

IoT Device Classification from Network traffic log using Machine Learning

Dr Gaurav Singal

Netaji Subhas University of Technology, Delhi

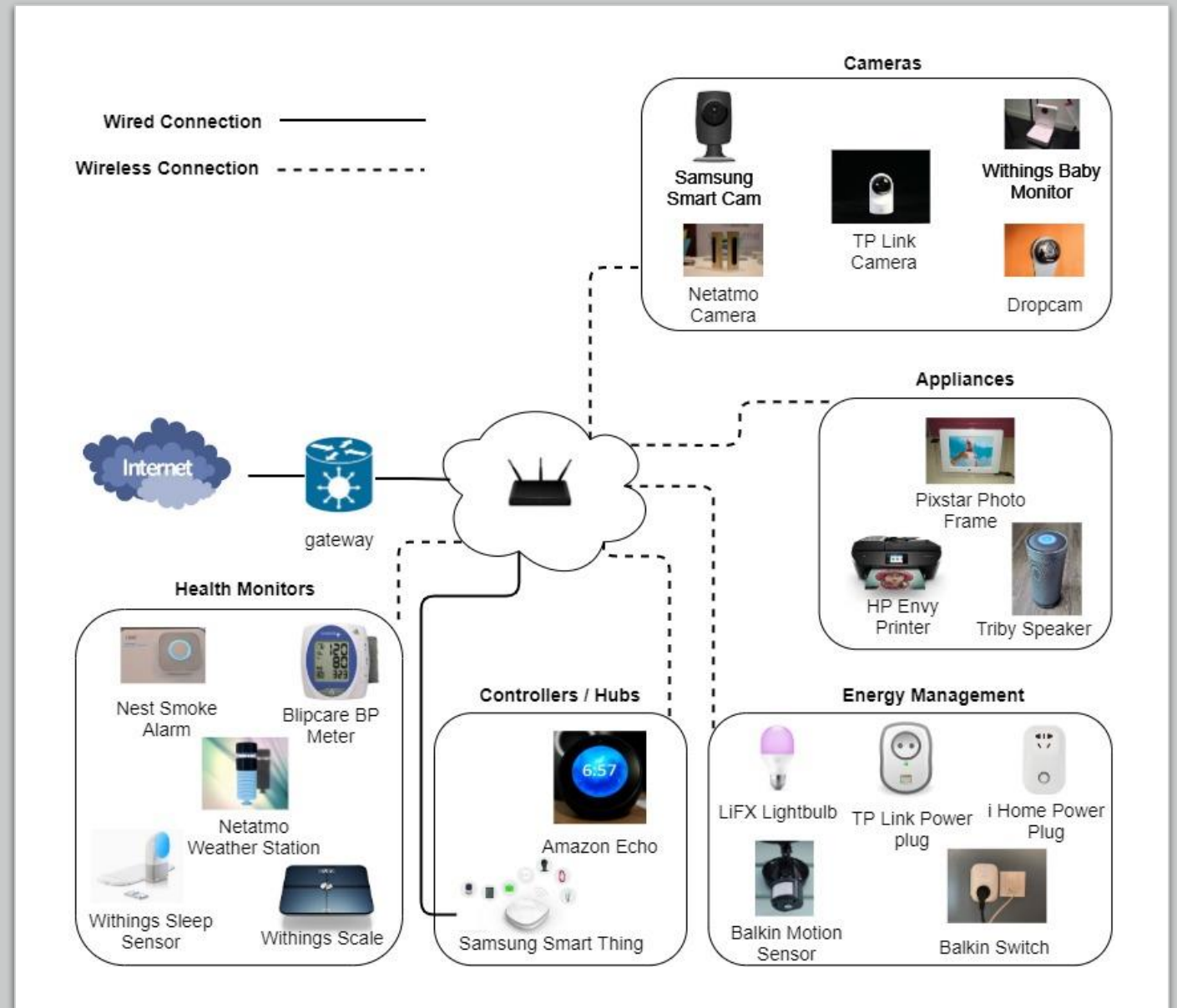


IoT Security

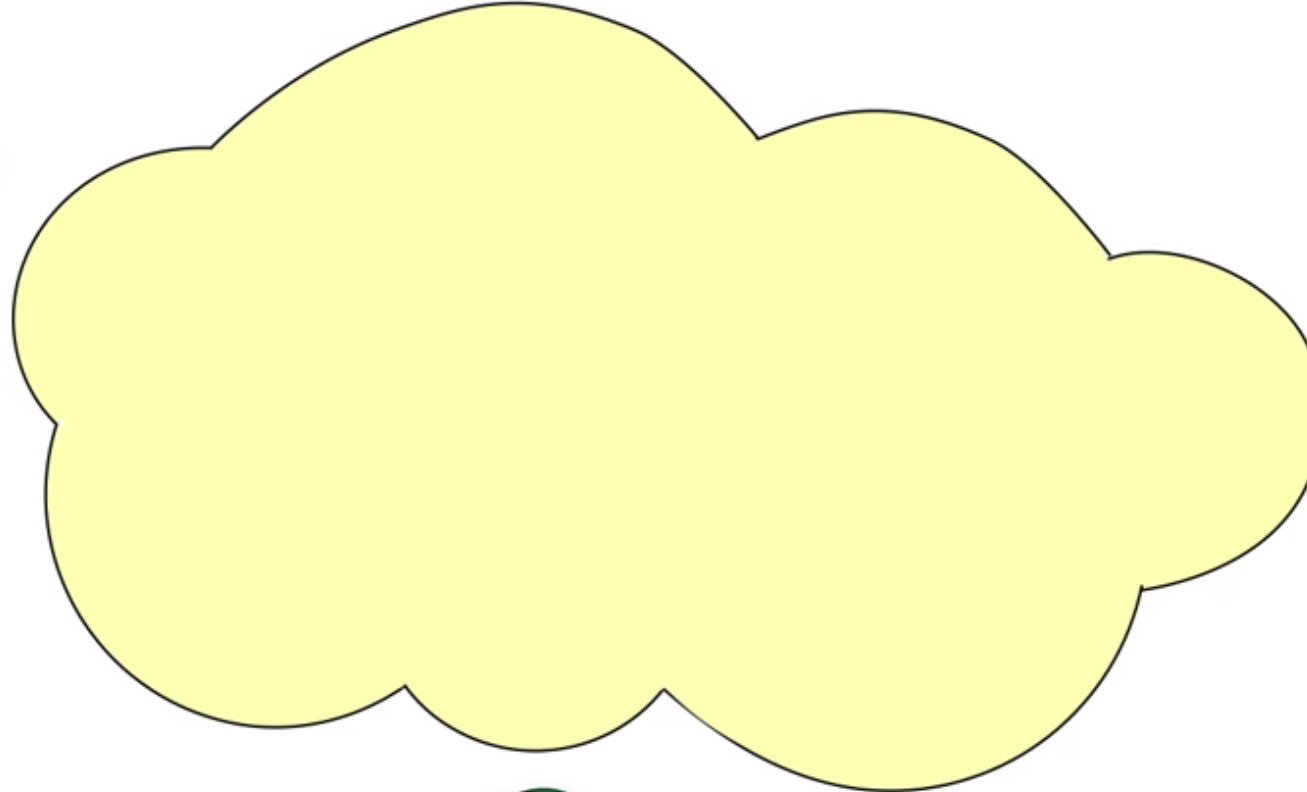


IoT network traffic classification

- Mechanism to **categorizes** the embedded **devices** connected to the internet and security **attacks** in the network.
- Beneficial to ensuring security, reliability, quality of services (QoS) and complete working of IoT devices.



Traffic analysis



Motivation

- Easy to hack, can easy to compromise and become a part of botnet[3,4].
- Need to classify the IoT devices[5].

