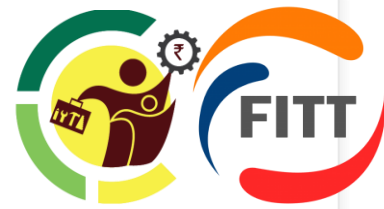


Lab Session on IoT

Lab Session

Agenda



- Lab Session Implementing IoT Appliance control using web server
- Data Packet Analysis Exercise using WireShark
- Preparation for Weekend Assignment

Let's Implement...



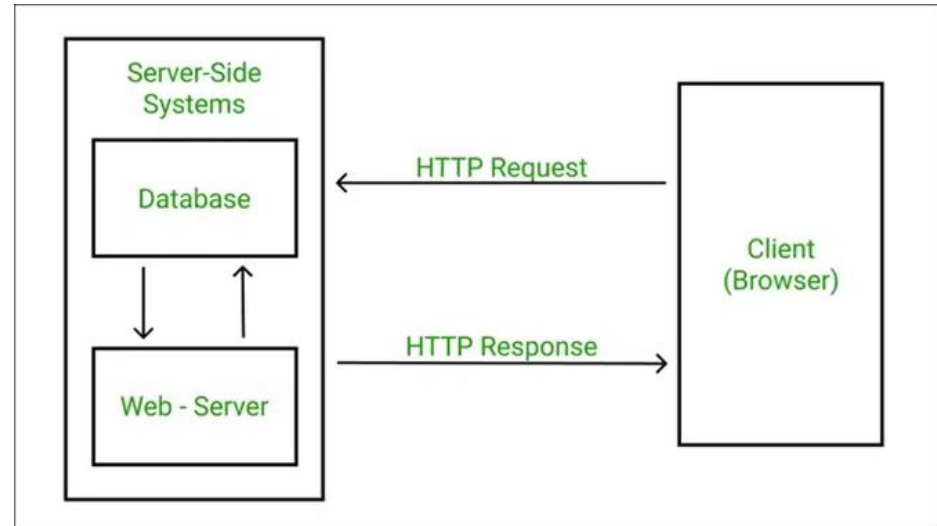
5G IoT communication Using HTTP Protocol

- IoT communication using Webserver with ESP32
- **Exercise** : LED Control using ESP32 board by creating Web Server

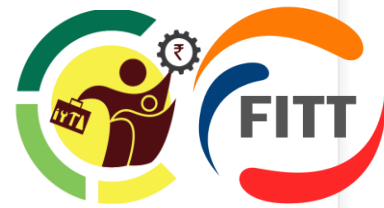
IoT communication Using HTTP Protocol

HTTP Protocol

- Explanation of HTTP (Hypertext Transfer Protocol)
- Key concepts: Request, Response, Methods (GET, POST)
- Importance in IoT communication



HTTP Protocol



- **Key concepts: Request, Response, Methods (GET, POST)**

- **Example Scenario: Request:**

http

Copy code

```
GET /api/data?id=123 HTTP/1.1
```

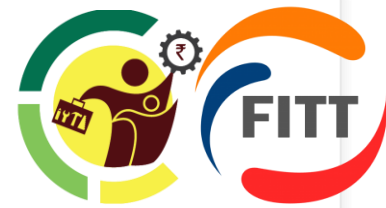
Host: example.com

Explanation:

The client is requesting data from the server.

The `id=123` parameter is included in the URL to specify the data needed.

HTTP Protocol



●Key concepts: Request, Response, Methods (GET, POST)

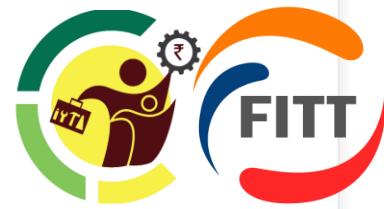
Response:

```
http
Copy code
HTTP/1.1 200 OK
Content-Type: application/json
{
  "id": 123,
  "name": "Example Data"
}
```

Explanation:

- The server responds with the requested data in JSON format.

Exercise : Appliance Control Over the Internet using HTTP Protocol



Introduction to ESP32

- Brief overview of ESP32 microcontroller

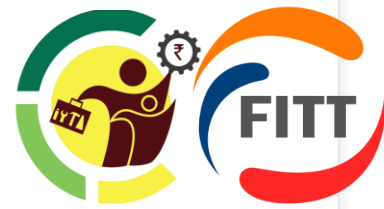
The ESP32 is a powerful microcontroller and Wi-Fi module, known for its versatility and capabilities.

- Features and capabilities

It features a dual-core processor, built-in Bluetooth, ample GPIO pins, and is widely used in IoT applications for its ability to connect to the internet and support various communication protocols.



Exercise : Appliance Control Over the Internet using HTTP Protocol



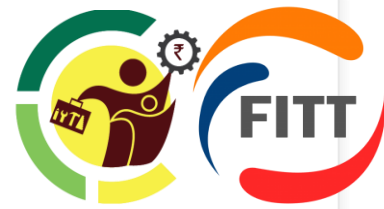
Source: <https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/>

- **ESP32 Web Server**
- creating a web server using ESP32
 - Install the Arduino IDE - **Covered in First week lab**
 - Add ESP32 board support -
 - **Tools-> Boards-> Boards Manager-> select ESP32**
 - Install necessary libraries
 - **Sketch-> Include Library-> Manage Libraries**
 - Configure ESP32 settings
 - **Tools -> Port-> Select Port**



Source for this exercise: <https://lastminuteengineers.com/creating-esp32-web-server-arduino-ide/>

