MPSSDEGB
कौशल से रोजगार, समृद्धि अपार

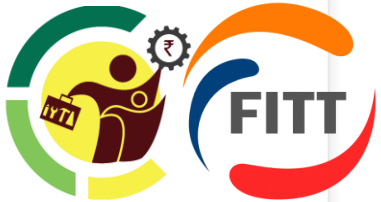
FITT
Foundation For Innovation And Technology Transfer

SANKALP
Ministry of Skill Development & Entrepreneurship

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

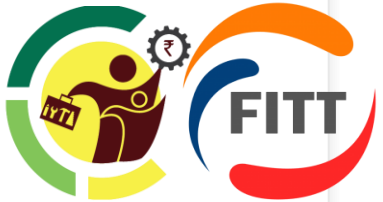
MPSSDEGB
कौशल से रोजगार, समृद्धि अपार

FITT
Foundation For Innovation And Technology Transfer

SANKALP
Ministry of Skill Development
& Entrepreneurship

Let’s Implement…

5G IoT communication Using HTTP Protocol



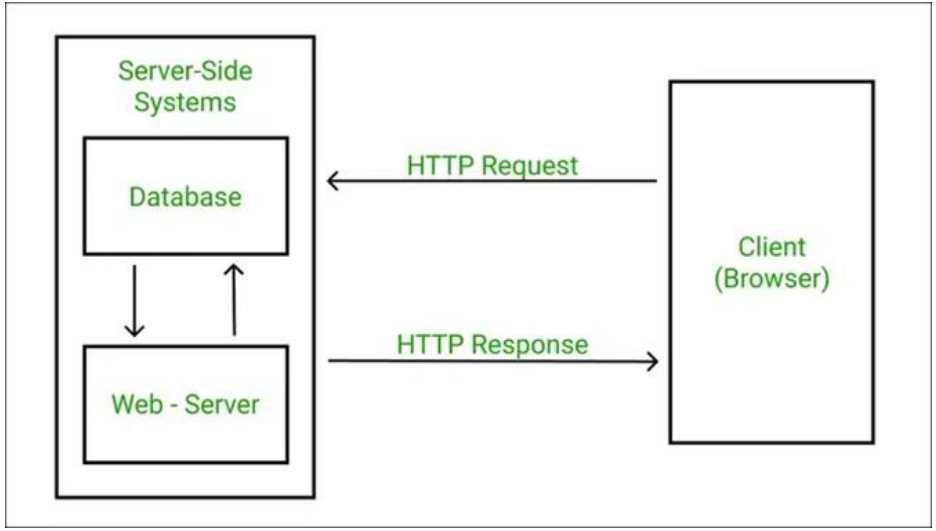
- IoT communication using Webserver with ESP32