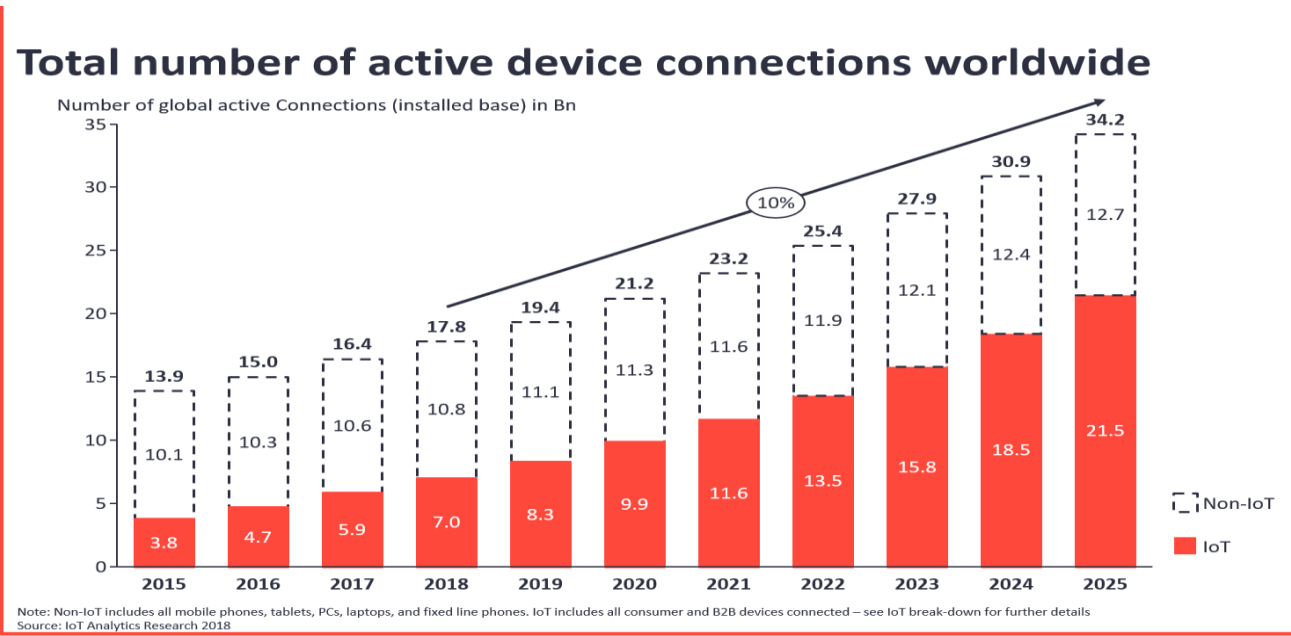


Fig: Number of IoT devices connected worldwide

Figure courtesy : <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>

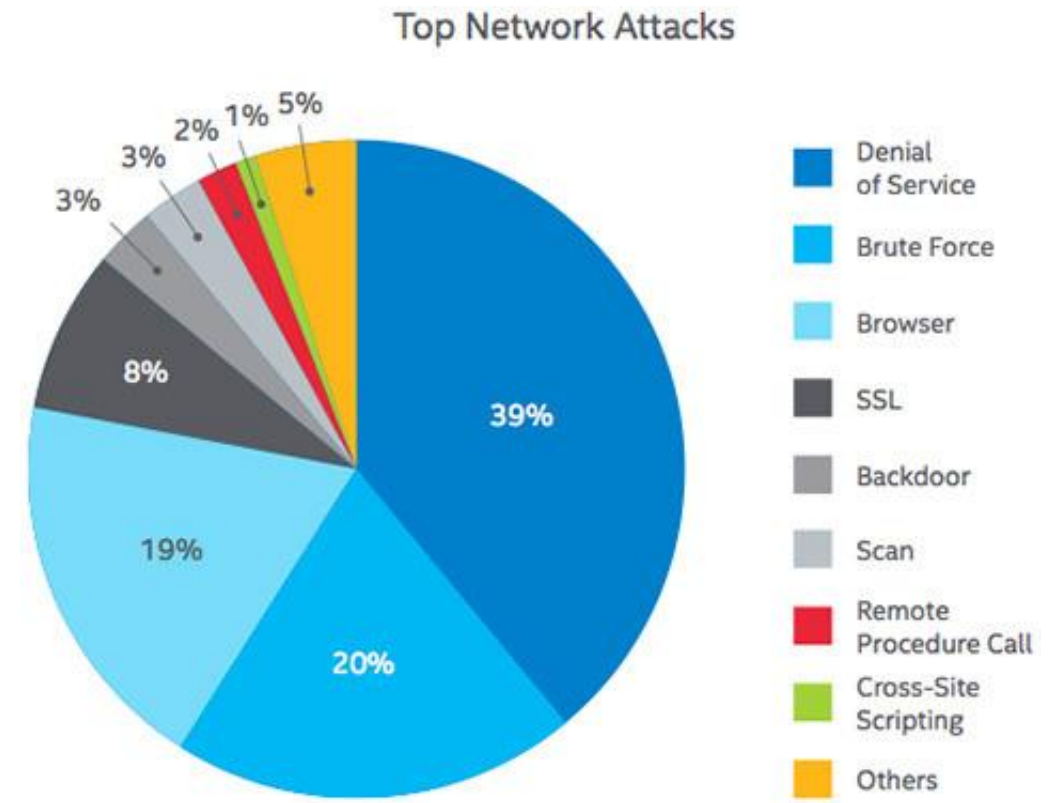


Fig.: IoT Attacks Statistics

Figure courtesy : <https://www.cisecurity.org/blog/top-10-malware-july-2020/>

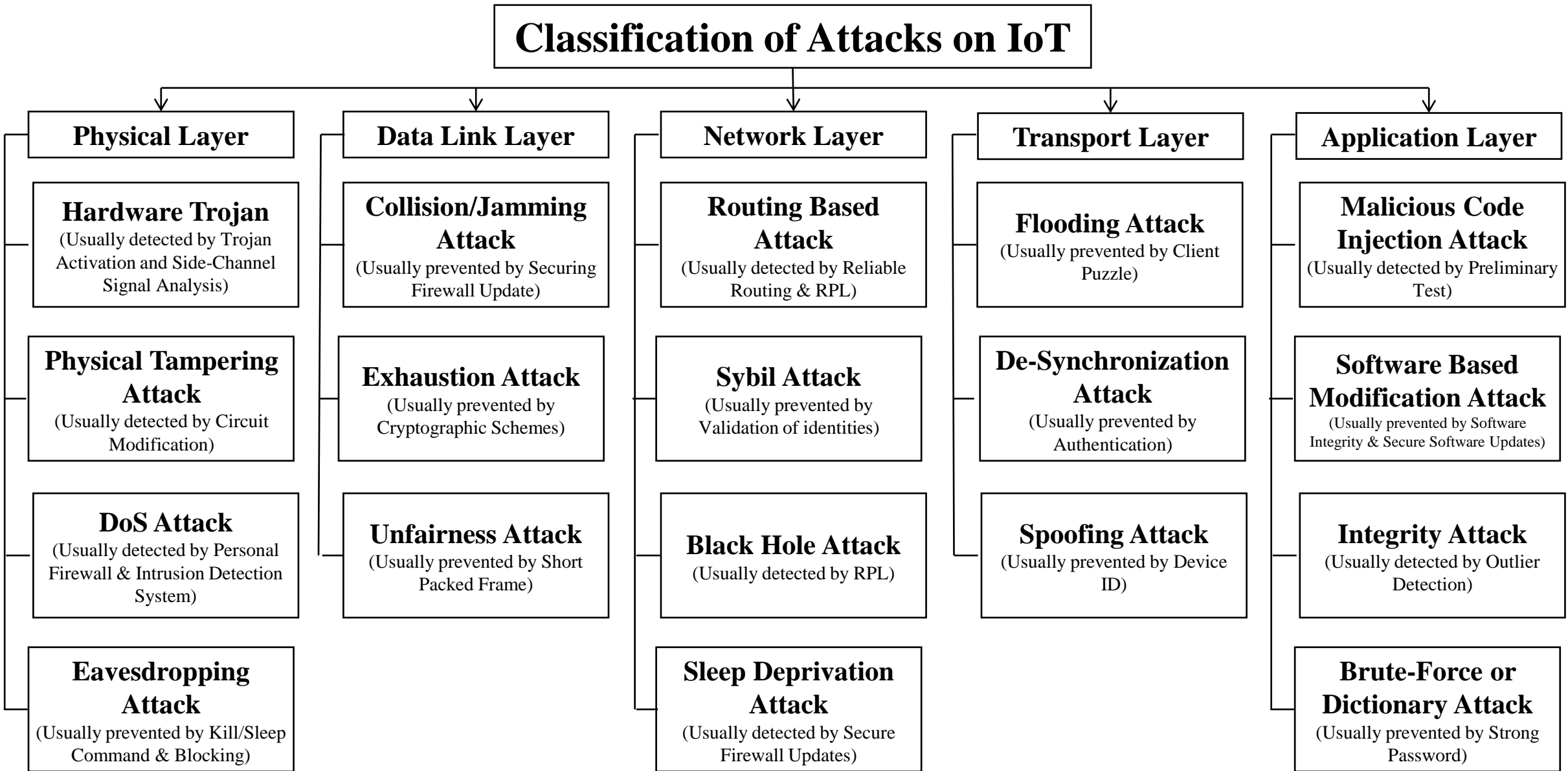
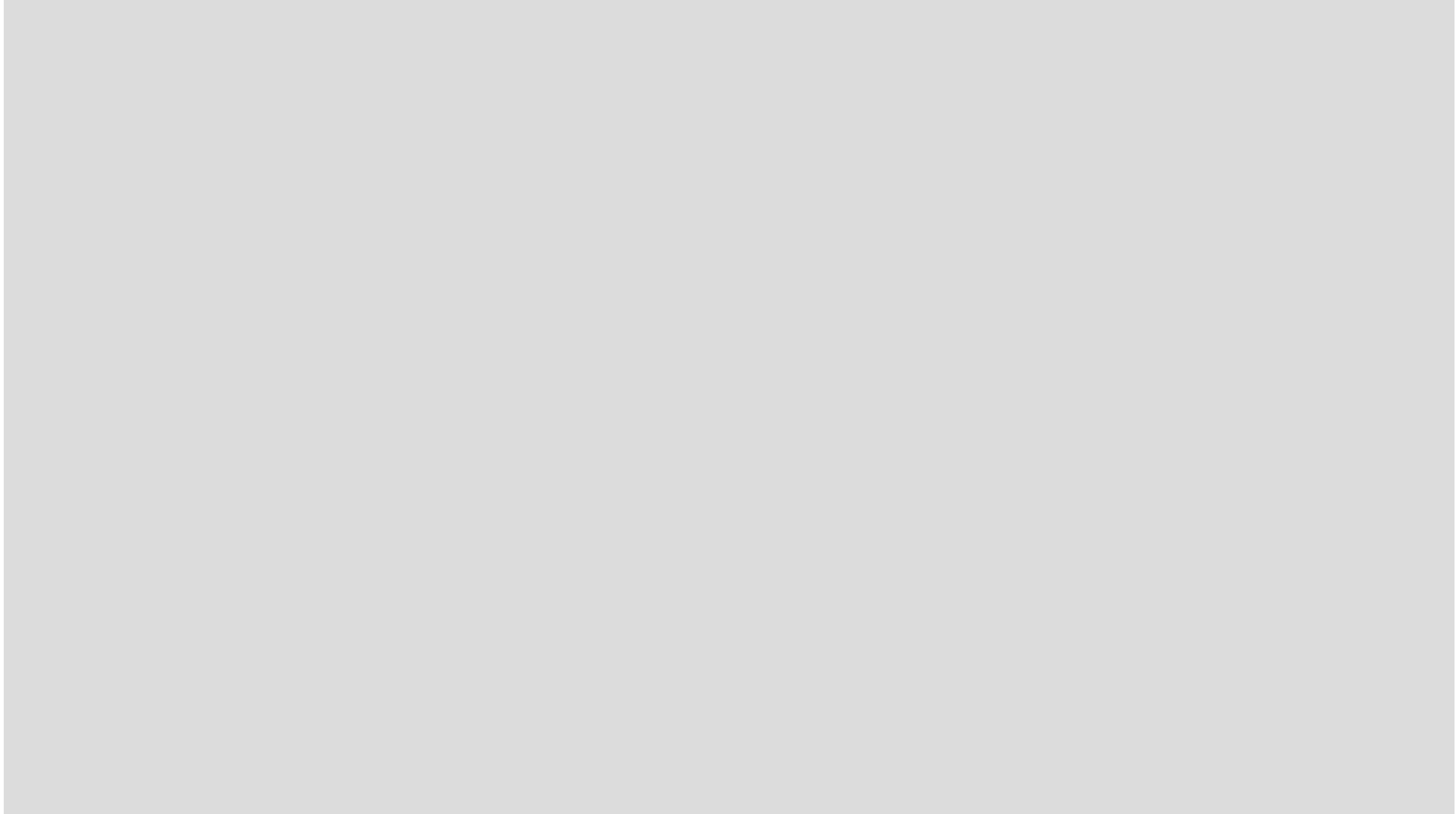