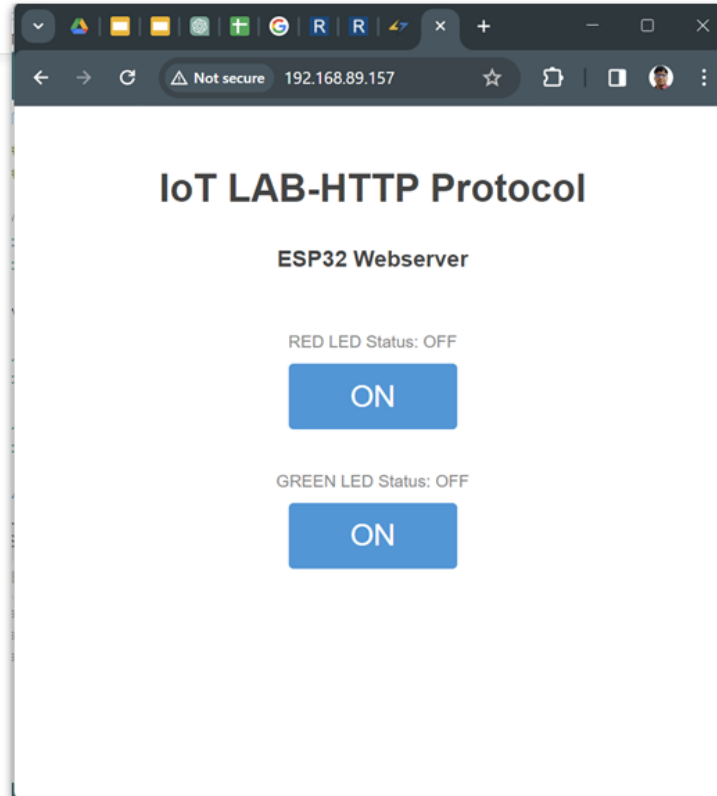
Exercise : Appliance Control Over the Internet using HTTP Protocol



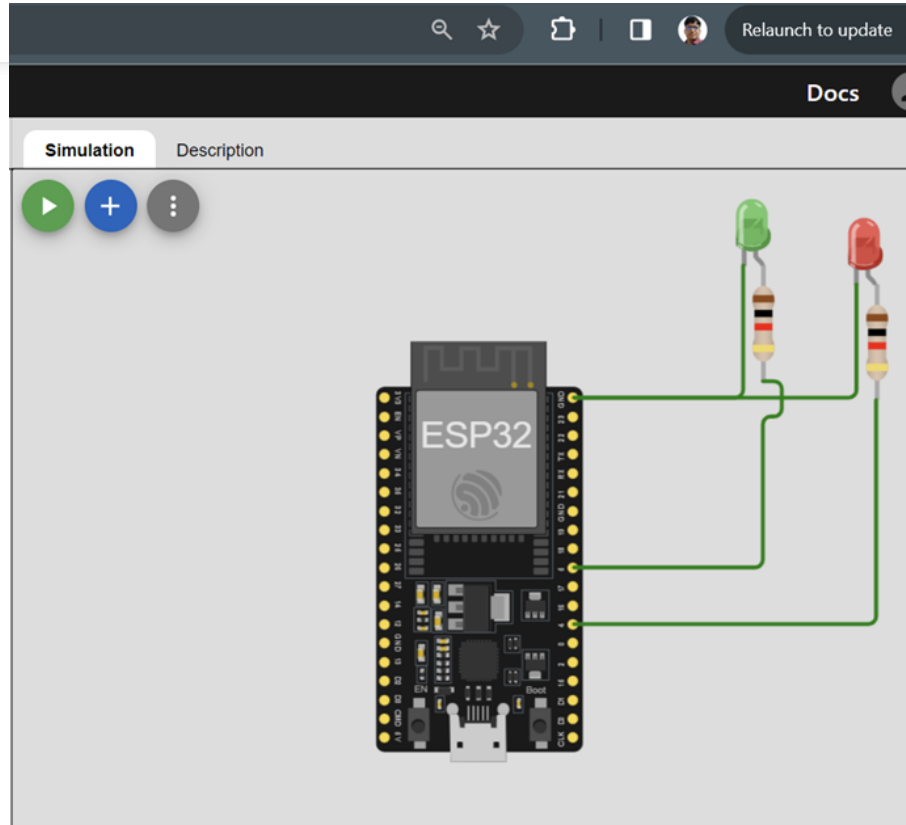
Testing the LED Control Web Server

- Steps to upload the code to ESP32
 - **Sketch-> Upload** (Make sure right board and port are selected on which the device is connected)
- Connecting ESP32 to the internet
 - Put Correct **SSID and Password** in the code and make sure that the wifi is discoverable
- Accessing the web server from a browser
 - Put the **URL in the browser** and hit Enter

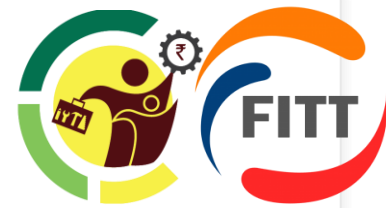
Exercise : Appliance Control Over the Internet using HTTP Protocol



Exercise : Appliance Control Over the Internet using HTTP Protocol



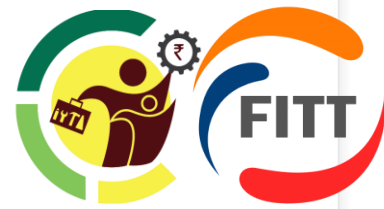
Exercise : Appliance Control using ESP32 webserver



ESP32WebServerTest.ino

```
1  #include <WiFi.h>
2  #include <WebServer.h>
3
4  /* Put your SSID & Password */
5  const char* ssid = "ESP32"; // Enter SSID here
6  const char* password = "12345678"; //Enter Password here
7
8  /* Put IP Address details */
9  IPAddress local_ip(192,168,1,1);
10 IPAddress gateway(192,168,1,1);
11 IPAddress subnet(255,255,255,0);
12
13 WebServer server(80);
14
15 uint8_t LED1pin = 4;
16 bool LED1status = LOW;
17
18 uint8_t LED2pin = 5;
19 bool LED2status = LOW;
20
```