- **Exercise** : LED Control using ESP32 board by creating Web Server

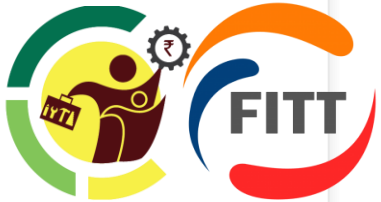
IoT communication Using HTTP Protocol

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:**

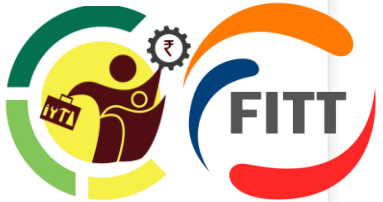
http
Copy code

```
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.

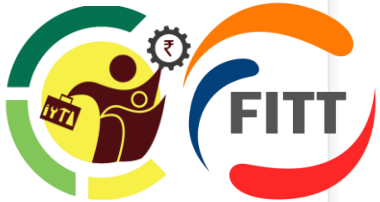
Exercise : Appliance Control Over the Internet using HTTP Protocol

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/

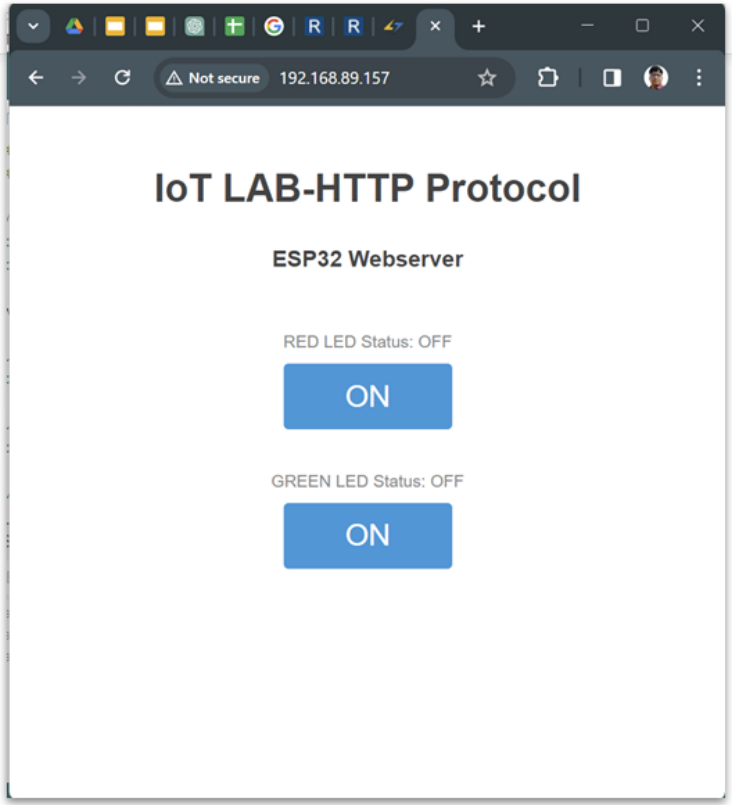
Exercise : Appliance Control Over the Internet using HTTP Protocol



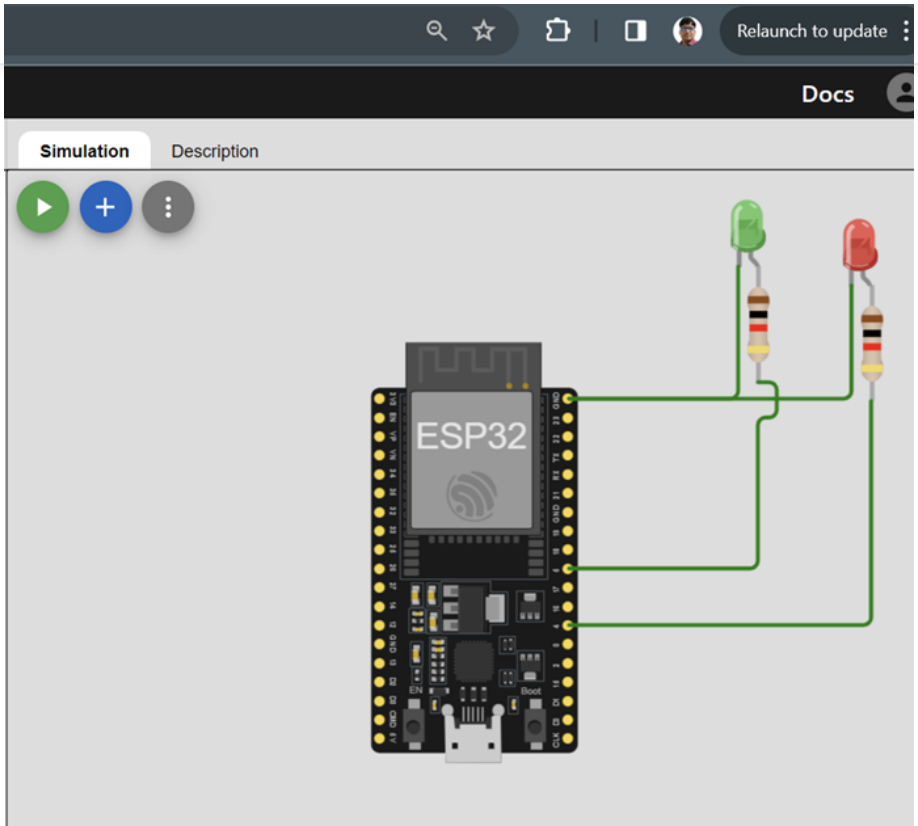
Testing the LED Control Web Server

- 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 browse**r and hit Enter

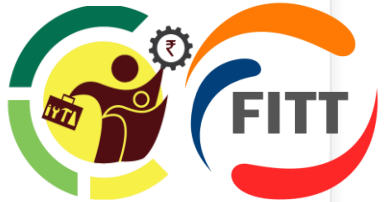
Exercise : Appliance Control Over the Internet using HTTP Protocol



Exercise : Appliance Control Over the Internet using HTTP Protocol



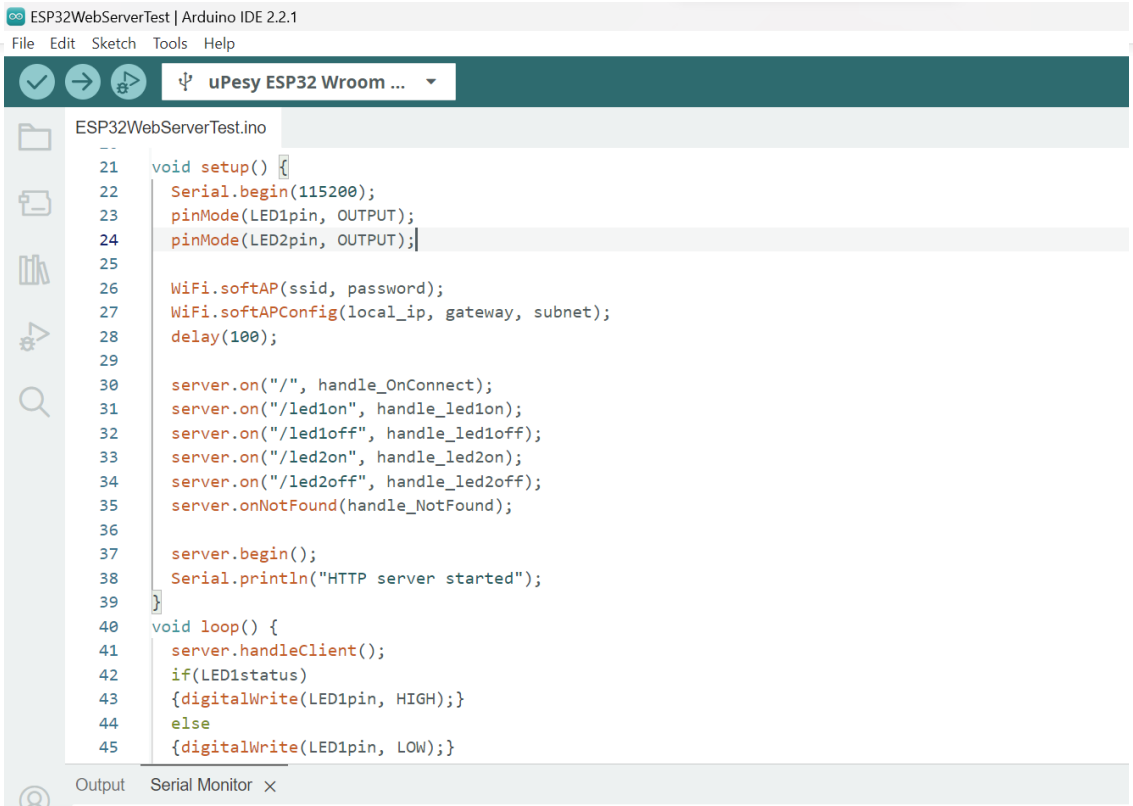
Exercise : Appliance Control using ESP32 webserver



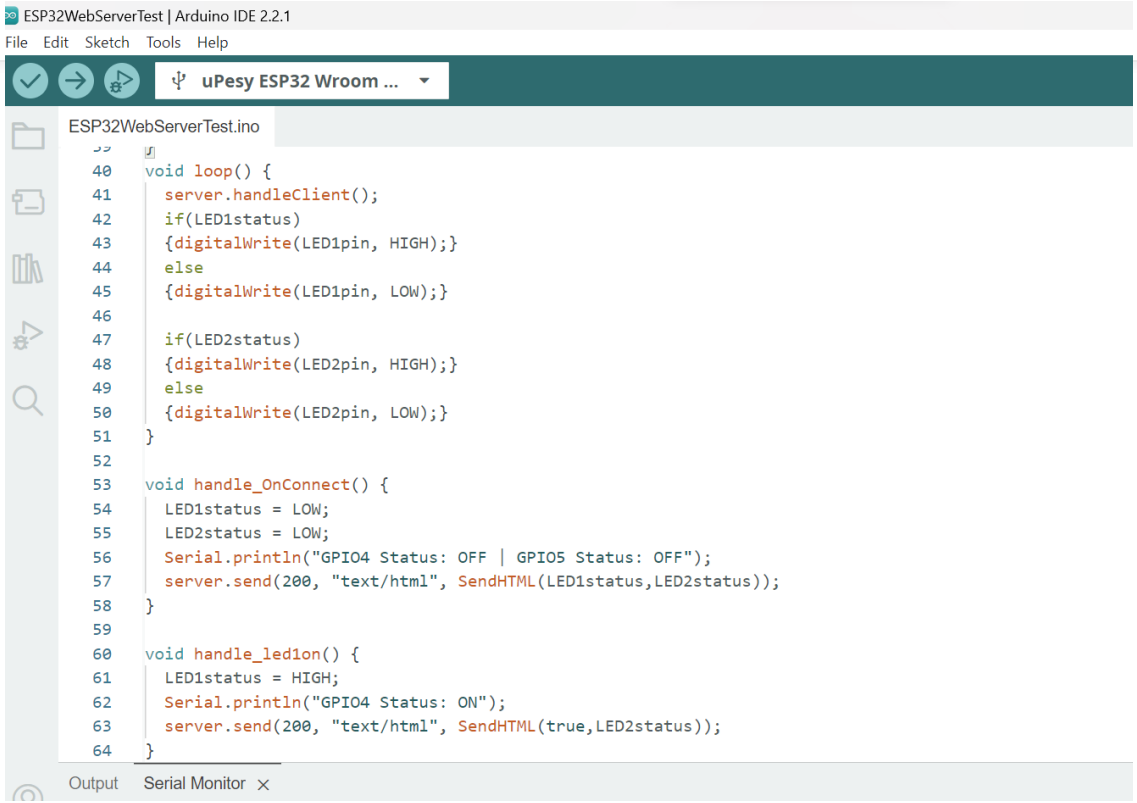
ESP32WebServerTest.ino