Fig.: Layer-wise IoT Attacks [6] [7] [8]

How DDoS is working?



Issues in IoT Networks Traffic classification

- **Variety** and **limited** number of IoT devices for classification.
- **Overlapping** instances problem increases as traffic increases from IoT devices.
- Issues in IoT devices classification due to periodic **updates**.
- Limited number of large **datasets** available publicly.
- **User** security and **privacy** Issues by data breaching.
- **Unbalanced** traffic from IoT devices (biased).
- **Unknown** (new) device and attacks in IoT network traffic.

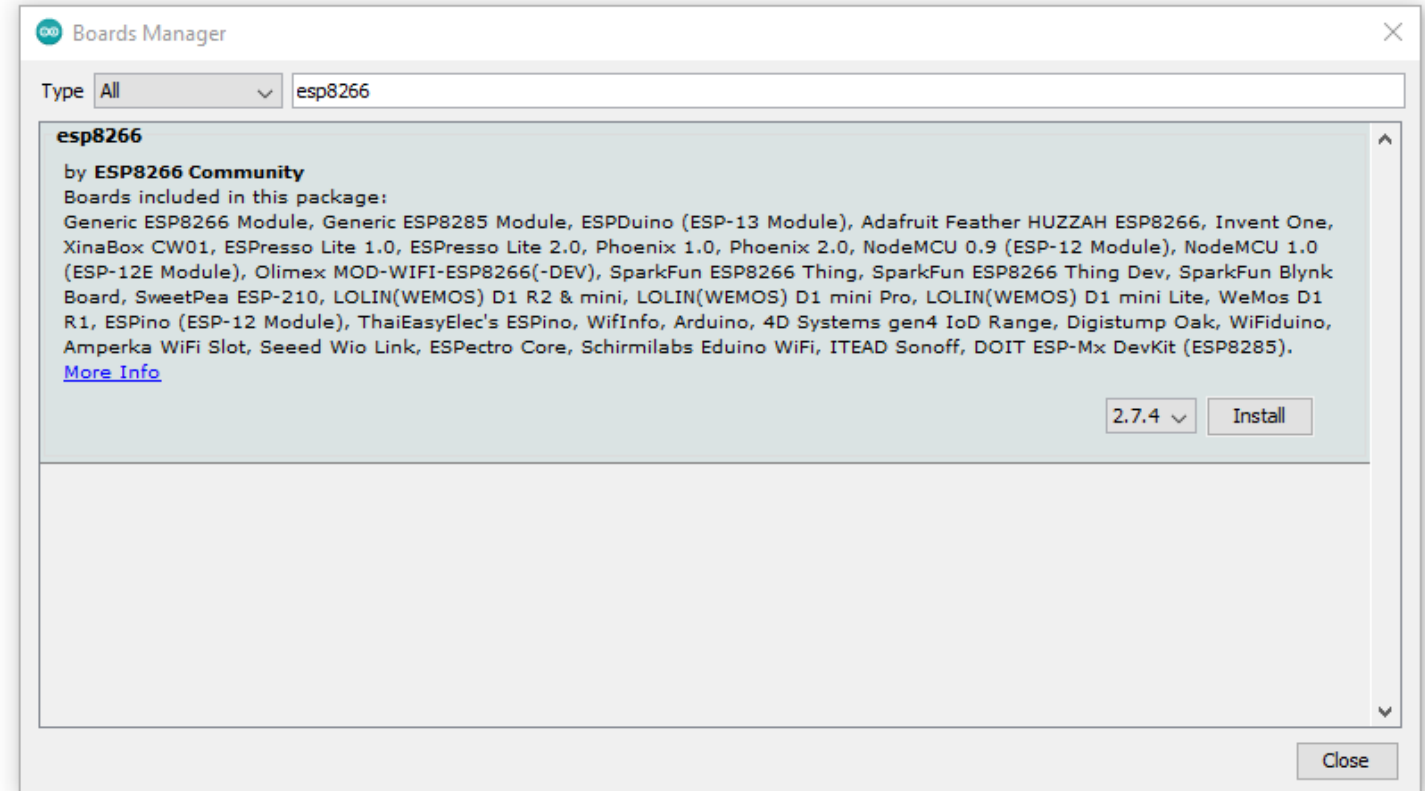


Capturing Packets through MQTT Protocol

Generating Dataset

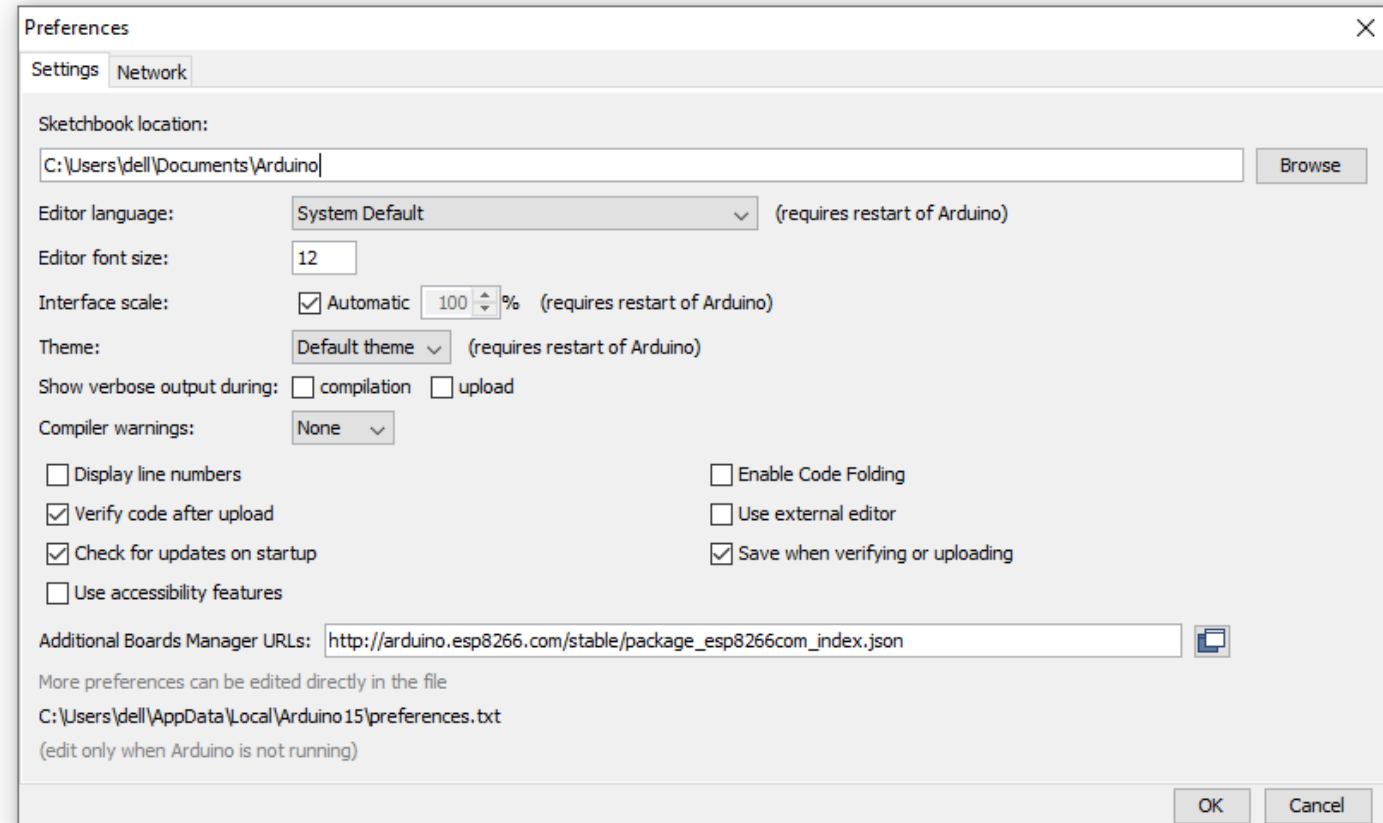
Setup Installation

- Install Arduino IDE :
- Go to **File --> Preference -->** paste the URL in additional board manager URL -->

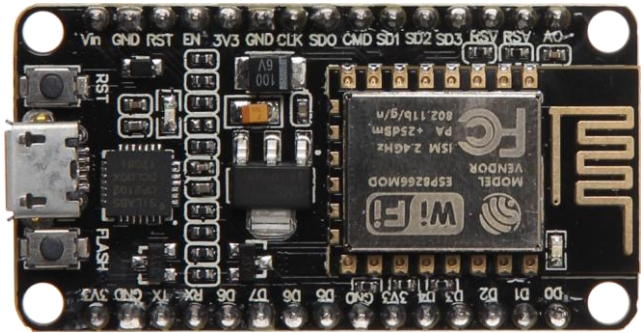


Continue

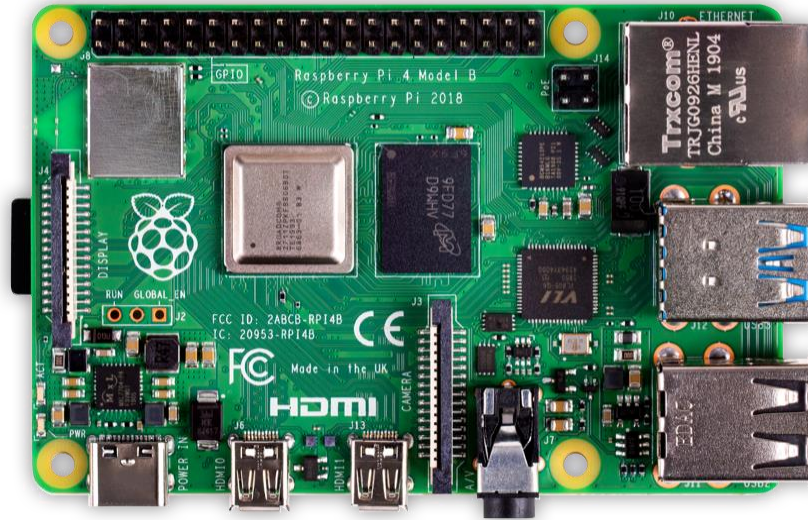