Exercise : Appliance Control Over the Internet using HTTP Protocol

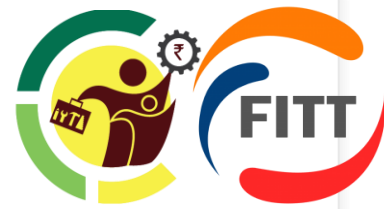


```
ESP32WebServerTest | Arduino IDE 2.2.1
File Edit Sketch Tools Help

uPesy ESP32 Wroom ...

ESP32WebServerTest.ino
--
21 void setup() {
22   Serial.begin(115200);
23   pinMode(LED1pin, OUTPUT);
24   pinMode(LED2pin, OUTPUT);
25
26   WiFi.softAP(ssid, password);
27   WiFi.softAPConfig(local_ip, gateway, subnet);
28   delay(100);
29
30   server.on("/", handle_OnConnect);
31   server.on("/led1on", handle_led1on);
32   server.on("/led1off", handle_led1off);
33   server.on("/led2on", handle_led2on);
34   server.on("/led2off", handle_led2off);
35   server.onNotFound(handle_NotFound);
36
37   server.begin();
38   Serial.println("HTTP server started");
39 }
40 void loop() {
41   server.handleClient();
42   if(LED1status)
43   {digitalWrite(LED1pin, HIGH);}
44   else
45   {digitalWrite(LED1pin, LOW);}
```

Exercise : Appliance Control Over the Internet using HTTP Protocol



ESP32WebServerTest | Arduino IDE 2.2.1

File Edit Sketch Tools Help



uPesy ESP32 Wroom ...



ESP32WebServerTest.ino



```
//  
40 void loop() {  
41   server.handleClient();  
42   if(LED1status)  
43     {digitalWrite(LED1pin, HIGH);}  
44   else  
45     {digitalWrite(LED1pin, LOW);}  
46  
47   if(LED2status)  
48     {digitalWrite(LED2pin, HIGH);}  
49   else  
50     {digitalWrite(LED2pin, LOW);}  
51 }  
52  
53 void handle_OnConnect() {  
54   LED1status = LOW;  
55   LED2status = LOW;  
56   Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");  
57   server.send(200, "text/html", SendHTML(LED1status,LED2status));  
58 }  
59  
60 void handle_led1on() {  
61   LED1status = HIGH;  
62   Serial.println("GPIO4 Status: ON");  
63   server.send(200, "text/html", SendHTML(true,LED2status));  
64 }
```



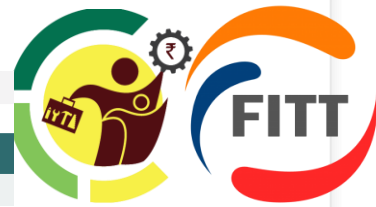
Output Serial Monitor x

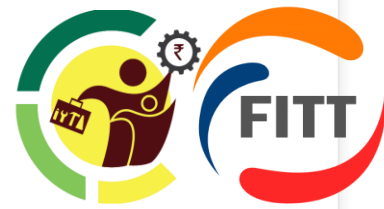
Exercise :

File Edit Sketch Tools Help

```
ESP32WebServerTest.ino
80 Serial.println("GPIO5 Status: OFF");
81 server.send(200, "text/html", SendHTML(LED1status,false));
82 }
83
84 void handle_NotFound(){
85     server.send(404, "text/plain", "Not found");
86 }
87
88 String SendHTML(uint8_t led1stat,uint8_t led2stat){
89     String ptr = "<!DOCTYPE html> <html>\n";
90     ptr += "<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
91     ptr += "<title>LED Control</title>\n";
92     ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";
93     ptr += "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";
94     ptr += ".button {display: block;width: 80px;background-color: #3498db;border: none;color: white;padding: 13px 30px;text-decoration: none;}\n";
95     ptr += ".button-on {background-color: #3498db;}\n";
96     ptr += ".button-on:active {background-color: #2980b9;}\n";
97     ptr += ".button-off {background-color: #34495e;}\n";
98     ptr += ".button-off:active {background-color: #2c3e50;}\n";
99     ptr += "p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
100    ptr += "</style>\n";
101    ptr += "</head>\n";
102    ptr += "<body>\n";
103    ptr += "h1>ESP32 Web Server</h1>\n";
104    ptr += "h3>Using Access Point(AP) Mode</h3>\n";
105
106    if(led1stat)
107    {ptr += "<p>LED1 Status: ON<p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";
108    else
109    {ptr += "<p>LED1 Status: OFF<p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";
110
111    if(led2stat)
112    {ptr += "<p>LED2 Status: ON<p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";
113    else
114    {ptr += "<p>LED2 Status: OFF<p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";
115
116    ptr += "</body>\n";
117    ptr += "</html>\n";
118    return ptr;
119 }
```

Output Serial Monitor x





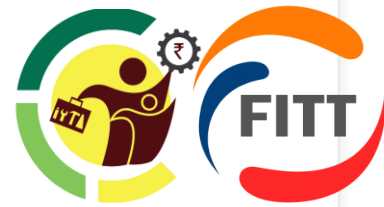
Lets see this in the video

https://drive.google.com/file/d/1crNYX_04rHnv0bIKowVJHKgMuZQ-VNmO/view?usp=drive_link

We discuss a lot about packet data transfer in last sessions

Let's Try Wireshark and see data packets in action...

Wireshark



Wireshark is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education.

It can capture live network traffic and also provides detailed information about different network protocols.