```
#include <WiFi.h>
#include <WebServer.h>

/* Put your SSID & Password */
const char* ssid = "ESP32";  // Enter SSID here
const char* password = "12345678";  //Enter Password here

/* Put IP Address details */
IPAddress local_ip(192,168,1,1);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);

WebServer server(80);

uint8_t LED1pin = 4;
bool LED1status = LOW;

uint8_t LED2pin = 5;
bool LED2status = LOW;

```

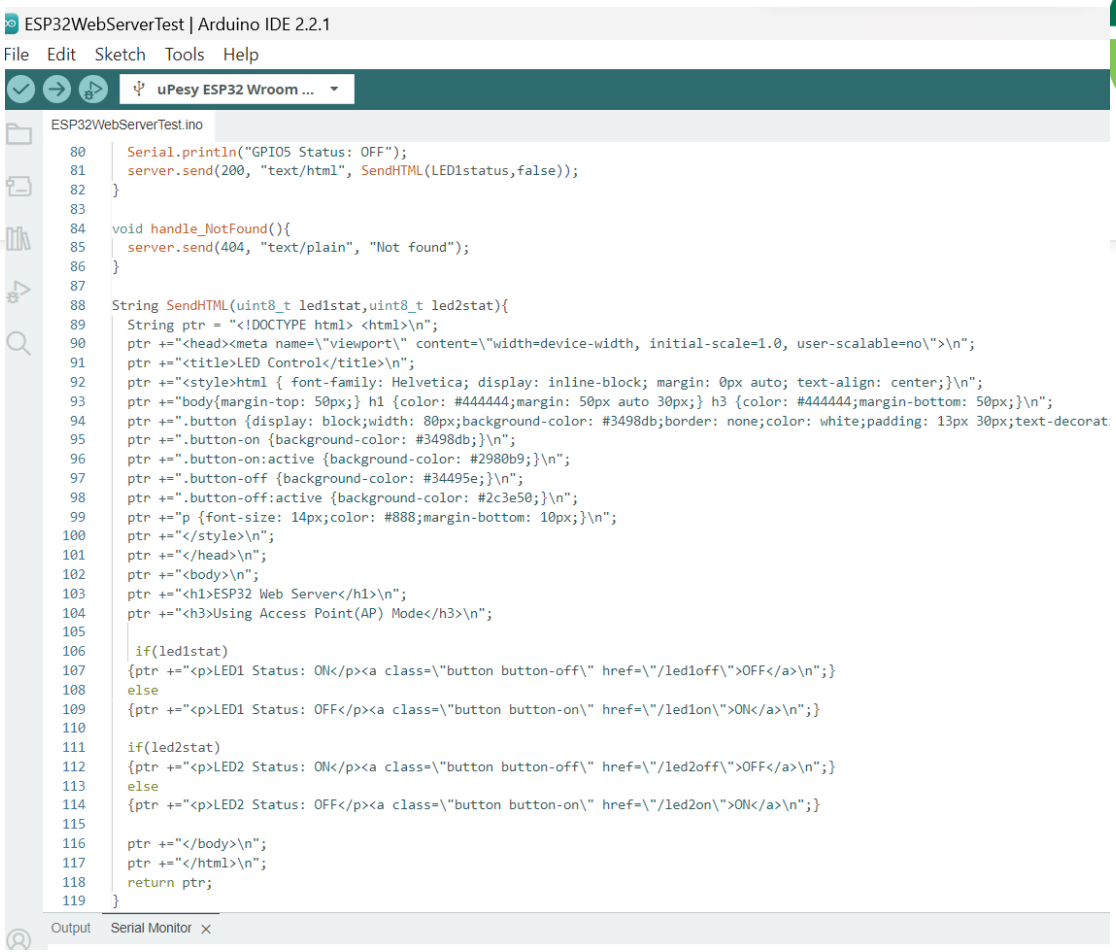
Exercise : Appliance Control Over the Internet using HTTP Protocol



Exercise : Appliance Control Over the Internet using HTTP Protocol



Exercise :




```
  Serial.println("GPIO5 Status: OFF");
  server.send(200, "text/html", SendHTML(LED1status,false));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(uint8_t led1stat,uint8_t led2stat){
  String ptr = "<!DOCTYPE html> <html>\n";
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
  ptr +="<title>LED Control</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";
  ptr +=".button {display: block;width: 80px;background-color: #3498db;border: none;color: white;padding: 13px 30px;text-decorat
  ptr +=".button-on {background-color: #3498db;}\n";
  ptr +=".button-on:active {background-color: #2980b9;}\n";
  ptr +=".button-off {background-color: #34495e;}\n";
  ptr +=".button-off:active {background-color: #2c3e50;}\n";
  ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
  ptr +="</style>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<h1>ESP32 Web Server</h1>\n";
  ptr +="<h3>Using Access Point(AP) Mode</h3>\n";

   if(led1stat)
  {ptr +="<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n";}
  else
  {ptr +="<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n";}

  if(led2stat)
  {ptr +="<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n";}
  else
  {ptr +="<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n";}

  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Lets see this in the video

https://drive.google.com/file/d/1crNYX_04rHnv0blKowVJHKgMuZQ-VNmO/view?usp=drive_link

We discusses 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)
- 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.