- Go to **Tools --> Board --> Board Manager --> Search for Esp8266** and install library
- **Go to Sketch --> Include Library --> Manage Library --> Search MQTT** and download **Adafruit MQTT Library, EspMQTTClient**
- Install Wireshark:
 - `sudo apt-get update`
 - `sudo apt-get upgrade`
 - `sudo apt-get install wireshark`



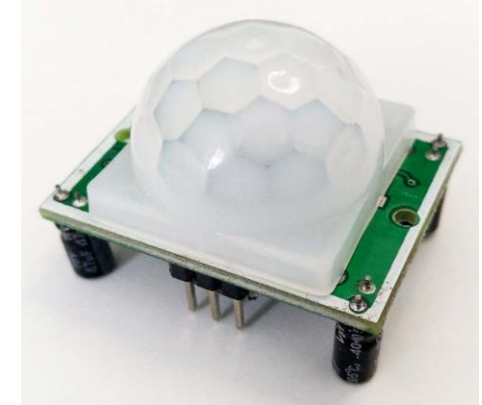
Devices Used



NodeMCU (ESP8266)



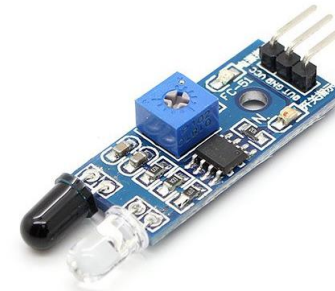
Raspberry Pi 4



PIR Sensor



Ultrasonic Sensor

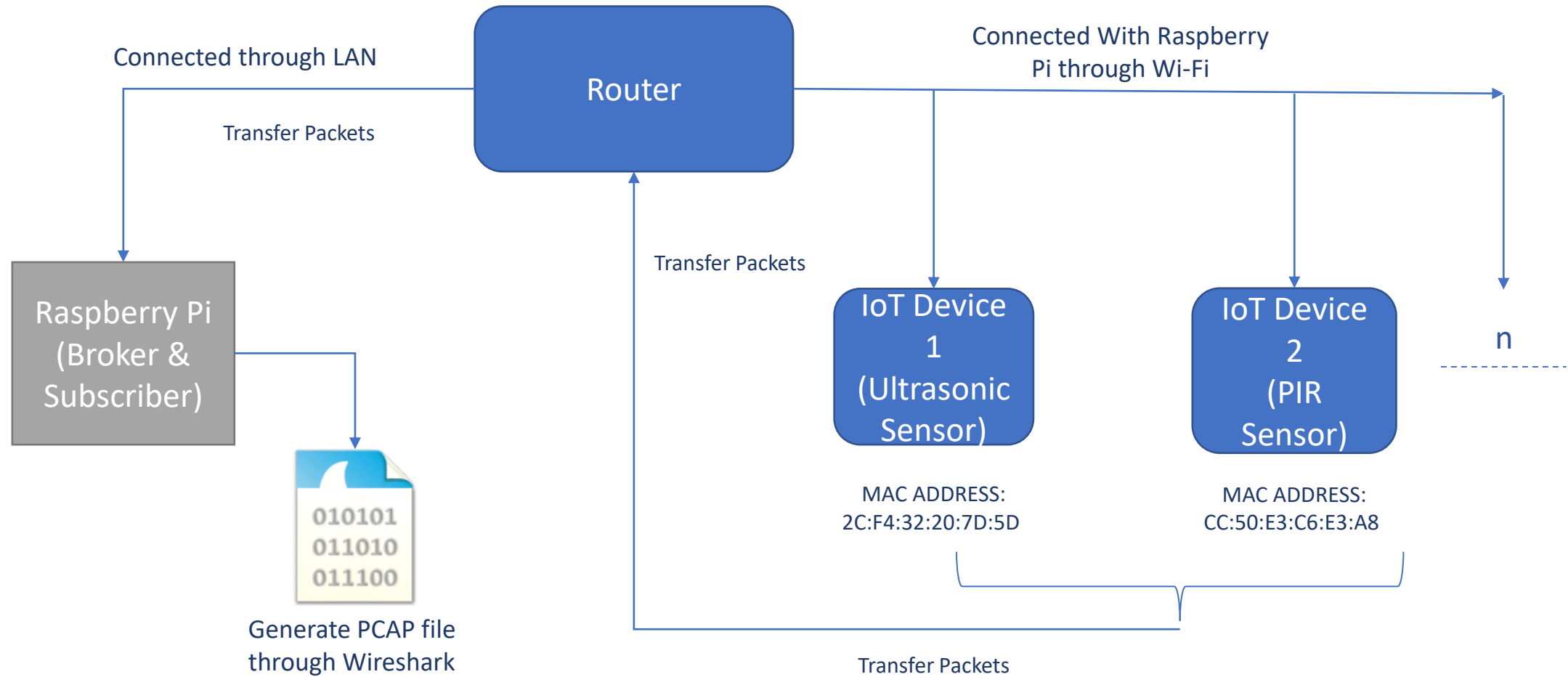


IR Sensor

About MQTT

- Message Queue Telemetry Transport Protocol
- It has “Publish/Subscriber architecture. Device can publish any topics and can also subscribe for any updates
- It runs over TCP
- Message size is Small
- MQTT session divided into four stages:
 - Connection
 - Authentication
 - Communication
 - Termination
- It is many-to-many communication protocol for passing messages between multiple clients through a central broker.
- Message format is binary with 2 Byte header