Download link: <https://www.wireshark.org/download.html>

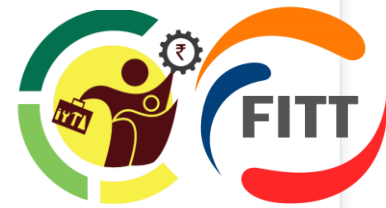
Requirements:

- Python (Optional-IDE) (prefer Miniconda [link](https://shorturl.at/sBHRY), <https://shorturl.at/sBHRY>)
- For CoAP Implementation
 - Download the files [here](#) or <https://shorturl.at/dfyD8>
 - Extract the files in a folder
 - Install coapthon3 library using pip (pip install CoAPthon3)
 - Open powershell or terminal in the folder and run “coapserver.py” [To run a file go the folder where file is stored]
 - File can be run using : python3 coapserver.py (Keep this running, for next steps open a new powershell or terminal window)

Req. Contd.



- Start wireshark before the next step
- Similarly after running server, client can be executed
- In new terminal/powershell window
- Various commands:
 - **python3 coapclient.py -o GET -p coap://127.0.0.1:5683/temp**
 - **python3 coapclient.py -o DISCOVER -p coap://127.0.0.1:5683/**
- more commands and outputs are discussed in next slides



Using WireShark for CoAP

Starting CoAP Client

```
geosysiot@geosysiot-H310M-S2-2-0: ~/Desktop/COAP x geosysiot@geosysiot-H310M-S2-2-0: ~/Desktop/COAP x geosysiot@
(base) geosysiot@geosysiot-H310M-S2-2-0:~/Desktop/COAP$ python3 coapclient.py -o GET -p coap://127.0.0.1:5683/temp
```

```
geosysiot@geosysiot-H310M-S2-2-0: ~/Desktop/COAP x geosysiot@geosysiot-H310M-S2-2-0: ~/Desktop/COAP x
(base) geosysiot@geosysiot-H310M-S2-2-0:~/Desktop/COAP$ python3 coapclient.py -o GET -p coap://127.0.0.1:5683/temp
Source: ('127.0.0.1', 5683)
Destination: None
Type: ACK
MID: 31524
Code: CONTENT
Token: 50eb
Payload:
{temp:100}

(base) geosysiot@geosysiot-H310M-S2-2-0:~/Desktop/COAP$
```

Capturing from any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

coap

No.	Time	Source	Destination	Protocol	Length	Info
1937	34.936785...	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:31524, GET, TKN:50 eb, /temp
1938	34.937274...	127.0.0.1	127.0.0.1	CoAP	61	ACK, MID:31524, 2.05 Content, TKN:50 eb, /temp

Frame 1937: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface any, id 0

- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- User Datagram Protocol, Src Port: 45172, Dst Port: 5683
 - Source Port: 45172
 - Destination Port: 5683
 - Length: 19
 - Checksum: 0xfe26 [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 81]
 - [Timestamps]
 - UDP payload (11 bytes)
- Constrained Application Protocol, Confirmable, GET, MID:31524
 - 01... .. = Version: 1
 - ..00... .. = Type: Confirmable (0)
 -0010 = Token Length: 2
 - Code: GET (1)
 - Message ID: 31524
 - Token: 50eb
 - Opt Name: #1: Uri-Path: temp
 - [Uri-Path: /temp]
 - [Response In: 1938](#)

0000 00 00 03 04 00 06 00 00 00 00 00 00 01 06 08 00
 0010 45 00 00 27 ed 3a 40 00 40 11 4f 89 7f 00 00 01 E...: @- @-0.....
 0020 7f 00 00 01 b0 74 16 33 00 13 fe 26 42 01 7b 24 ...t-3 ...&B-{\$
 0030 50 eb b4 74 65 6d 70 P...temp

Wireshark - Capture Information

ARP/RARP ICMPv6
 IPv4
 IPv6
 TCP
 UDP
 Other

11549 packets, 00:02:44

Stop Capture Close

Bytes 51-54: Uri-Path (coap.opt.uri_path)

Packets: 11549 · Displayed: 2 (0.0%)

Profile: Default

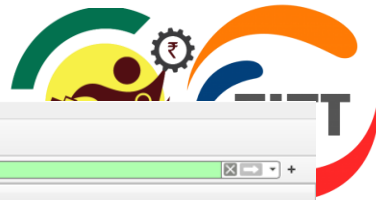
```
(base) geosysiot@geosysiot-H310M-S2-2-0:~/Desktop/COAP$ python3 coapclient.py -o DISCOVER -p coap://127.0.0.1:5683/  
Source: ('127.0.0.1', 5683)  
Destination: None  
Type: ACK  
MID: 32378  
Code: CONTENT  
Token: None  
Content-Type: 40  
Payload:  
</humidity>;obs,</temp>;obs,
```

Discover request

```
(base) geosysiot@geosysiot-H310M-S2-2-0:~/Desktop/COAP$ python3 coapclient.py -o POST -p coap://127.0.0.1:5683/temp -P {temp:40}  
Source: ('127.0.0.1', 5683)  
Destination: None  
Type: ACK  
MID: 22126  
Code: CREATED  
Token: 2d63  
Location-Path: temp  
Payload:  
None
```

```
(base) geosysiot@geosysiot-H310M-S2-2-0:~/Desktop/COAP$ python3 coapclient.py -o GET -p coap://127.0.0.1:5683/temp  
Source: ('127.0.0.1', 5683)  
Destination: None  
Type: ACK  
MID: 46385  
Code: CONTENT  
Token: ab9c  
Payload:  
{temp:40}
```


For Discover request



Wireshark interface showing a CoAP packet capture. The packet list shows two packets: a CoAP GET request (No. 284) and its acknowledgment (No. 285). The packet details pane shows the structure of the GET request, including the URI path `/.well-known/core`.

Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
284	5.503952686	127.0.0.1	127.0.0.1	CoAP	65	CON, MID:25181, GET, /.well-known/core
285	5.504554754	127.0.0.1	127.0.0.1	CoAP	79	ACK, MID:25181, 2.05 Content

Packet Details (Frame 284):

- Frame 284: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- User Datagram Protocol, Src Port: 41172, Dst Port: 5683
 - Source Port: 41172
 - Destination Port: 5683
 - Length: 29
 - Checksum: 0xfe30 [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 17]
 - [Timestamps]
 - UDP payload (21 bytes)
- Constrained Application Protocol, Confirmable, GET, MID:25181
 - 01.. = Version: 1
 - ..00 = Type: Confirmable (0)
 - 0000 = Token Length: 0
 - Code: GET (1)
 - Message ID: 25181
 - Opt Name: #1: Uri-Path: .well-known
 - Opt Desc: Type 11, Critical, Unsafe
 - 1011 = Opt Delta: 11
 - 1011 = Opt Length: 11
 - Uri-Path: .well-known
 - Opt Name: #2: Uri-Path: core
 - Opt Desc: Type 11, Critical, Unsafe
 - 0000 = Opt Delta: 0
 - 0100 = Opt Length: 4
 - Uri-Path: core
 - [Uri-Path: /.well-known/core]

Packet Bytes:

```
0000 00 00 03 04 00 06 00 00 00 00 00 00 00 08 00 .....  
0010 45 00 00 31 c2 a8 40 00 40 11 7a 11 7f 00 00 01 E..1..@. @.z....  
0020 7f 00 00 01 a0 d4 16 33 00 1d fe 30 40 01 62 5d .....3 ...0@.b]  
0030 bb 2e 77 65 6c 6c 2d 6b 6e 6f 77 6e 04 63 6f 72 ..well-k nown cor  
0040 65 e
```

Wireshark - Capture Information

1105 packets, 00:00:22

Stop Capture Close

Packets: 1105 - Displayed: 2 (0.2%)

Profile: Default

Post request capture response

The image displays a Wireshark network traffic capture window titled "Capturing from any". The main pane shows a list of captured packets, with packet 21750 selected. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
22084	338.161066535	127.0.0.1	127.0.0.1	CoAP	60	ACK, MID:46385, 2.05 Content, TKN:ab 9c, /temp
22083	338.160517566	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:46385, GET, TKN:ab 9c, /temp
21751	334.831245829	127.0.0.1	127.0.0.1	CoAP	55	ACK, MID:22126, 2.01 Created, TKN:2d 63, /temp
21750	334.830744732	127.0.0.1	127.0.0.1	CoAP	65	CON, MID:22126, POST, TKN:2d 63, /temp
21116	324.380384463	127.0.0.1	127.0.0.1	CoAP	61	ACK, MID:20385, 2.05 Content, TKN:e5 37, /temp
21115	324.379730273	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:20385, GET, TKN:e5 37, /temp
21015	321.731579571	127.0.0.1	127.0.0.1	CoAP	55	ACK, MID:56851, 2.01 Created, TKN:31 7f, /temp
21014	321.731088794	127.0.0.1	127.0.0.1	CoAP	66	CON, MID:56851, POST, TKN:31 7f, /temp
19645	298.190314601	127.0.0.1	127.0.0.1	CoAP	57	ACK, MID:49373, 2.05 Content, TKN:6a 73, /temp
19644	298.189238455	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:49373, GET, TKN:6a 73, /temp
18825	284.624993627	127.0.0.1	127.0.0.1	CoAP	55	ACK, MID:30219, 2.01 Created, TKN:9d 88, /temp
18824	284.624520933	127.0.0.1	127.0.0.1	CoAP	62	CON, MID:30219, POST, TKN:9d 88, /temp
15787	240.562248172	127.0.0.1	127.0.0.1	CoAP	79	ACK, MID:32378, 2.05 Content
15786	240.561746609	127.0.0.1	127.0.0.1	CoAP	65	CON, MID:32378, GET, /.well-known/core
1938	34.937274165	127.0.0.1	127.0.0.1	CoAP	61	ACK, MID:31524, 2.05 Content, TKN:50 eb, /temp
1937	34.936785856	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:31524, GET, TKN:50 eb, /temp

The packet details pane for packet 21750 shows the following structure:

- Frame 21750: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- User Datagram Protocol, Src Port: 55919, Dst Port: 5683
 - Source Port: 55919
 - Destination Port: 5683
 - Length: 29
 - Checksum: 0xfe30 [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 566]
 - [Timestamps]
 - UDP payload (21 bytes)
- Constrained Application Protocol, Confirmable, POST, MID:22126
 - 01... = Version: 1
 - ..00... = Type: Confirmable (0)
 -0010 = Token Length: 2
 - Code: POST (2)
 - Message ID: 22126
 - Token: 2d63
 - Opt Name: #1: Uri-Path: temp
 - End of options marker: 255
- Payload: Payload Content-Format: application/octet-stream (no Content-Format), Length: 9
 - Payload Desc: application/octet-stream
 - [Payload Length: 9]
 - [Uri-Path: /temp]
 - [Response In: 21751]
- Data (9 bytes)
 - Data: 7b74656d703a34307d
 - [Length: 9]

The packet bytes pane shows the raw data in hexadecimal and ASCII:

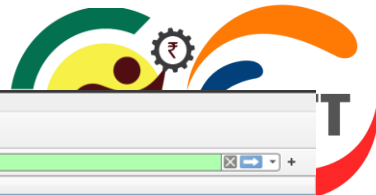
```
0000 00 00 03 04 00 06 00 00 00 00 00 00 0d f2 08 00 .....
0010 45 00 00 31 b8 57 40 00 40 11 84 62 7f 00 00 01 E..1.W@. @..b...
0020 7f 00 00 01 da 6f 16 33 00 1d fe 30 42 02 56 6e .....o-3...@B.Vn
0030 2d 63 b4 74 65 6d 70 ff 7b 74 65 6d 70 3a 34 30 -c-temp- {temp:40
0040 7d }
```

A "Wireshark - Capture Information" dialog box is open, showing the capture statistics:

- 34210 packets, 00:08:17
- Stop Capture
- Close

The status bar at the bottom indicates "Bytes 56-64: Data (data.data)" and "Packets: 34210 - Displayed: 16 (0.0%)". The profile is set to "Default".