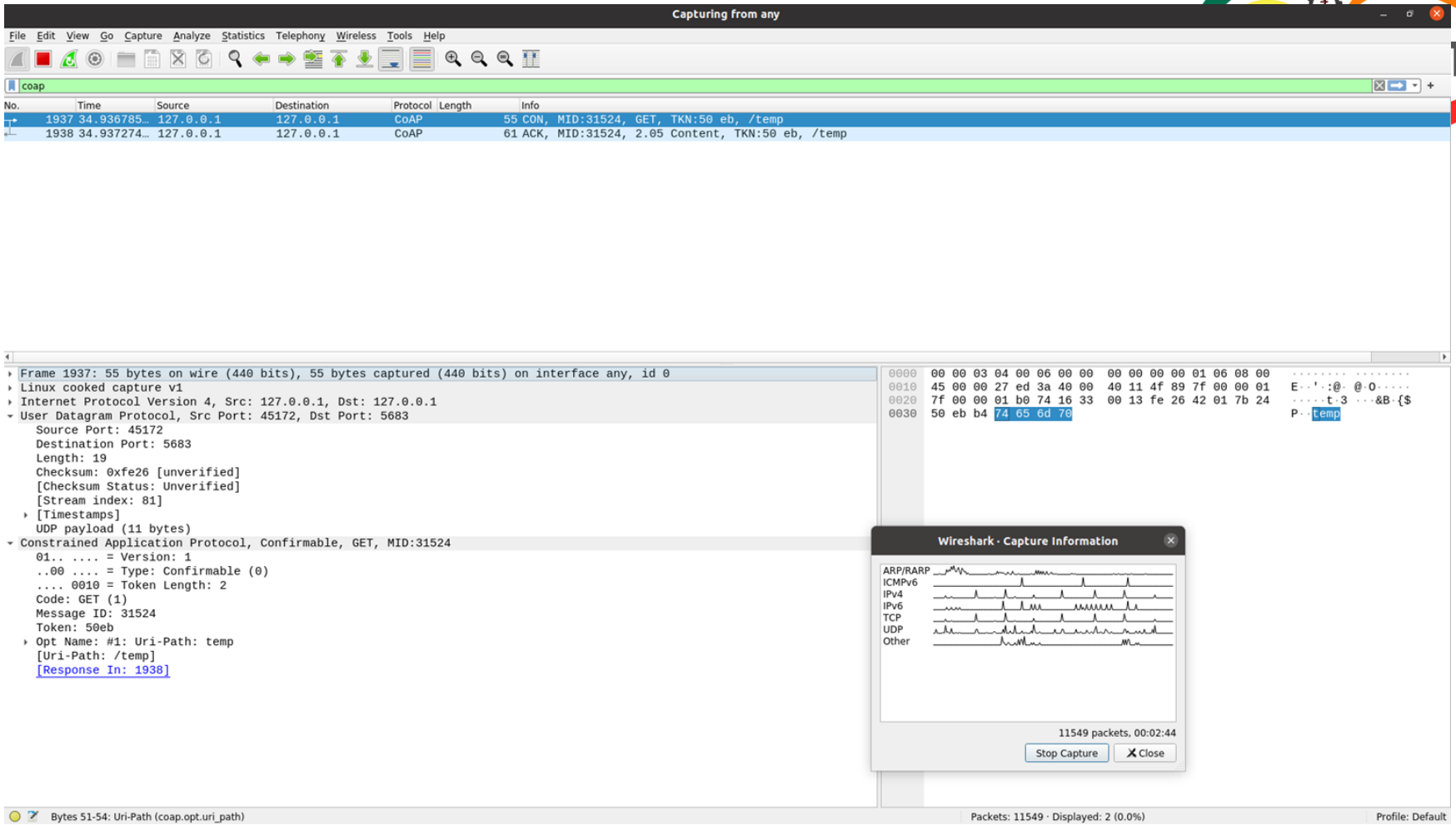
- Start wireshark before the next step
- Similarly after running server, client can be executed
- In new terminal/powershell window
- Various commands:
  - **python3 coapclient.py -o GET -p coap://127.0.0.1:5683/temp**
  - **python3 coapclient.py -o DISCOVER -p coap://127.0.0.1:5683/**
- more commands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