Workflow

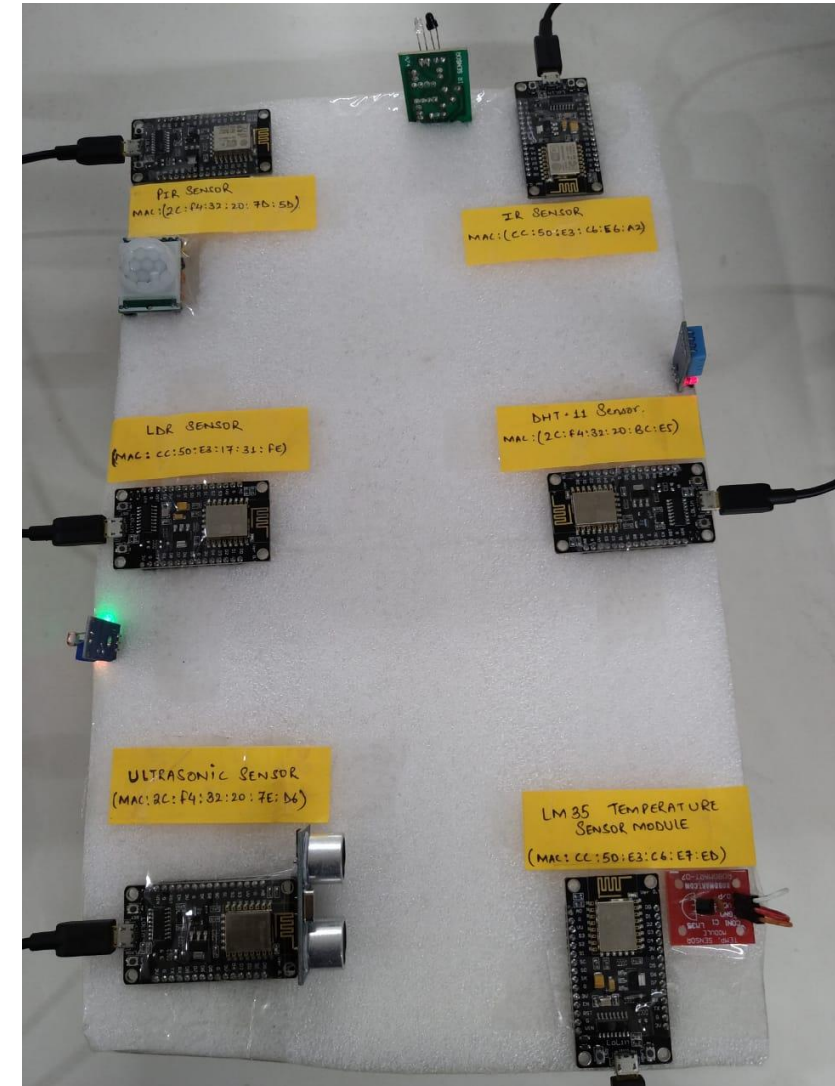
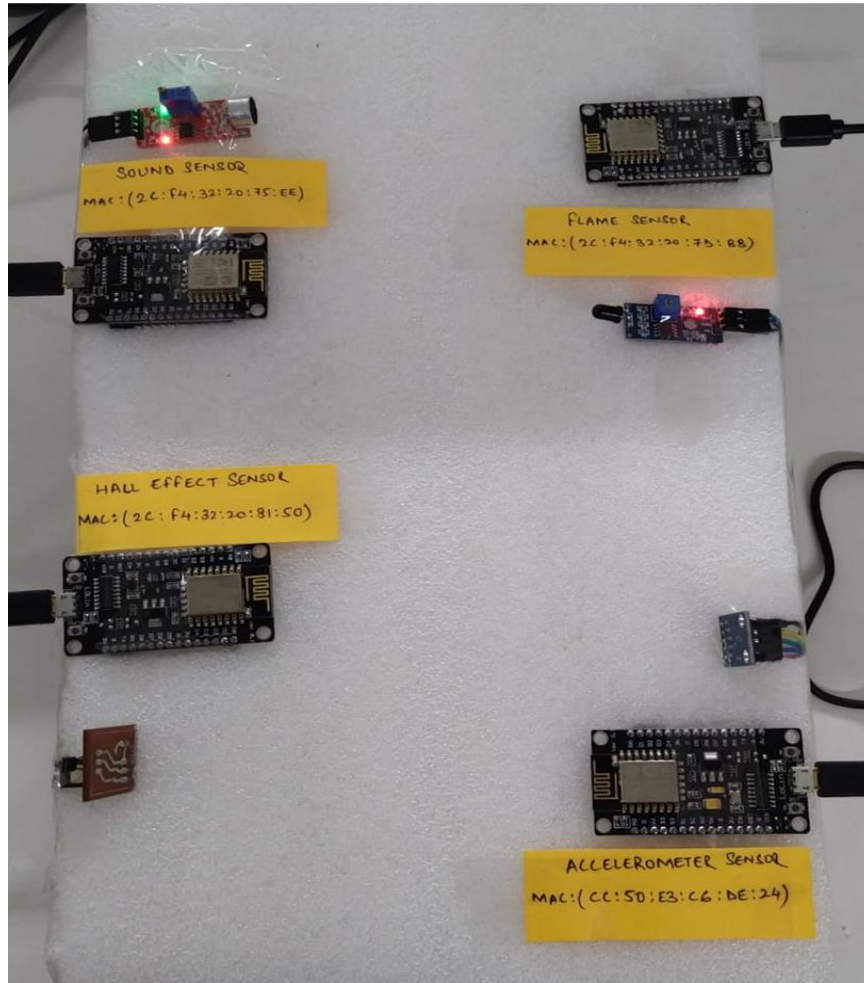


Description

MQTT Broker Setup

- Install MQTT broker
 - `sudo apt-get install mosquitto`
- Install command line clients in case for debugging
 - `sudo apt-get install mosquitto-clients -y`
- Open the Mosquitto MQTT broker configuration
 - `sudo nano /etc/mosquitto/mosquitto.conf`
- Create new user with username and password
 - `sudo mosquitto_passwd -c /etc/mosquitto/pwfile username`
 - `sudo mosquitto_passwd -c /etc/mosquitto/pwfile password`
- See current status of MQTT broker
 - `sudo systemctl status mosquitto`
- Stop Mosquitto:
 - `sudo systemctl stop mosquitto`

Experiment Setup



S.NO.	IOT DEVICE NAME	MAC ADDRESS	PROTOCOLS	APPLICATION AREA
1	Ultrasonic Sensor 1	2C:F4:32:20:7E:D6	MQTT	Motion Sensor or Distance Sensor
2	PIR Sensor	2C:F4:32:20:7D:5D	MQTT	Smart HVAC or Smart Lighting
3	IR Sensor	CC:50:E3:C6:E6:A2	MQTT	Scan a room Prepare a Heat map and control the temperature
4	DHT11 Sensor	2C:F4:32:20:BC:E5	MQTT	Measure room temperature and Humidity and controlling fan
5	LDR Sensor	CC:50:E3:17:31:FE	MQTT	Street Lights, Light Intensity Meters, Burglar Alarm Circuits
6	Flame Sensor	2C:F4:32:20:7D:BB	MQTT	Gas, Heaters monitor, Flame quality monitor.
7	Tilt Sensor	CC:50:E3:C6:0E:32	MQTT	Garage door control, smart from of mobile devices
8	Sound Sensor	2C:F4:32:20:75:EE	MQTT	Audio Amplifier, smartphones, sound level recognition
9	Moisture Sensor	2C:F4:32:20:BC:2A	MQTT	Gardening
10	Vibration Sensor	2C:F4:32:20:BE:A4	MQTT	HVAC
11	Smoke Sensor	CC:50:E3:C6:DA:75	MQTT	Fire Alarm
12	Rain Sensor	2C:F4:32:20:BB:50	MQTT	Used in car rain sensing wiper
13	Hall Effect Sensor	2C:F4:32:20:81:50	MQTT	Position sensing and fluid monitoring
14	LM35 Temperature Sensor	CC:50:E3:C6:E7:ED	MQTT	Battery monitoring in car
15	Accelerometer Sensor	CC:50:E3:C6:DE:24	MQTT	Opening and closing doors
16	Pulse Sensor	2C:F4:32:20:BD:EA	MQTT	Health Monitoring
17	GPS Module	F4:CF:A2:F5:0A:BD	MQTT	Smart Phones, Car positioning monitoring
18	TCRT5000	8C:AA:B5:59:91:55	MQTT	Object detection
19	Laser Sensor	8C:AA:B5:59:8E:FD	MQTT	Security and Surveillance

