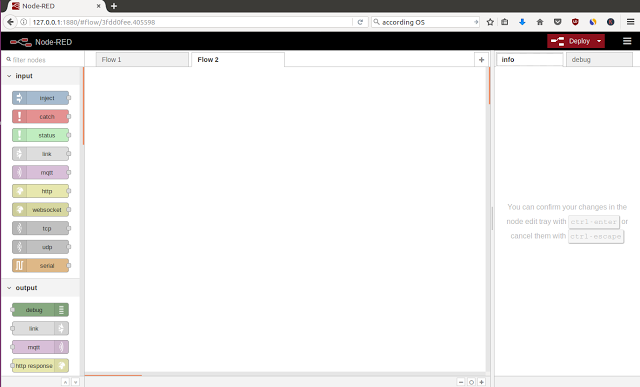
[In this tutorial, we will learn how to use Internet features of ESP32. This will become basic knowledge to apply for Internet of Things (IoT). We will learn some concepts such as: TCP/UDP-IP, Web Server, MQTT (Message Queuing Telemetry Transport) protocol. Beside that we also need a Tool/Application to test Internet features of ESP32, so we will use Node-RED. Node-RED is a flow-based programming tool, original developed by](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)[IBM’s Emerging Technology Services](https://emerging-technology.co.uk/)[team and now a part of the](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)[JS Foundation](https://js.foundation/)[.](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)

**[1.1 TCP/UDP - IP](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)**

[TCP and UDP are protocols that are used for sending data packets over the internet to an IP address.  
TCP (Transmission Control Protocol) cares about reliability. It means TCP ensure that the recipient will receive the packets in correct order and  no errors. The recipient sends ACKnowledge back to the sender to notify that it got the packets. If the sender doesn’t get ACKnowledge in a timeout period, it will retransmit the packets to the recipient until the recipient  got the packets. The retransmission just occurs in a specific number of times not forever. Many high layer application protocols use TCP/IP to get to the Internet. They are Hypertext Transfer Protocol (HTTP), the File Transfer Protocol (FTP), Telnet (Telnet), and the Simple Mail Transfer Protocol (SMTP). TCP/IP uses the client/server model in which server will wait and listen for a connection from client.   
UDP (User Datagram Protocol) does not care about reliability. It means the sender won’t ensure the recipient received the packet. If the recipient lost some UDP packets, the recipient can’t ask for those packets again. But losing all overhead will make the communication more quickly.  
In this session, we will learn how to make ESP32 as a TCP client or server (2 demos). I will also describe the requirements for 2 demos here:  
- Demo 8.1: when ESP32 is in server mode it will wait for connection, data (a “hello world” string) from client and then print this data to Terminal.  
- Demo 8.2: when ESP32 is in client mode it will create a connection and send data (a “hello world” string) to server every 5 seconds.](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)

**[1.2 Node-RED](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)**

[In order to play around with TCP/UDP, I suggest we should use a tool called Node-RED  
About Node-RED, it is a Flow-based programming for the Internet of Things. To understand more about it you can read at: https://nodered.org/#features  
For installation, you can follow the steps (according to your OS) which are mention here:](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)<http://nodered.org/docs/getting-started/installation.html> [In order to run it, you can follow the steps here:](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)<https://nodered.org/docs/getting-started/running> [After running it, open your web browser an go to the address:](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)<http://127.0.0.1:1880/>

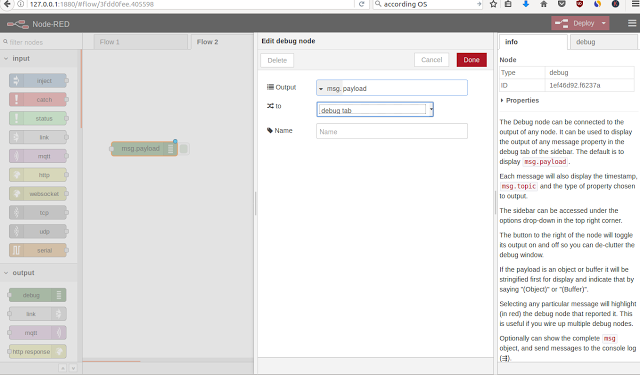
[](http://www.iotsharing.com/2017/05/how-to-use-arduino-esp32-to-display-oled.html)