Using WireShark for CoAP

Starting CoAP Client

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

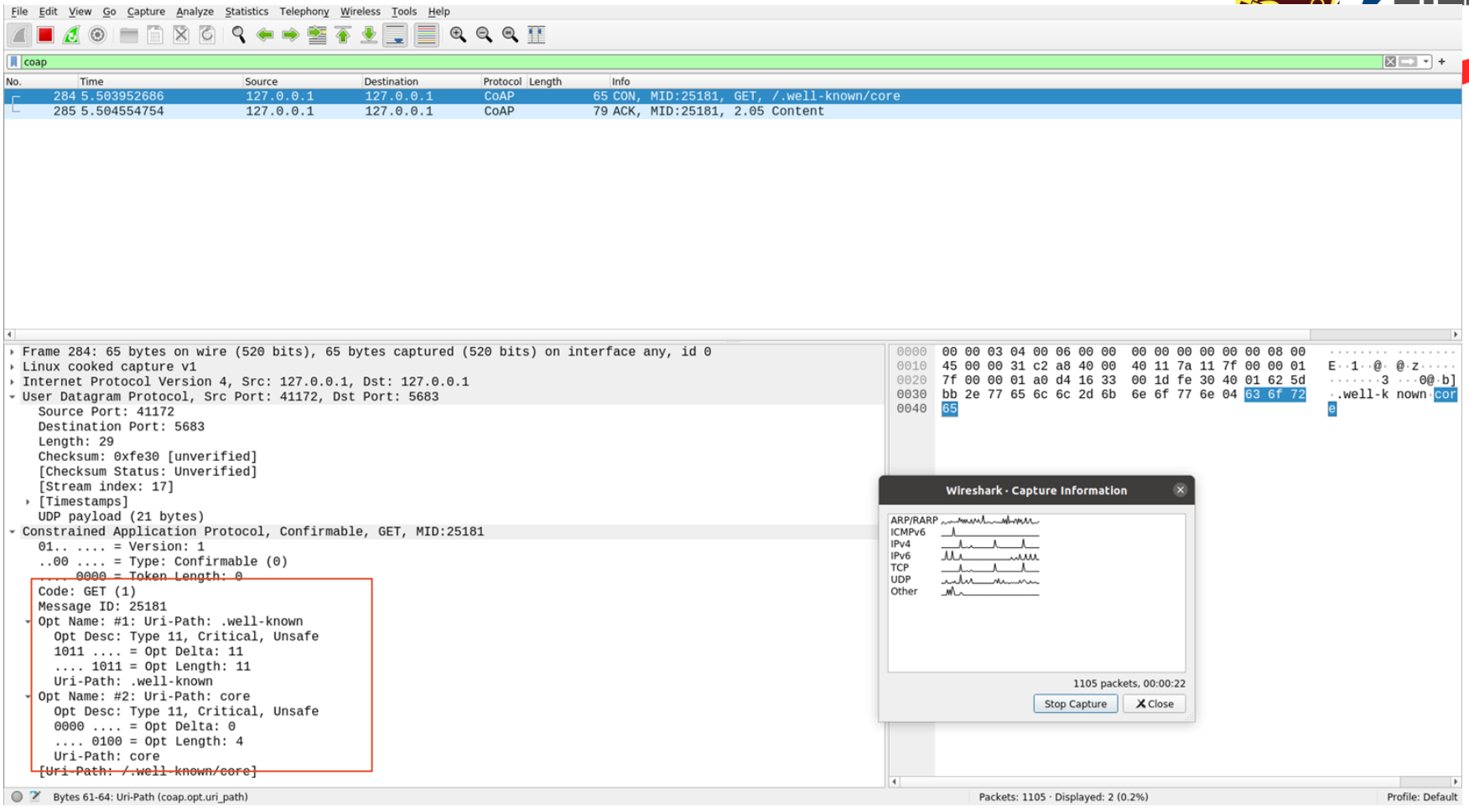

Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
coap
No. Time Source Destination Protocol Length Info
1937 34.936785… 127.0.0.1 127.0.0.1 CoAP 55 CON, MID:31524, GET, TKN:50 eb, /temp
1938 34.937274… 127.0.0.1 127.0.0.1 CoAP 61 ACK, MID:31524, 2.05 Content, 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
0020 7f 00 00 01 b0 74 16 33 00 13 fe 26 42 01 7b 24
0030 50 eb b4 74 65 6d 70
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,
```

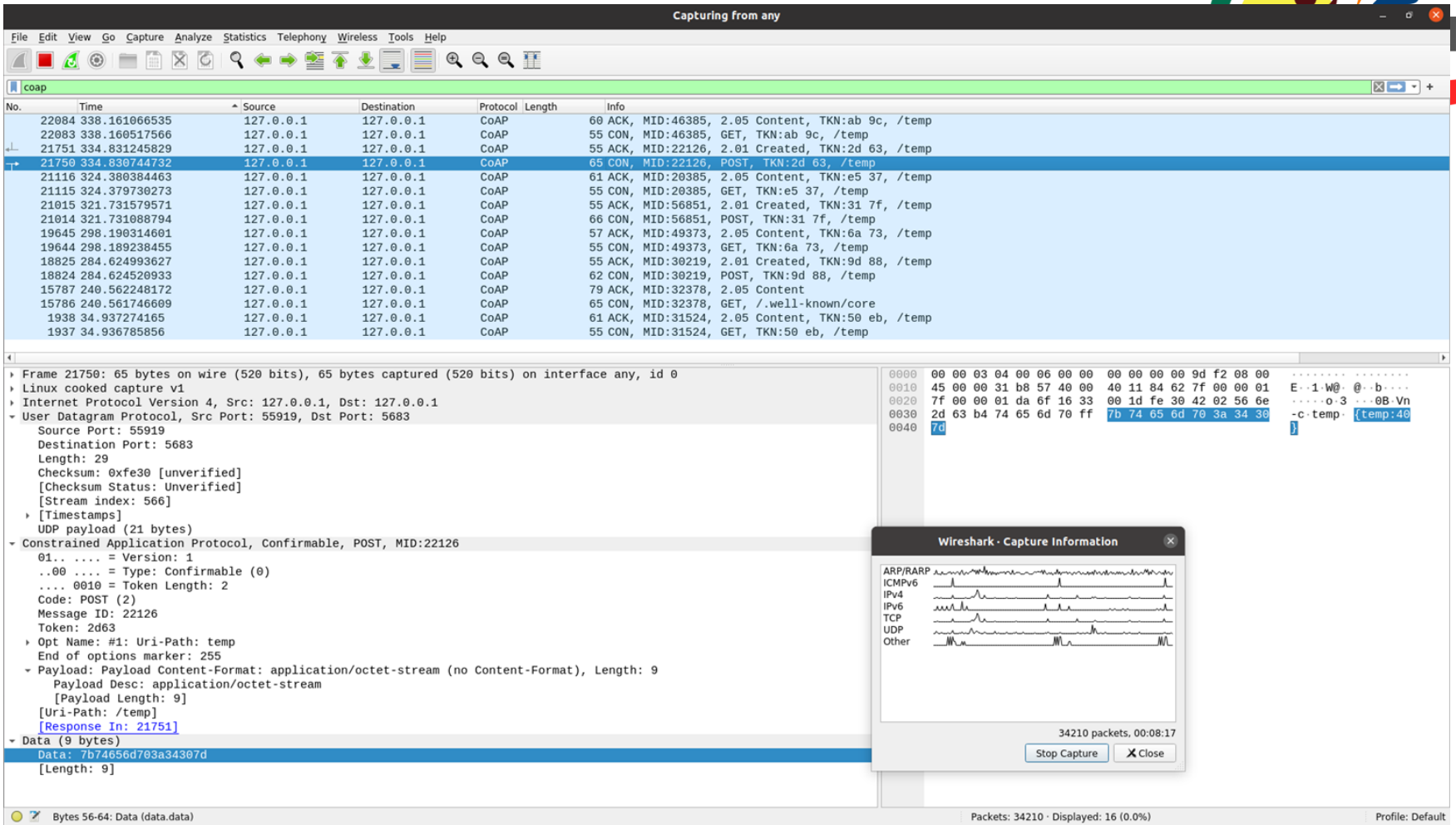
Discover request