Get request capture response



Wireshark interface showing a CoAP capture. The packet list on the left shows a sequence of CoAP messages. The packet details pane on the right shows the structure of a CoAP message, including the message type (ACK), MID, and Content. The packet bytes pane on the right shows the raw data of the selected packet.

Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
22084	338.161066535	127.0.0.1	127.0.0.1	CoAP	60	ACK, MID:46385, 2.05 Content, TKN:ab 9c, /temp
22083	338.160517566	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:46385, GET, TKN:ab 9c, /temp
21751	334.831245829	127.0.0.1	127.0.0.1	CoAP	55	ACK, MID:22126, 2.01 Created, TKN:2d 63, /temp
21750	334.830744732	127.0.0.1	127.0.0.1	CoAP	65	CON, MID:22126, POST, TKN:2d 63, /temp
21116	324.380384463	127.0.0.1	127.0.0.1	CoAP	61	ACK, MID:20385, 2.05 Content, TKN:e5 37, /temp
21115	324.379730273	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:20385, GET, TKN:e5 37, /temp
21015	321.731579571	127.0.0.1	127.0.0.1	CoAP	55	ACK, MID:56851, 2.01 Created, TKN:31 7f, /temp
21014	321.731088794	127.0.0.1	127.0.0.1	CoAP	66	CON, MID:56851, POST, TKN:31 7f, /temp
19645	298.190314601	127.0.0.1	127.0.0.1	CoAP	57	ACK, MID:49373, 2.05 Content, TKN:6a 73, /temp
19644	298.189238455	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:49373, GET, TKN:6a 73, /temp
18825	284.624993627	127.0.0.1	127.0.0.1	CoAP	55	ACK, MID:30219, 2.01 Created, TKN:9d 88, /temp
18824	284.624520933	127.0.0.1	127.0.0.1	CoAP	62	CON, MID:30219, POST, TKN:9d 88, /temp
15787	240.562248172	127.0.0.1	127.0.0.1	CoAP	79	ACK, MID:32378, 2.05 Content
15786	240.561746609	127.0.0.1	127.0.0.1	CoAP	65	CON, MID:32378, GET, /.well-known/core
1938	34.937274165	127.0.0.1	127.0.0.1	CoAP	61	ACK, MID:31524, 2.05 Content, TKN:50 eb, /temp
1937	34.936785856	127.0.0.1	127.0.0.1	CoAP	55	CON, MID:31524, GET, TKN:50 eb, /temp

Packet Details (Frame 22084):

- Frame 22084: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- User Datagram Protocol, Src Port: 5683, Dst Port: 59650
 - Source Port: 5683
 - Destination Port: 59650
 - Length: 24
 - Checksum: 0xfe2b [unverified]
 - [Checksum Status: Unverified]
 - [Stream index: 573]
 - [Timestamps]
 - UDP payload (16 bytes)
- Constrained Application Protocol, Acknowledgement, 2.05 Content, MID:46385
 - 01.. = Version: 1
 - ..10 = Type: Acknowledgement (2)
 - 0010 = Token Length: 2
 - Code: 2.05 Content (69)
 - Message ID: 46385
 - Token: ab9c
 - End of options marker: 255
- Payload: Payload Content-Format: application/octet-stream (no Content-Format), Length: 9
 - Payload Desc: application/octet-stream
 - [Payload Length: 9]
 - [Uri-Path: /temp]
 - [Request In: 22083](#)
 - [Response Time: 0.000548969 seconds]
- Data (9 bytes)
 - Data: 7b74656d703a34307d
 - [Length: 9]

Packet Bytes:

0000 00 00 03 04 00 06 00 00 00 00 00 00 00 00 00 00 00 08 00
0010 45 00 00 2c b9 a4 40 00 40 11 83 1a 7f 00 00 01 E...@...@.....
0020 7f 00 00 01 16 33 e9 02 00 18 fe 2b 62 45 b5 313...+bE.1
0030 ab 9c ff 7b 74 65 6d 70 3a 34 30 7d ...[temp :40]

Wireshark - Capture Information

33718 packets, 00:08:09

Stop Capture X Close

Bytes 51-59: Data (data.data) Packets: 33718 · Displayed: 16 (0.0%) Profile: Default

Requirements MQTT:

- Python (Optional-IDE) (prefer Miniconda [link](https://shorturl.at/sBHRY), <https://shorturl.at/sBHRY>)
- For MQTT Implementation
 - Download the files [here](https://shorturl.at/LSV36) or <https://shorturl.at/LSV36>
 - Extract the files in a folder
 - Download and Install mosquitto broker from <https://mosquitto.org/download/>
 - Run mosquitto -v in powershell/terminal to start broker
 - Install paho-mqtt library using pip (pip3 install paho-mqtt)
 - Start wireshark before next step
 - Open powershell or terminal in the folder and run “sub.py” [To run a file go the folder where file is stored]
 - File can be run using : python3 sub.py (Keep the file running, for next steps open new terminal/powershell window)
 - Open a new powershell/terminal window and run : python3 pub.py