S.NO.	IOT DEVICE NAME	MAC ADDRESS	PROTOCOLS	APPLICATION AREA
20	Real Time Clock Module Sensor	84:CC:A8:83:76:18	MQTT	Control the Object for a specific time
21	Gyroscope Sensor	f4:cf:a2:f5:14:80	HTTP	used for car navigation systems, electronic stability control systems fo vehicles, motion sensing for mobile games
22	Pressure Sensor	f4:cf:a2:f5:15:a6	HTTP	GPS modules, air pressure, water flow pressure, leak/moisture detection
23	Color Code Sensor	f4:cf:a2:f5:0e:0c	HTTP	detect the color of an object and send command to the smart lighting for same color detect the color of an object and tells the color code of it.
24	Air Quality Sensor (MQ135)	f4:cf:a2:f5:0c:b5	HTTP	Measuring the air quality
25	Alcohol Sensor (MQ3)	8c:aa:b5:59:8f:dc	HTTP	Detect the presence of alcohol
26	Load Cell Sensor	f4:cf:a2:f2:fc:69	HTTP	Used for weighing of an object, used in door opening and close easily

File Edit Tabs Help

```
1607334520: New client connected from 192.168.4.9 as 192.168.0.2 (c1, k15, u'raspberrry').
1607334520: Client 192.168.0.2 disconnected.
1607334521: New connection from 192.168.4.15 on port 1883.
1607334521: New client connected from 192.168.4.15 as 192.168.0.4 (c1, k15, u'raspberrry').
1607334521: Client 192.168.0.4 disconnected.
1607334521: New connection from 192.168.4.7 on port 1883.
1607334521: New client connected from 192.168.4.7 as 192.168.0.3 (c1, k15, u'raspberrry').
1607334522: Client 192.168.0.3 disconnected.
1607334523: New connection from 192.168.4.9 on port 1883.
1607334523: New client connected from 192.168.4.9 as 192.168.0.2 (c1, k15, u'raspberrry').
1607334523: Client 192.168.0.2 disconnected.
1607334525: New connection from 192.168.4.6 on port 1883.
1607334525: New client connected from 192.168.4.6 as 192.168.0.1 (c1, k15, u'raspberrry').
1607334525: New connection from 192.168.4.7 on port 1883.
1607334525: New client connected from 192.168.4.7 as 192.168.0.3 (c1, k15, u'raspberrry').
1607334525: Client 192.168.0.3 disconnected.
1607334526: New connection from 192.168.4.15 on port 1883.
1607334526: New client connected from 192.168.4.15 as 192.168.0.4 (c1, k15, u'raspberrry').
1607334526: Client 192.168.0.4 disconnected.
1607334527: Client 192.168.0.1 disconnected.
1607334528: New connection from 192.168.4.7 on port 1883.
1607334528: New client connected from 192.168.4.7 as 192.168.0.3 (c1, k15, u'raspberrry').
1607334528: Client 192.168.0.3 disconnected.
1607334529: New connection from 192.168.4.9 on port 1883.
1607334529: New client connected from 192.168.4.9 as 192.168.0.2 (c1, k15, u'raspberrry').
1607334529: Client 192.168.0.2 disconnected.
1607334531: New connection from 192.168.4.7 on port 1883.
1607334531: New client connected from 192.168.4.7 as 192.168.0.3 (c1, k15, u'raspberrry').
1607334531: Client 192.168.0.3 disconnected.
1607334532: New connection from 192.168.4.9 on port 1883.
1607334532: New client connected from 192.168.4.9 as 192.168.0.2 (c1, k15, u'raspberrry').
1607334532: Client 192.168.0.2 disconnected.
1607334532: New connection from 192.168.4.15 on port 1883.
```

Output of Broker

pi@raspberrypi: ~

File Edit Tabs Help

```
pi@raspberrypi:~ $ python get_MQTT_data.py
MQTT to InfluxDB bridge
Connected with result code 0
home/room/distance 2376.72
home/room/pir 0
home/room/ir 0
home/room/distance 2375.00
home/room/distance1 206.55
home/room/pir 0
home/room/distance 2379.31
home/room/ir 0
home/room/pir 0
home/room/distance1 207.02
home/room/distance 2379.49
home/room/pir 0
home/room/ir 0
home/room/distance 2378.34
home/room/pir 0
home/room/distance 2382.58
home/room/distance1 207.89
home/room/ir 0
home/room/pir 0
home/room/distance 2380.68
home/room/ir 0
home/room/distance 2379.84
home/room/pir 0
home/room/distance1 205.68
home/room/distance 2381.95
home/room/pir 0
home/room/ir 0
home/room/distance 2376.48
home/room/pir 0
```

Output of Publishers

Wireshark Report

report4Dec20.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
55	4.466314	10.14.8.74	192.168.4.7	TCP	54	1883 → 58123 [ACK] Seq=5 Ack=77 Win=64164 Len=0
56	4.466699	10.14.8.74	192.168.4.7	TCP	54	1883 → 58123 [FIN, ACK] Seq=5 Ack=77 Win=64164 Len=0
57	4.469467	192.168.4.7	10.14.8.74	TCP	54	58123 → 1883 [FIN, ACK] Seq=77 Ack=5 Win=2140 Len=0
58	4.469630	10.14.8.74	192.168.4.7	TCP	54	1883 → 58123 [ACK] Seq=6 Ack=78 Win=64163 Len=0
59	4.472208	192.168.4.7	10.14.8.74	TCP	54	58123 → 1883 [ACK] Seq=78 Ack=6 Win=2139 Len=0
60	5.392518	fe80::ad80:df4e:5d4...	ff02::fb	MDNS	144	Standard query response 0x0000 AAAA, cache flush fe80::ad80:df4e:5d4e:
61	5.404507	192.168.4.1	224.0.0.251	MDNS	87	Standard query response 0x0000 A, cache flush 192.168.4.1
62	5.508239	fe80::ad80:df4e:5d4...	ff02::fb	MDNS	144	Standard query response 0x0000 AAAA, cache flush fe80::ad80:df4e:5d4e:
63	5.510335	192.168.4.1	224.0.0.251	MDNS	87	Standard query response 0x0000 A, cache flush 192.168.4.1
64	6.207913	192.168.4.9	10.14.8.74	TCP	62	51551 → 1883 [SYN] Seq=0 Win=2144 Len=0 MSS=536 SACK_PERM=1
65	6.208093	10.14.8.74	192.168.4.9	TCP	62	1883 → 51551 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
66	6.210914	192.168.4.9	10.14.8.74	TCP	54	51551 → 1883 [ACK] Seq=1 Ack=1 Win=2144 Len=0
67	6.211302	192.168.4.9	10.14.8.74	MQTT	100	Connect Command
68	6.211375	10.14.8.74	192.168.4.9	TCP	54	1883 → 51551 [ACK] Seq=1 Ack=47 Win=64194 Len=0
69	6.211646	10.14.8.74	192.168.4.9	MQTT	58	Connect Ack
70	6.403471	192.168.4.9	10.14.8.74	MQTT	72	Publish Message [home/room/pir]

< >

> Frame 62: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits)

> Ethernet II, Src: Raspberr_0b:51:39 (dc:a6:32:0b:51:39), Dst: IPv6mcast_fb (33:33:00:00:00:fb)