```
(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}
```

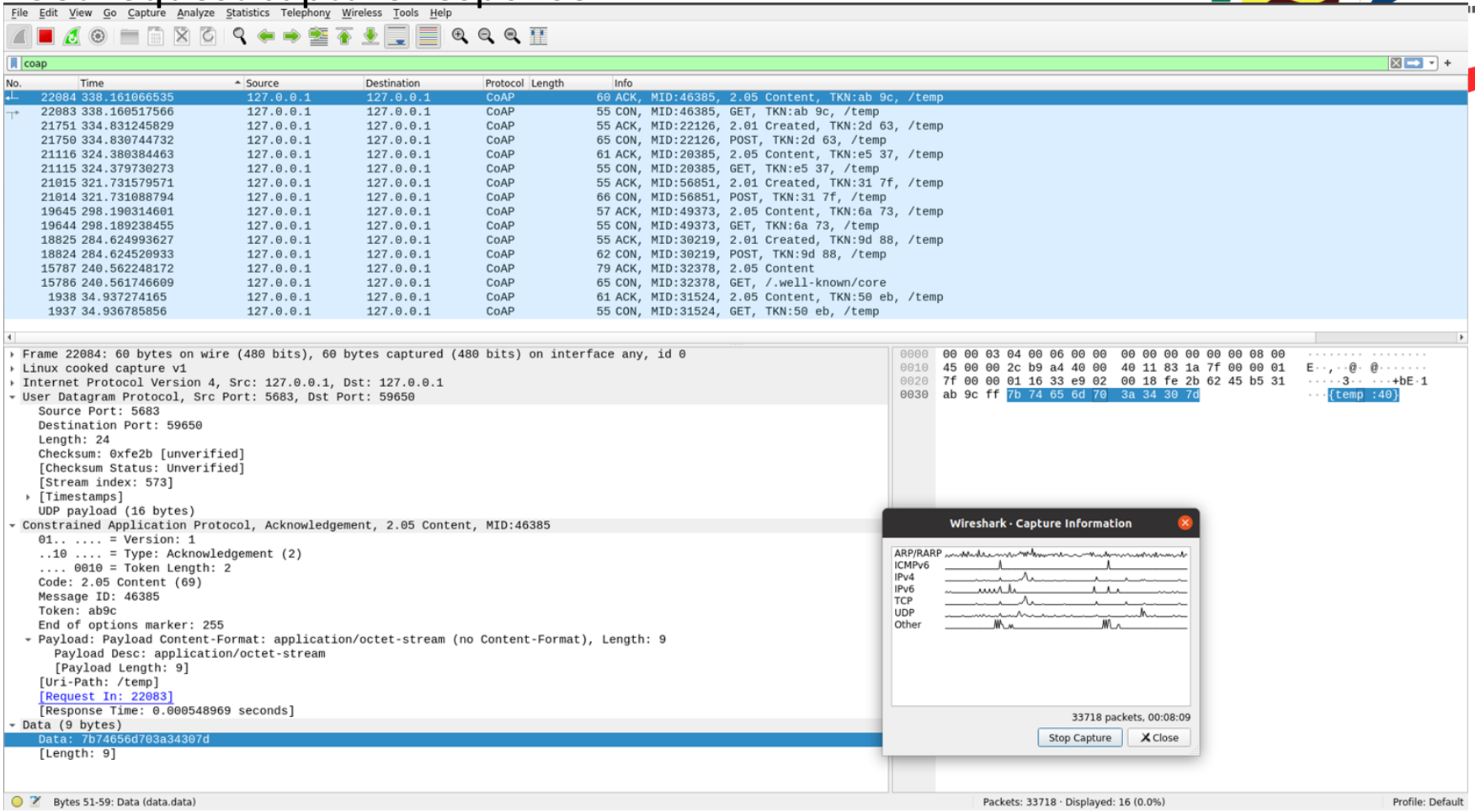
For Discover request



Post request capture response



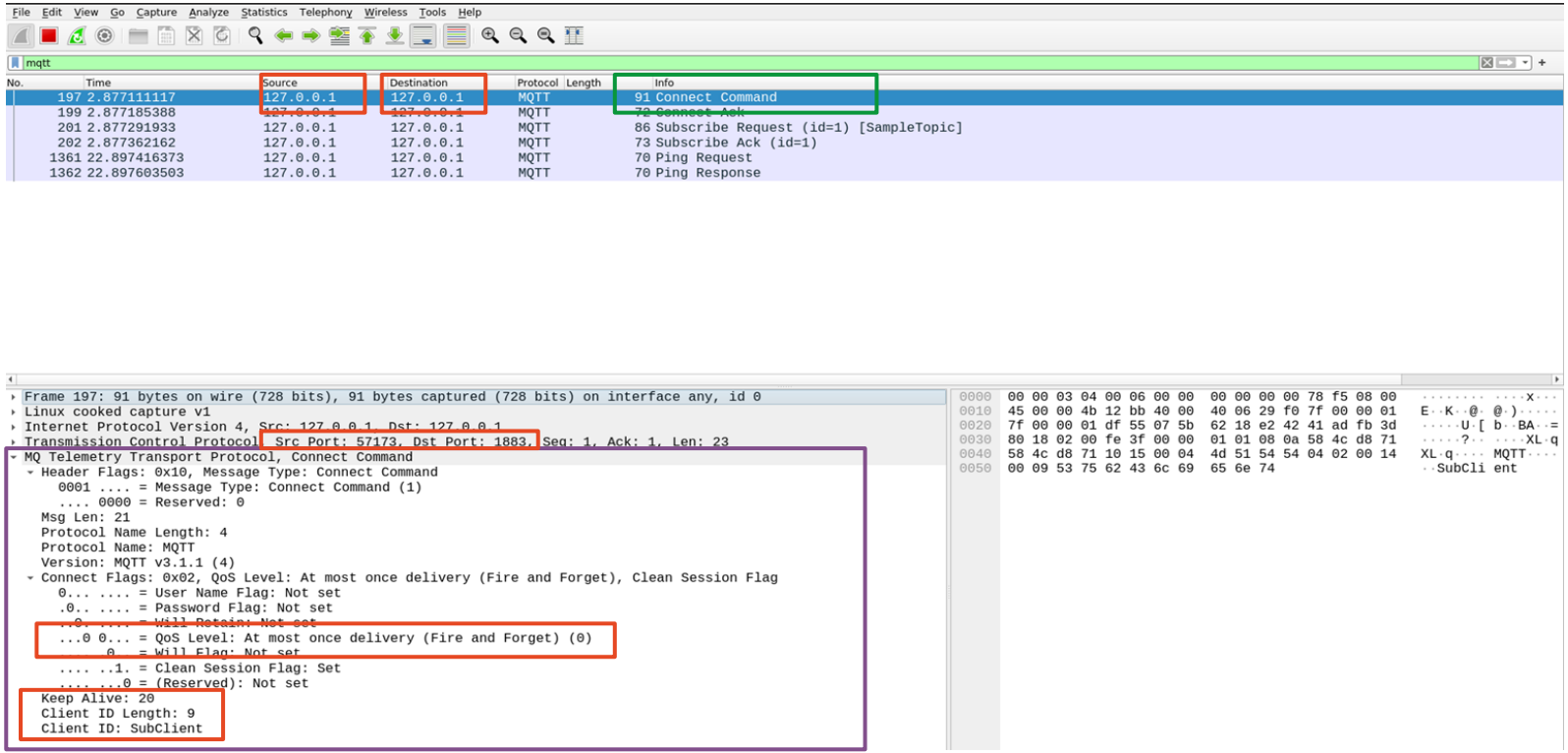
Get request capture response

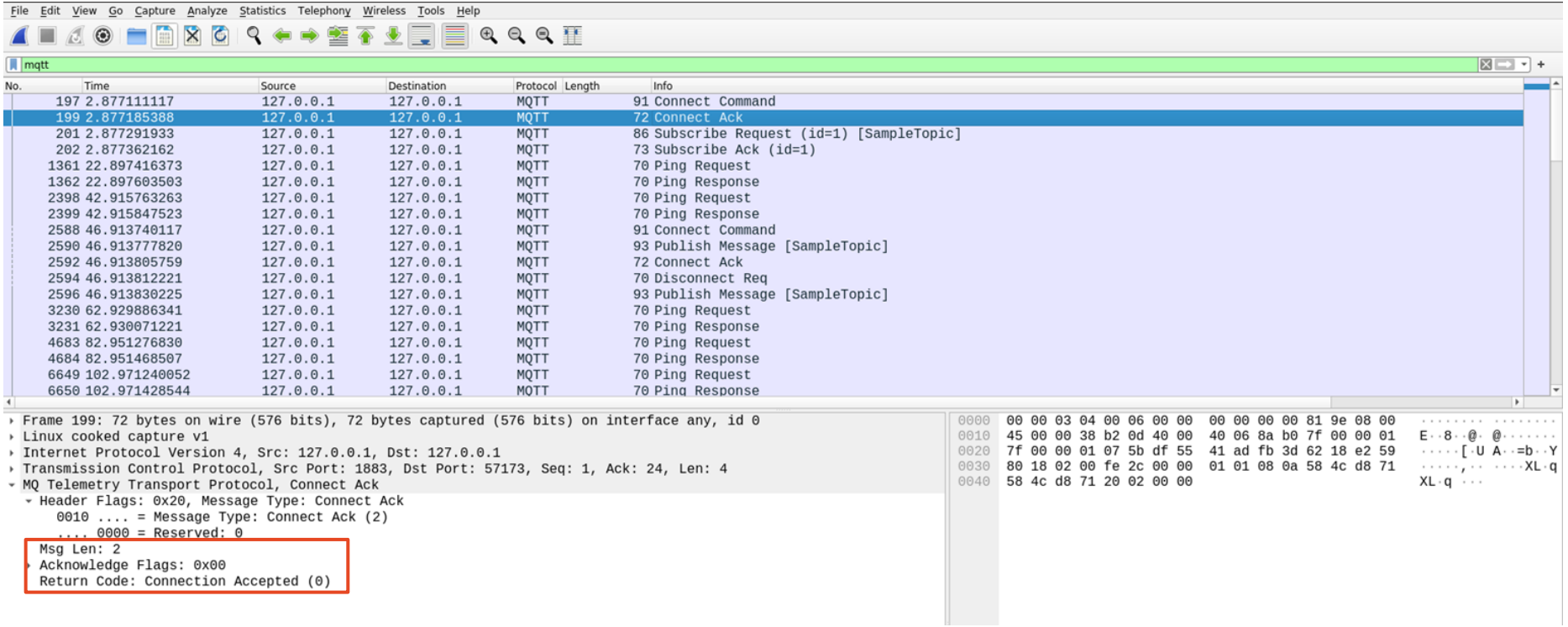


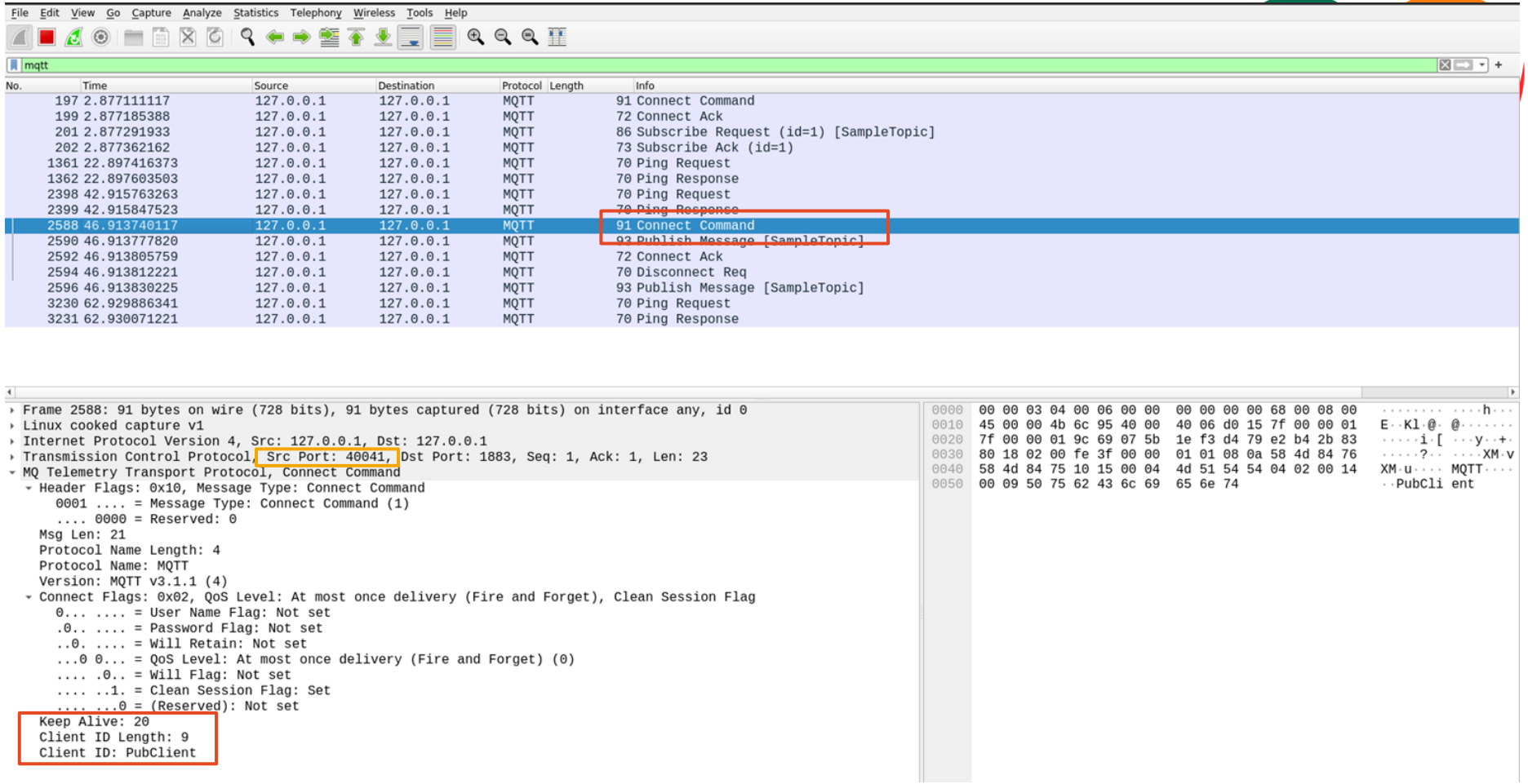
Requirements MQTT:

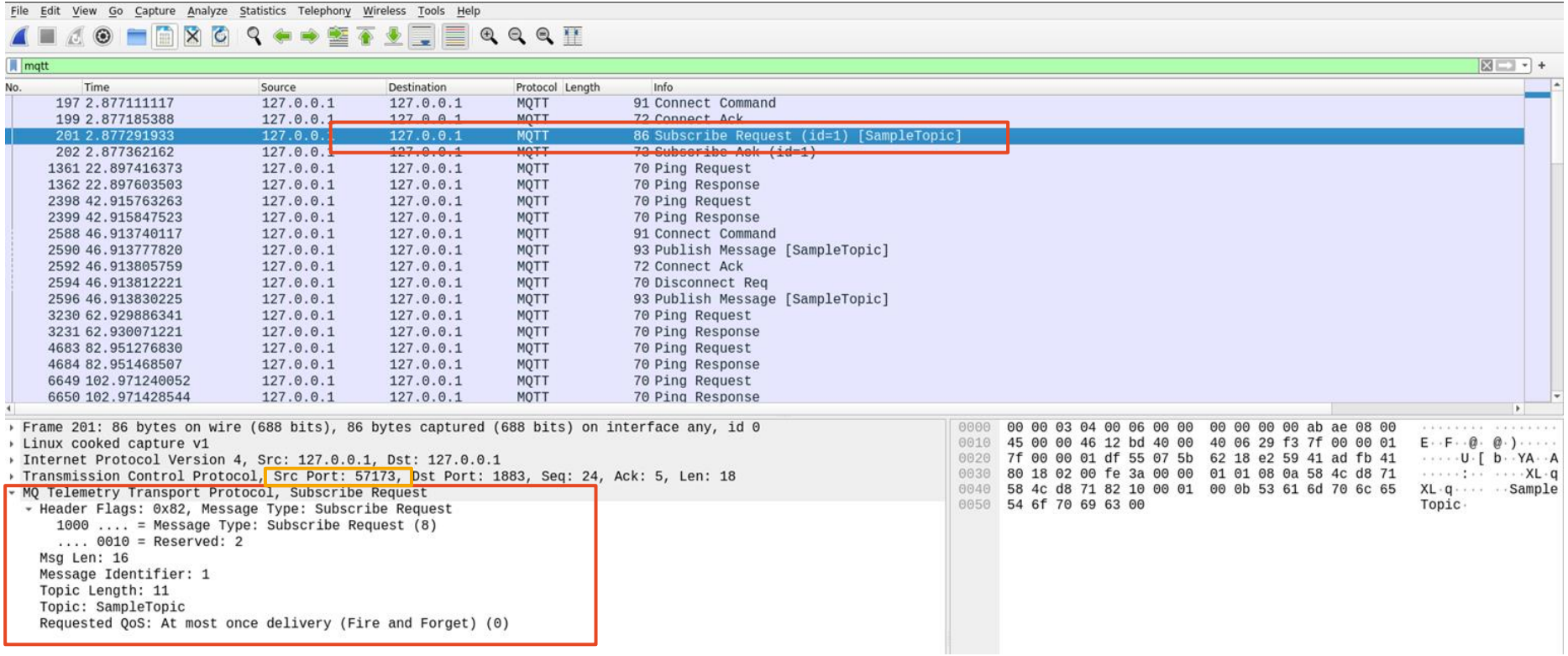
- Python (Optional-IDE) (prefer Miniconda link, https://shorturl.at/sBHRY)
- For MQTT Implementation
  - Download the files here or https://shorturl.at/LSV36
  - Extract the files in a folder
  - Download and Install mosquitto broker from https://mosquitto.org/download/
  - Run mosquitto -v in powershell/terminal 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




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
0020 7f 00 00 01 07 5b df 55 41 ad fb 3d 62 18 e2 59
0030 80 18 02 00 fe 2c 00 00 01 01 08 0a 58 4c d8 71
0040 58 4c d8 71 20 02 00 00


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
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
0010 45 00 00 4b 6c 95 40 00 40 06 d0 15 7f 00 00 01
0020 7f 00 00 01 9c 69 07 5b 1e f3 d4 79 e2 b4 2b 83
0030 80 18 02 00 fe 3f 00 00 01 01 08 0a 58 4d 84 76
0040 58 4d 84 75 10 15 00 04 4d 51 54 54 04 02 00 14
0050 00 09 50 75 62 43 6c 69 65 6e 74


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 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
0020 7f 00 00 01 df 55 07 5b 62 18 e2 59 41 ad fb 41
0030 80 18 02 00 fe 3a 00 00 01 01 08 0a 58 4c d8 71
0040 58 4c d8 71 82 10 00 01 00 0b 53 61 6d 70 6c 65
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/17Ic9R9EFZ53DjUsIVGaoczl7VbzlYK13?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 !