**Figure: GUI of Node-RED**

**You can see on the left, there is a list of protocols and utilities nodes which is supplied by Node-RED such as: tcp, udp, http, debug, inject, …**

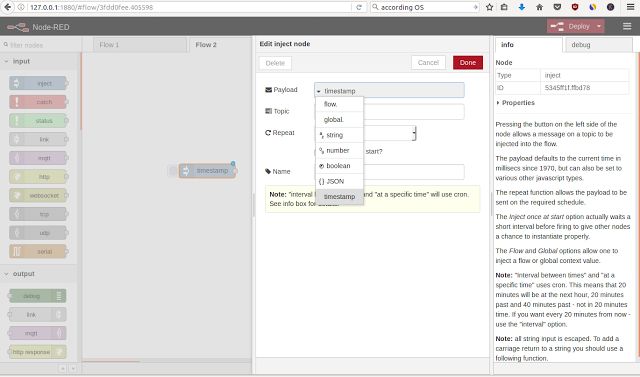
**In this session we just focus on some nodes like: tcp, udp, debug and inject.**

- **debug** node: **It can be used to display the output of any message property in the debug tab (on the top-right under Red Deploy button) of the sidebar. Now you can**drag and drop the node into the FLOW workspace and then double click on it, you will see all information about it.

**[](https://1.bp.blogspot.com/-9ybFHBgtW0o/WRuPh5c6MxI/AAAAAAAAD2g/NLPFDfmoUcc0LaPssJRqfA_106szAKUiACEw/s1600/internet2.png)**

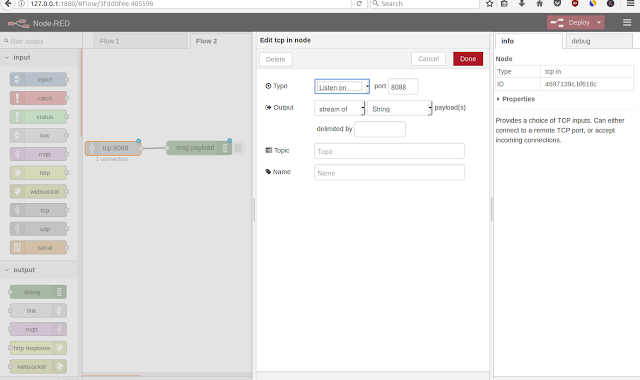
**Figure: information about debug node**

**- inject**node: Pressing the button on the left side of the node allows a message to be injected into the flow. You can drag and drop it into FLOW workspace for more information. Here message is under payload name, it can be string, boolean, json, ...

**[](https://2.bp.blogspot.com/-cu3i0wa-ORM/WRuPiUMf2-I/AAAAAAAAD2k/EXFtd5ZyaSc_H2p1SVeqsCnUdxtp_O4PACEw/s1600/internet3.png)**

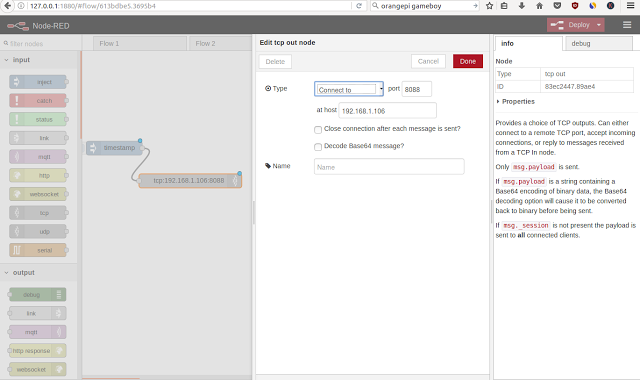
**Figure: GUI of inject node**

- **tcp input** node: this node can act as a client or server mode, but in our case we need this to to act as a server, accept the ESP32 client connection, receive data and forward to debug node for printing.