> Internet Protocol Version 6, Src: fe80::ad80:df4e:5d4e:a170, Dst: ff02::fb

> User Datagram Protocol, Src Port: 5353, Dst Port: 5353

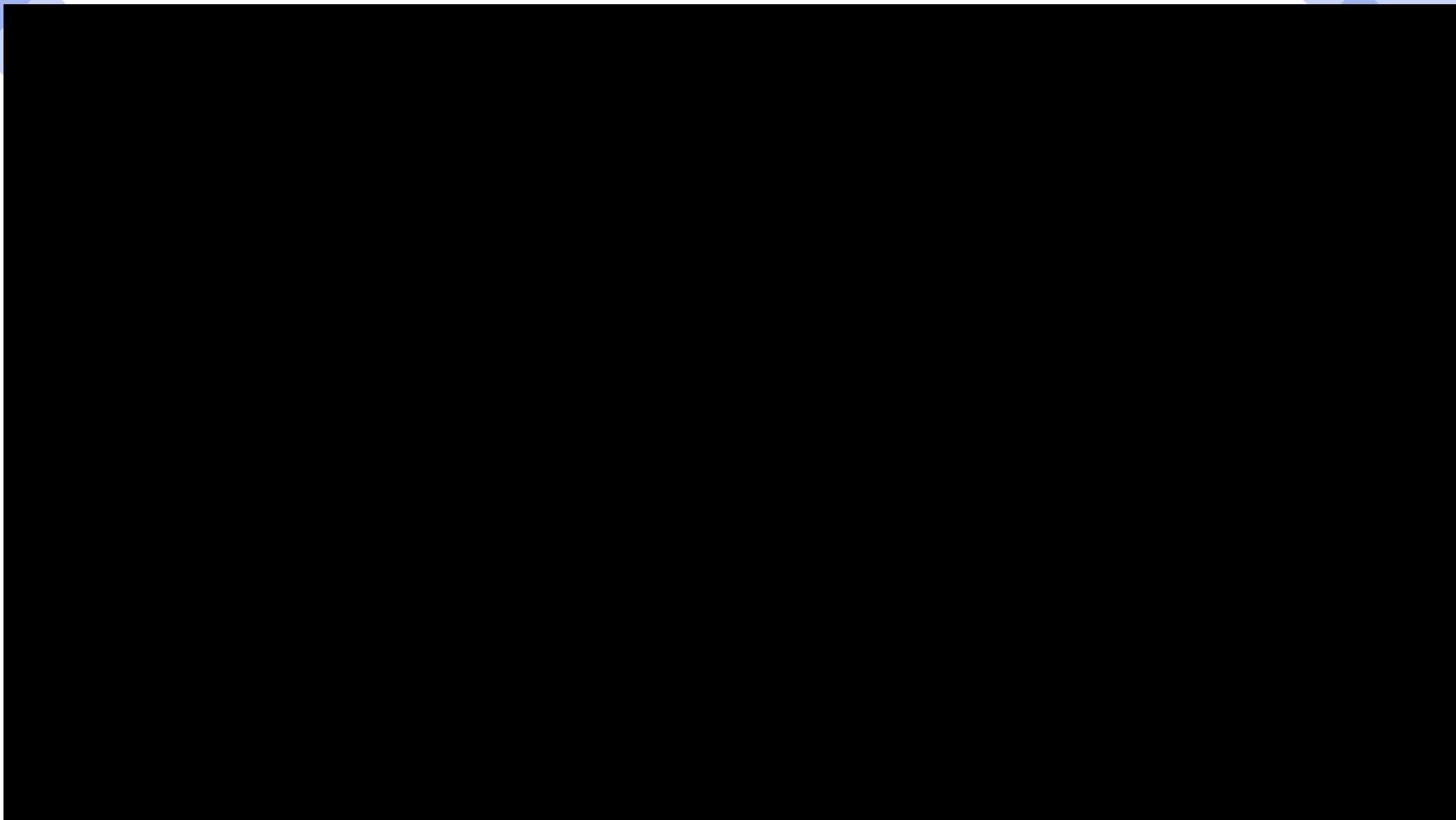
> Multicast Domain Name System (response)

0000	33 33 00 00 00 fb dc a6 32 0b 51 39 86 dd 60 0c	33..... 2·Q9···
0010	a9 a2 00 5a 11 ff fe 80 00 00 00 00 00 00 ad 80	...Z.....
0020	df 4e 5d 4e a1 70 ff 02 00 00 00 00 00 00 00 00	·N]N·p·
0030	00 00 00 00 00 fb 14 e9 14 e9 00 5a a3 24 00 00 ·Z·\$·
0040	84 00 00 00 00 01 00 00 00 00 24 66 65 38 30 2d ·\$fe80-
0050	30 2d 30 2d 30 2d 61 64 38 30 2d 64 66 34 65 2d	0-0-0-ad 80-df4e-
0060	35 64 34 65 2d 61 31 37 30 2d 77 6c 61 6e 30 05	5d4e-a17 0-wlan0·

report4Dec20.pcap

Packets: 64658 · Displayed: 64658 (100.0%)

Profile: Default



IoT Traffic Classification

Demo

Machine Learning in IoT

- Machine learning in IoT [22] [23] [24] as follows:

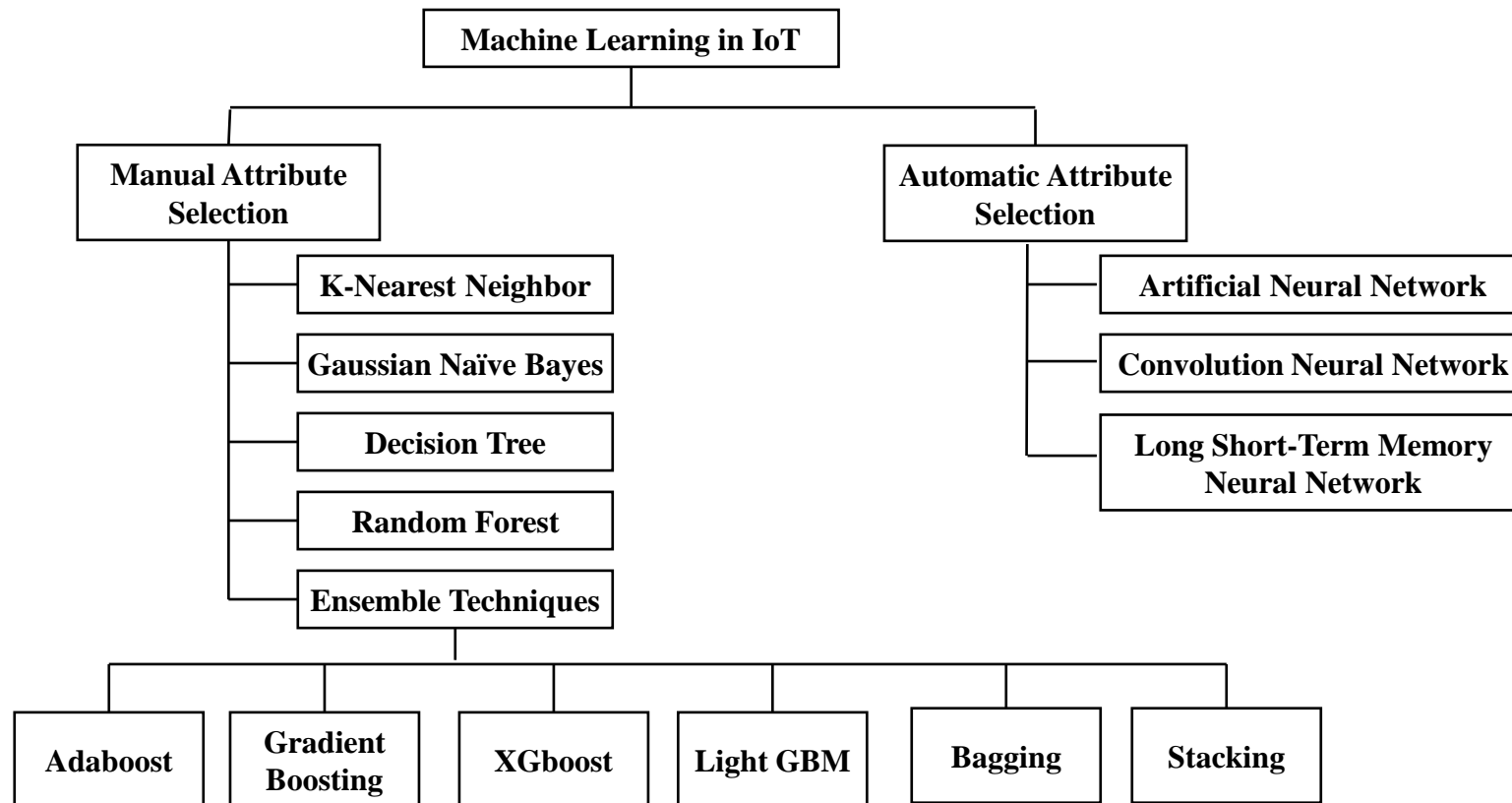


Fig. 5: Machine Learning Techniques used for IoT Classification

[5]. Sivanathan, Arunan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. "Classifying IoT devices in smart environments using network traffic characteristics." IEEE Transactions on Mobile Computing, vol. 18, pp. 1745-1759, 2019