Using Wireshark for MQTT



Wireshark interface showing MQTT traffic capture. The packet list pane displays several MQTT packets, with the selected packet (No. 197) highlighted. The packet details pane shows the structure of the MQTT Connect Command, including fields like Message Type, Protocol Name, Version, and Connect Flags. The packet bytes pane displays the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
197	2.877111117	127.0.0.1	127.0.0.1	MQTT	91	Connect Command
199	2.877185388	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack
201	2.877291933	127.0.0.1	127.0.0.1	MQTT	86	Subscribe Request (id=1) [SampleTopic]
202	2.877362162	127.0.0.1	127.0.0.1	MQTT	73	Subscribe Ack (id=1)
1361	22.897416373	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
1362	22.897603503	127.0.0.1	127.0.0.1	MQTT	70	Ping Response

Frame 197: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 57173, Dst Port: 1883, Seq: 1, Ack: 1, Len: 23

MQ Telemetry Transport Protocol, Connect Command

- Header Flags: 0x10, Message Type: Connect Command
 - 0001 = Message Type: Connect Command (1)
 - 0000 = Reserved: 0
- Msg Len: 21
- Protocol Name Length: 4
- Protocol Name: MQTT
- Version: MQTT v3.1.1 (4)
- Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag
 - 0... = User Name Flag: Not set
 - .0.. = Password Flag: Not set
 - ..0. = Will Retain: Not set
 - ...0 0... = QoS Level: At most once delivery (Fire and Forget) (0)
 - ...0 = Will Flag: Not set
 -1. = Clean Session Flag: Set
 -0 = (Reserved): Not set
- Keep Alive: 20
- Client ID Length: 9
- Client ID: SubClient

0000 00 00 03 04 00 06 00 00 00 00 00 00 78 f5 08 00X...
0010 45 00 00 4b 12 bb 40 00 40 06 29 f0 7f 00 00 01 E..K..@. @.)...
0020 7f 00 00 01 df 55 07 5b 62 18 e2 42 41 ad fb 3dU.[b..BA..=
0030 80 18 02 00 fe 3f 00 00 01 01 08 0a 58 4c d8 71?.. MQTT..q
0040 58 4c d8 71 10 15 00 04 4d 51 54 54 04 02 00 14 XL q... MQTT...
0050 00 09 53 75 62 43 6c 69 65 6e 74SubClient



File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

mqtt

No.	Time	Source	Destination	Protocol	Length	Info
197	2.877111117	127.0.0.1	127.0.0.1	MQTT	91	Connect Command
199	2.877185388	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack
201	2.877291933	127.0.0.1	127.0.0.1	MQTT	86	Subscribe Request (id=1) [SampleTopic]
202	2.877362162	127.0.0.1	127.0.0.1	MQTT	73	Subscribe Ack (id=1)
1361	22.897416373	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
1362	22.897603503	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
2398	42.915763263	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
2399	42.915847523	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
2588	46.913740117	127.0.0.1	127.0.0.1	MQTT	91	Connect Command
2590	46.913777820	127.0.0.1	127.0.0.1	MQTT	93	Publish Message [SampleTopic]
2592	46.913805759	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack
2594	46.913812221	127.0.0.1	127.0.0.1	MQTT	70	Disconnect Req
2596	46.913830225	127.0.0.1	127.0.0.1	MQTT	93	Publish Message [SampleTopic]
3230	62.929886341	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
3231	62.930071221	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
4683	82.951276830	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
4684	82.951468507	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
6649	102.971240052	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
6650	102.971428544	127.0.0.1	127.0.0.1	MQTT	70	Ping Response

Frame 199: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface any, id 0

- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 1883, Dst Port: 57173, Seq: 1, Ack: 24, Len: 4
- MQ Telemetry Transport Protocol, Connect Ack
 - Header Flags: 0x20, Message Type: Connect Ack
 - 0010 = Message Type: Connect Ack (2)
 - 0000 = Reserved: 0
 - Msg Len: 2
 - Acknowledge Flags: 0x00
 - Return Code: Connection Accepted (0)

0000 00 00 03 04 00 06 00 00 00 00 00 00 81 9e 08 00
0010 45 00 00 38 b2 0d 40 00 40 06 8a b0 7f 00 00 01 E...8...@...@.....
0020 7f 00 00 01 07 5b df 55 41 ad fb 3d 62 18 e2 59[.U A...=b...Y
0030 80 18 02 00 fe 2c 00 00 01 01 08 0a 58 4c d8 71XL.q
0040 58 4c d8 71 20 02 00 00



mqtt

No.	Time	Source	Destination	Protocol	Length	Info
197	2.877111117	127.0.0.1	127.0.0.1	MQTT		91 Connect Command
199	2.877185388	127.0.0.1	127.0.0.1	MQTT		72 Connect Ack
201	2.877291933	127.0.0.1	127.0.0.1	MQTT		86 Subscribe Request (id=1) [SampleTopic]
202	2.877362162	127.0.0.1	127.0.0.1	MQTT		73 Subscribe Ack (id=1)
1361	22.897416373	127.0.0.1	127.0.0.1	MQTT		70 Ping Request
1362	22.897603503	127.0.0.1	127.0.0.1	MQTT		70 Ping Response
2398	42.915763263	127.0.0.1	127.0.0.1	MQTT		70 Ping Request
2399	42.915847523	127.0.0.1	127.0.0.1	MQTT		70 Ping Response
2588	46.913740117	127.0.0.1	127.0.0.1	MQTT		91 Connect Command
2590	46.913777820	127.0.0.1	127.0.0.1	MQTT		92 Publish Message [SampleTopic]
2592	46.913805759	127.0.0.1	127.0.0.1	MQTT		72 Connect Ack
2594	46.913812221	127.0.0.1	127.0.0.1	MQTT		70 Disconnect Req
2596	46.913830225	127.0.0.1	127.0.0.1	MQTT		93 Publish Message [SampleTopic]
3230	62.929886341	127.0.0.1	127.0.0.1	MQTT		70 Ping Request
3231	62.930071221	127.0.0.1	127.0.0.1	MQTT		70 Ping Response