**[](https://1.bp.blogspot.com/-_sGsNYeg_90/WRuPjdhEDjI/AAAAAAAAD28/NsjgEKrJCGUWlSKPxsFLzAe8B_O0ptS3ACEw/s1600/internet5.png)**

**Figure: tcp input node as a server listen on port 8088 and output a string**

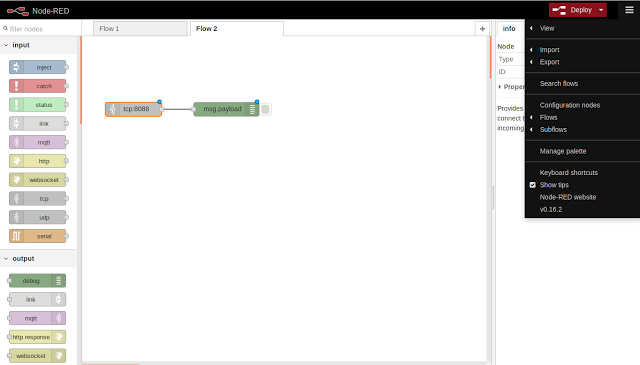
**- tcp output**node:this node can act as a client or server mode, but in our case we need this to to act as a client. It will connect, and forward data from inject node to ESP32 server.

[](https://1.bp.blogspot.com/-kb0C-VFRatk/WRuPjHav2GI/AAAAAAAAD28/W30vKnnm8xgPDm0TC9jCf4KsXFTM4v9VwCEw/s1600/internet6.png)

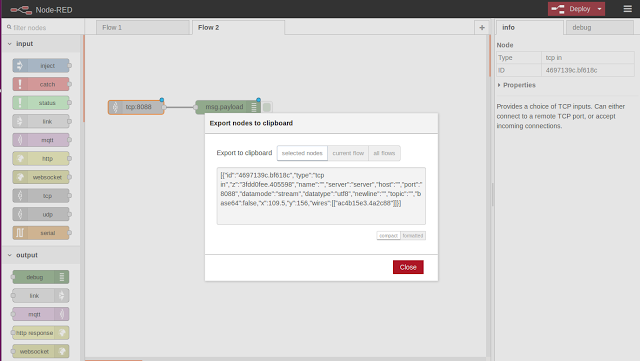
**Figure: tcp output node as a client connect to ESP32 server**

**Note:** Node-RED support **Import/Export**model feature so that you can Save (Export) model under JSON string format. And then Re-open (Import) it. This is the result of Export feature:

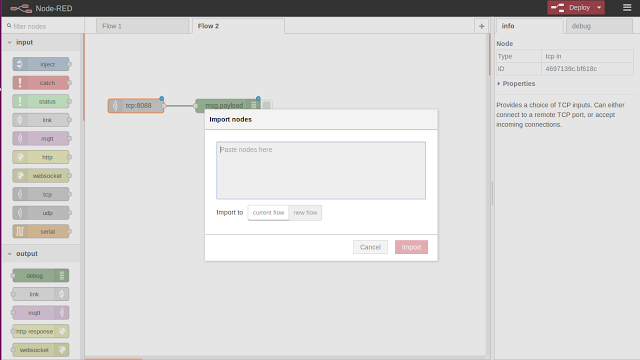
**[{"id":"4697139c.bf618c","type":"tcp in","z":"3fdd0fee.405598","name":"","server":"server","host":"","port":"8088","datamode":"stream","datatype":"utf8","newline":"","topic":"","base64":false,"x":109.5,"y":156,"wires":[["ac4b15e3.4a2c88"]]}]**

**[](https://4.bp.blogspot.com/-W0V9ojJCctg/WRuPjwOcXRI/AAAAAAAAD28/HSp7w1AZt-8RAb2rebZYeBzoQzMpi2BDQCEw/s1600/internet8.png)**