Frame 2588: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 40041, Dst Port: 1883, Seq: 1, Ack: 1, Len: 23

MQ Telemetry Transport Protocol, Connect Command

Header Flags: 0x10, Message Type: Connect Command

0001 = Message Type: Connect Command (1)

.... 0000 = Reserved: 0

Msg Len: 21

Protocol Name Length: 4

Protocol Name: MQTT

Version: MQTT v3.1.1 (4)

Connect Flags: 0x02, QoS Level: At most once delivery (Fire and Forget), Clean Session Flag

0... = User Name Flag: Not set

.0.. = Password Flag: Not set

..0. = Will Retain: Not set

...0 0... = QoS Level: At most once delivery (Fire and Forget) (0)

.... 0... = Will Flag: Not set

.... ..1. = Clean Session Flag: Set

.... ...0 = (Reserved): Not set

Keep Alive: 20

Client ID Length: 9

Client ID: PubClient

```

0000 00 00 03 04 00 06 00 00 00 00 00 00 68 00 08 00 .....h...
0010 45 00 00 4b 6c 95 40 00 40 06 d0 15 7f 00 00 01 E..Kl.@: @....
0020 7f 00 00 01 9c 69 07 5b 1e f3 d4 79 e2 b4 2b 83 ....i.[ ...y.+
0030 80 18 02 00 fe 3f 00 00 01 01 08 0a 58 4d 84 76 ....?...XM-v
0040 58 4d 84 75 10 15 00 04 4d 51 54 54 04 02 00 14 XM-u...MQTT...
0050 00 09 50 75 62 43 6c 69 65 6e 74 ..PubClient

```




File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help



mqtt

No.	Time	Source	Destination	Protocol	Length	Info
197	2.877111117	127.0.0.1	127.0.0.1	MQTT	91	Connect Command
199	2.877185388	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack
201	2.877291933	127.0.0.1	127.0.0.1	MQTT	86	Subscribe Request (id=1) [SampleTopic]
202	2.877362162	127.0.0.1	127.0.0.1	MQTT	70	Subscribe Ack (id=1)
1361	22.897416373	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
1362	22.897603503	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
2398	42.915763263	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
2399	42.915847523	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
2588	46.913740117	127.0.0.1	127.0.0.1	MQTT	91	Connect Command
2590	46.913777820	127.0.0.1	127.0.0.1	MQTT	93	Publish Message [SampleTopic]
2592	46.913805759	127.0.0.1	127.0.0.1	MQTT	72	Connect Ack
2594	46.913812221	127.0.0.1	127.0.0.1	MQTT	70	Disconnect Req
2596	46.913830225	127.0.0.1	127.0.0.1	MQTT	93	Publish Message [SampleTopic]
3230	62.929886341	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
3231	62.930071221	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
4683	82.951276830	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
4684	82.951468507	127.0.0.1	127.0.0.1	MQTT	70	Ping Response
6649	102.971240052	127.0.0.1	127.0.0.1	MQTT	70	Ping Request
6650	102.971428544	127.0.0.1	127.0.0.1	MQTT	70	Ping Response

- Frame 201: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface any, id 0
- Linux cooked capture v1
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 57173, Dst Port: 1883, Seq: 24, Ack: 5, Len: 18

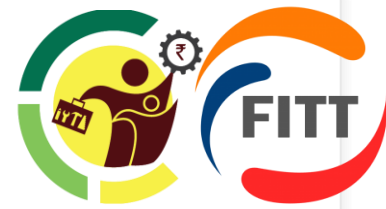
MQ Telemetry Transport Protocol, Subscribe Request

- Header Flags: 0x82, Message Type: Subscribe Request
- 1000 = Message Type: Subscribe Request (8)
- 0010 = Reserved: 2
- Msg Len: 16
- Message Identifier: 1
- Topic Length: 11
- Topic: SampleTopic
- Requested QoS: At most once delivery (Fire and Forget) (0)

0000	00 00 03 04 00 06 00 00	00 00 00 00 ab ae 08 00
0010	45 00 00 46 12 bd 40 00	40 06 29 f3 7f 00 00 01	E..F..@. @.).....
0020	7f 00 00 01 df 55 07 5b	62 18 e2 59 41 ad fb 41U.[b..YA..A
0030	80 18 02 00 fe 3a 00 00	01 01 08 0a 58 4c d8 71:.....XL q
0040	58 4c d8 71 82 10 00 01	00 0b 53 61 6d 70 6c 65	XL q.....Sample
0050	54 6f 70 69 63 00		Topic..

Lets see this in the video





Lets see this in the video

Links for video demos for CoAP, MQTT and installation help for wireshark and mosquitto.

CoAP-

- 1) https://drive.google.com/drive/folders/12Wyjdilz0sp1GcJxBxXV8cW6qYi3C4xb?usp=drive_link,
- 2) https://drive.google.com/drive/folders/17lc9R9EFZ53DjUsIVGaoczI7VbzIYK13?usp=drive_link,
- 3) https://drive.google.com/drive/folders/1Vjmh26exWFI7oBTG9An-W4eybPUxAoFe?usp=drive_link

Weekend Assignment



- 1. Replicate entire web server exercise using ESP32 in your college lab**
- 2. Download wireshark and perform and packet data analysis as shown today.**
- 3. Take screenshots for all steps you have done and prepare a document containing every step. This will be needed at the time of your submission.**

Thank you !