**Figure: open Import/Export menu (click top-right hamburger button**

**[](https://1.bp.blogspot.com/-k7xfwUAztlk/WRuPjio1evI/AAAAAAAAD28/EBg6JdHHcYoz8_Btof1DWNLFntBNmt9fACEw/s1600/internet7.png)**

**Figure: Export model to ClipBoard**

[](https://4.bp.blogspot.com/-5P02MYxgyKY/WRuPkV62bhI/AAAAAAAAD28/bplvnPvS95wDJGe3WKDtD72qKysF287EwCEw/s1600/internet9.png)

**Figure: Import model from ClipBoard (copy and paste the Exported string to textbox)**

That is enough for TCP/IP usage.

**1.3 TCP/IP ESP32**

ESP32 supply a library that make ESP32 become a client or server. There are some classes that you need to know:

**1.3.1 WiFi**  
We use this class to create a connection to WiFi. It is a static class and It has some important member functions:

- WiFi.begin(ssid, password): You should replace ssid and password with your WiFi ssid and password accordingly.  
- WiFi.status(): return the current status of WiFi (connect or not)  
- WiFi.localIP(): to get the current local IP address of ESP32.  
**1.3.2 WiFiClient**  
We use this class to create a TCP Client instance. It has some important member functions:  
- connect(host, port): connect to a TCP server at host (IP address) and port  
- print(data[]): send data to server  
- stop(): stop connection  
- available(): check whether data available for reading or not  
- read(): read one byte of data  
- read(uint8\_t \*buf, size\_t size): read size bytes of data in to buf  
- write(uint8\_t data): write one byte data  
- write(const uint8\_t \*buf, size\_t size): write size bytes in buf  
- connected(): is client connected

**1.3.3 WiFiServer**  
We use this class to create a TCP Server instance. It has some important member functions:  
- begin(): start Server  
- available(): is there a client want to connect.  
- accept(): accept the client connection  
- write(uint8\_t data): write one byte data  
- write(const uint8\_t \*buf, size\_t size): write size bytes in buf  
For more information you can refer WiFiClient.h and WiFiServer.h

**1.4 Demo 8.1 - ESP32 is TCP Server - Node-RED is TCP Client**  
ESP32 is in server mode it will wait for connection, data (a “hello world” string) from client and then print this data to Terminal.  
**Tips:** to get the IP address of your PC. Use the command below from Terminal:  
- Windows OS: use ipconfig command  
- Linux/Mac OS: use ifconfig command

**1.4.1 Node-RED model**  
Just **Import** the Json string below (remember to change the IP address of server in tcp node):  
**[{"id":"866fb572.d4ebc","type":"tcp out","z":"f5a25538.3c5518","host":"192.168.1.105","port":"8088","beserver":"client","base64":false,"end":false,"name":"","x":420.5,"y":163,"wires":[]}]**  
And **Click the button Deploy (top-right Red button)** to deploy the model.  
**Note:**  
- Wait until you see the string connected under tcp node then you can click the button on inject node

[IMG_263](https://2.bp.blogspot.com/-Nrs3UVwHcwo/WRuPd4wtCHI/AAAAAAAAD28/PhqRV43_3TIvtZy7Hy5u6nZBxCAc2NJ6QCEw/s1600/internet10.png)

**Figure: connected string under tcp node**

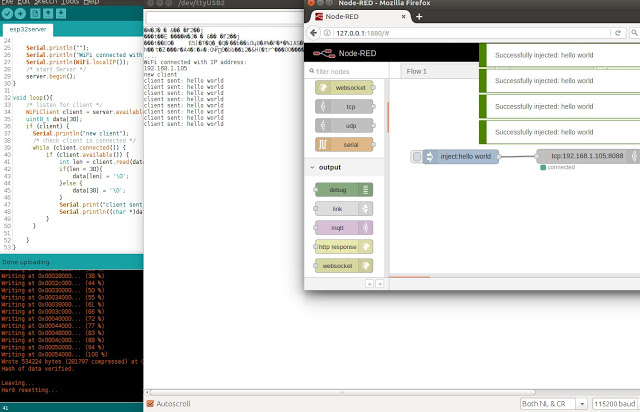
- If you see this error just wait until tcp node reconnect ESP32 Server

[IMG_264](https://2.bp.blogspot.com/-DFd6BQeXIL4/WRuPd_tcR6I/AAAAAAAAD28/3PwtwDu_oOUAx3eSQ2SM7wzBnYyKknZ1wCEw/s1600/internet11.png)

**Figure: error connection then wait tcp node reconnect**

**1.4.2 ESP32 side**  
Create **Arduino project**and Save as **esp32server** with code：

|  |
| --- |
| #include <WiFi.h>  const char\* ssid = "tenda";  const char\* password = "phong707";  /\* create a server and listen on port 8088 \*/  WiFiServer server(8088);  void setup()  {  Serial.begin(115200);  Serial.print("Connecting to ");  Serial.println(ssid);  /\* connecting to WiFi \*/  WiFi.begin(ssid, password);  /\*wait until ESP32 connect to WiFi\*/  while (WiFi.status() != WL\_CONNECTED) {  delay(500);  Serial.print(".");  }  Serial.println("");  Serial.println("WiFi connected with IP address: ");  Serial.println(WiFi.localIP());  /\* start Server \*/  server.begin();  }  void loop(){  /\* listen for client \*/  WiFiClient client = server.available();  uint8\_t data[30];  if (client) {  Serial.println("new client");  /\* check client is connected \*/  while (client.connected()) {  if (client.available()) {  int len = client.read(data, 30);  if(len < 30){  data[len] = '\0';  }else {  data[30] = '\0';  }  Serial.print("client sent: ");  Serial.println((char \*)data);  }  }  }  } |

[](https://2.bp.blogspot.com/-f9emyz5Bciw/WRuPgDOTbrI/AAAAAAAAD28/OKo28eesSPcKWHGQ58UtihucgljWdL6MwCEw/s1600/internet12.jpg)

**Figure: Result of Demo 8.1 (ESP32 is TCP server)**

**1.5 Demo 8.2 - ESP32 is TCP Client - Node-RED is TCP Server**

**1.5.1 Node-RED model**Just **Import** the Json string below (remember to change the IP address of server in tcp node): **[{"id":"4697139c.bf618c","type":"tcp in","z":"3fdd0fee.405598","name":"","server":"server","host":"","port":"8088","datamode":"stream","datatype":"utf8","newline":"","topic":"","base64":false,"x":109.5,"y":156,"wires":[["ac4b15e3.4a2c88"]]}]**

**1.5.2 ESP32**Create **Arduino project**and Save as **esp32client** with code**:**

|  |
| --- |
| #include <WiFi.h>  /\* change ssid and password according to yours WiFi\*/  const char\* ssid = "tenda";  const char\* password = "phong707";  /\*  \* This is the IP address of your PC  \* [Wins: use ipconfig command, Linux: use ifconfig command]  \*/  const char\* host = "192.168.1.106";  const int port = 8088;  void setup()  {  Serial.begin(115200);  Serial.print("Connecting to ");  Serial.println(ssid);  /\* connect to your WiFi \*/  WiFi.begin(ssid, password);  /\* wait until ESP32 connect to WiFi\*/  while (WiFi.status() != WL\_CONNECTED) {  delay(500);  Serial.print(".");  }  Serial.println("");  Serial.println("WiFi connected with IP address: ");  Serial.println(WiFi.localIP());  }  void loop()  {  delay(5000);  Serial.print("connecting to ");  Serial.println(host);  /\* Use WiFiClient class to create TCP connections \*/  WiFiClient client;    if (!client.connect(host, port)) {  Serial.println("connection failed");  return;  }  /\* This will send the data to the server \*/  client.print("hello world");  client.stop();  } |

1. **RESULT**