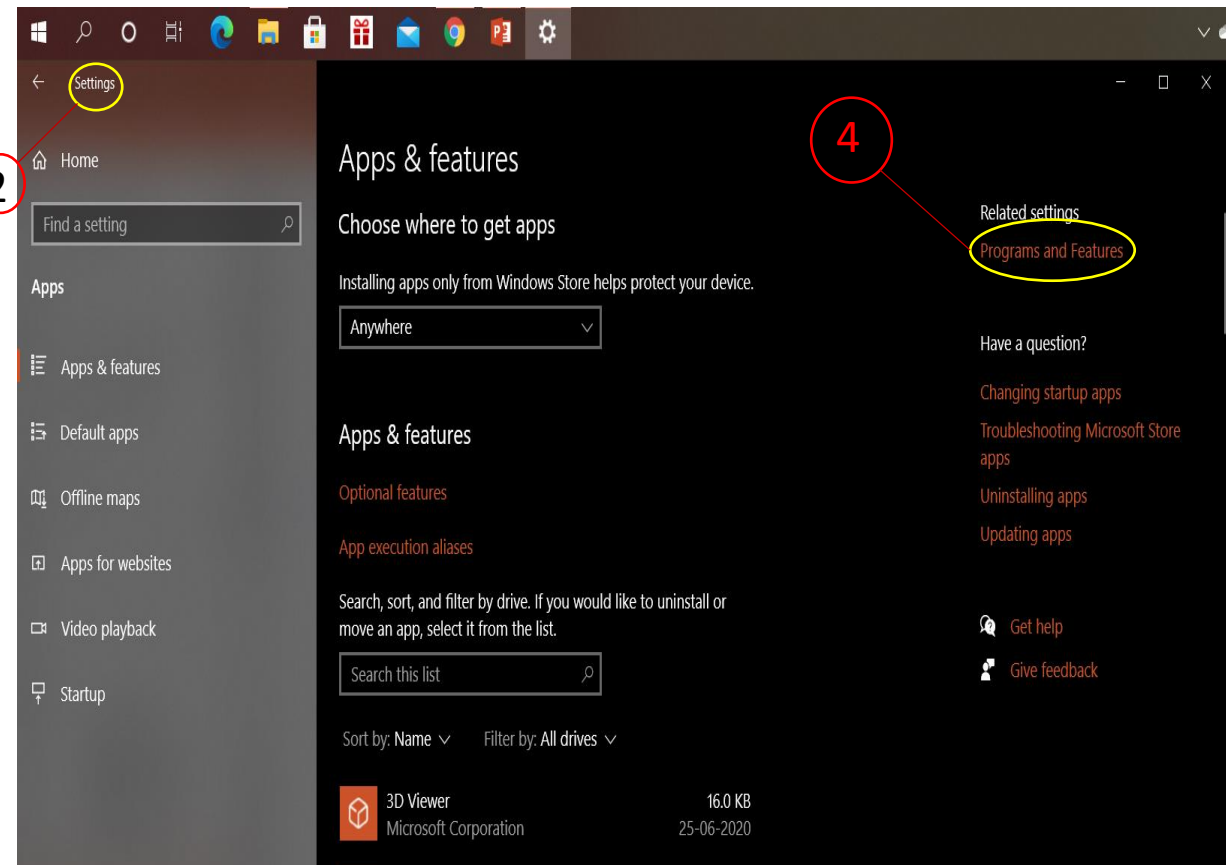
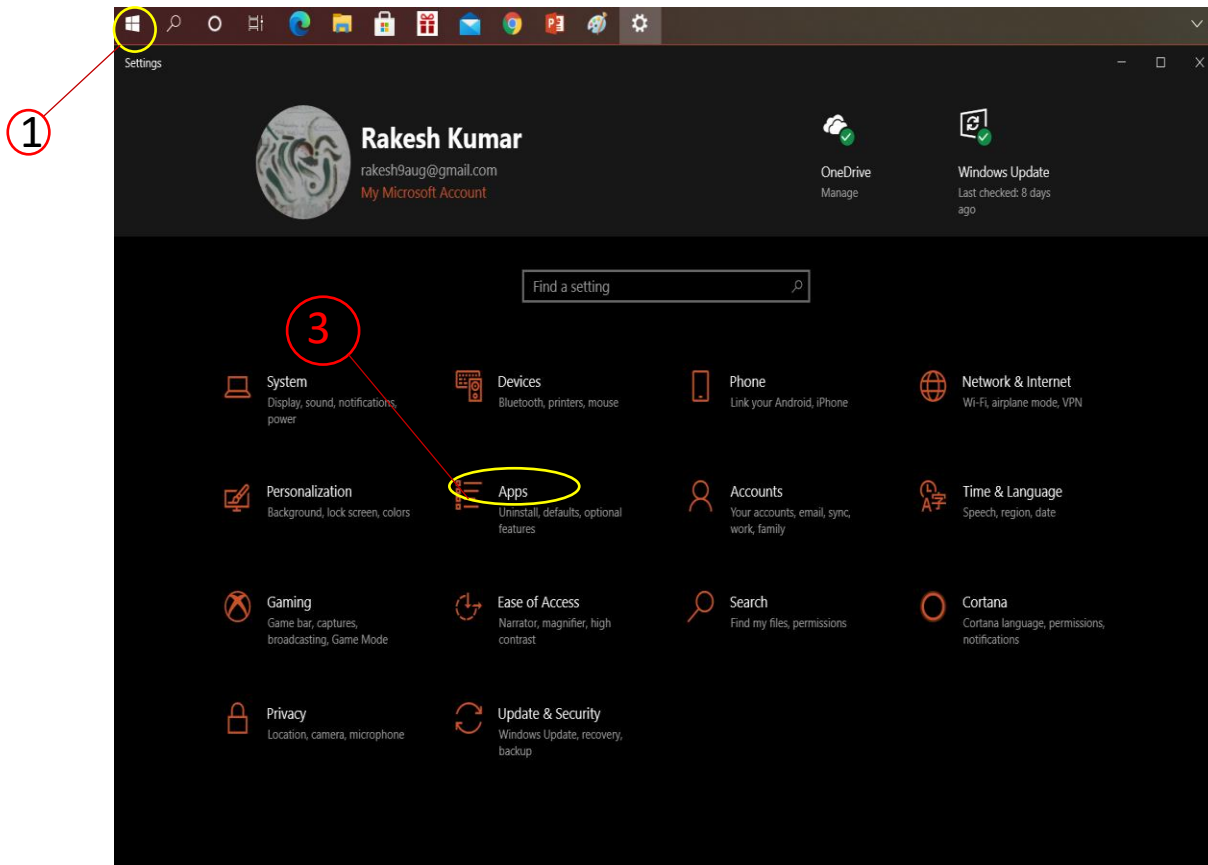
[23]. Pinheiro, Antônio J., Jeandro de M. Bezerra, Caio AP Burgardt, and Divanilson R. Campelo, "Identifying IoT devices and events based on packet length from encrypted traffic" Computer Communications, vol. 144, pp. 8-17, 2019.

[24]. A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, "Managing iotcyber-security using programmable telemetry and machine learning," IEEE Transactions on Network and Service Management, vol. 17, pp.60-74, 2020

Installation of Kali Linux on Windows (WSL) GUI

- Installation Steps as Follows:

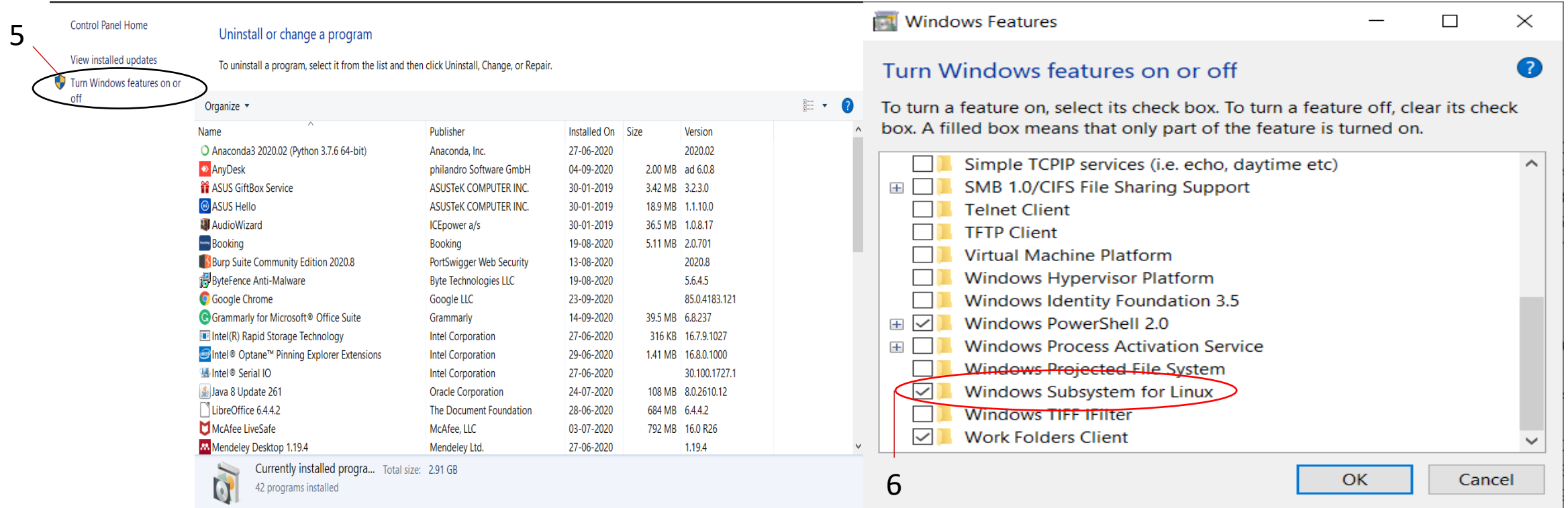
- Step1 : 1 Click on Windows Start Button > 2 Click Settings > 3 Click Apps > 4 Click Programs & Features (Top Right Corner) > 5 Click Turn Windows Features on or off > 6 Tick (✓) on Window Subsystem for Linux



Installation of Kali Linux on Windows (WSL) GUI

- Installation Steps as Follows:

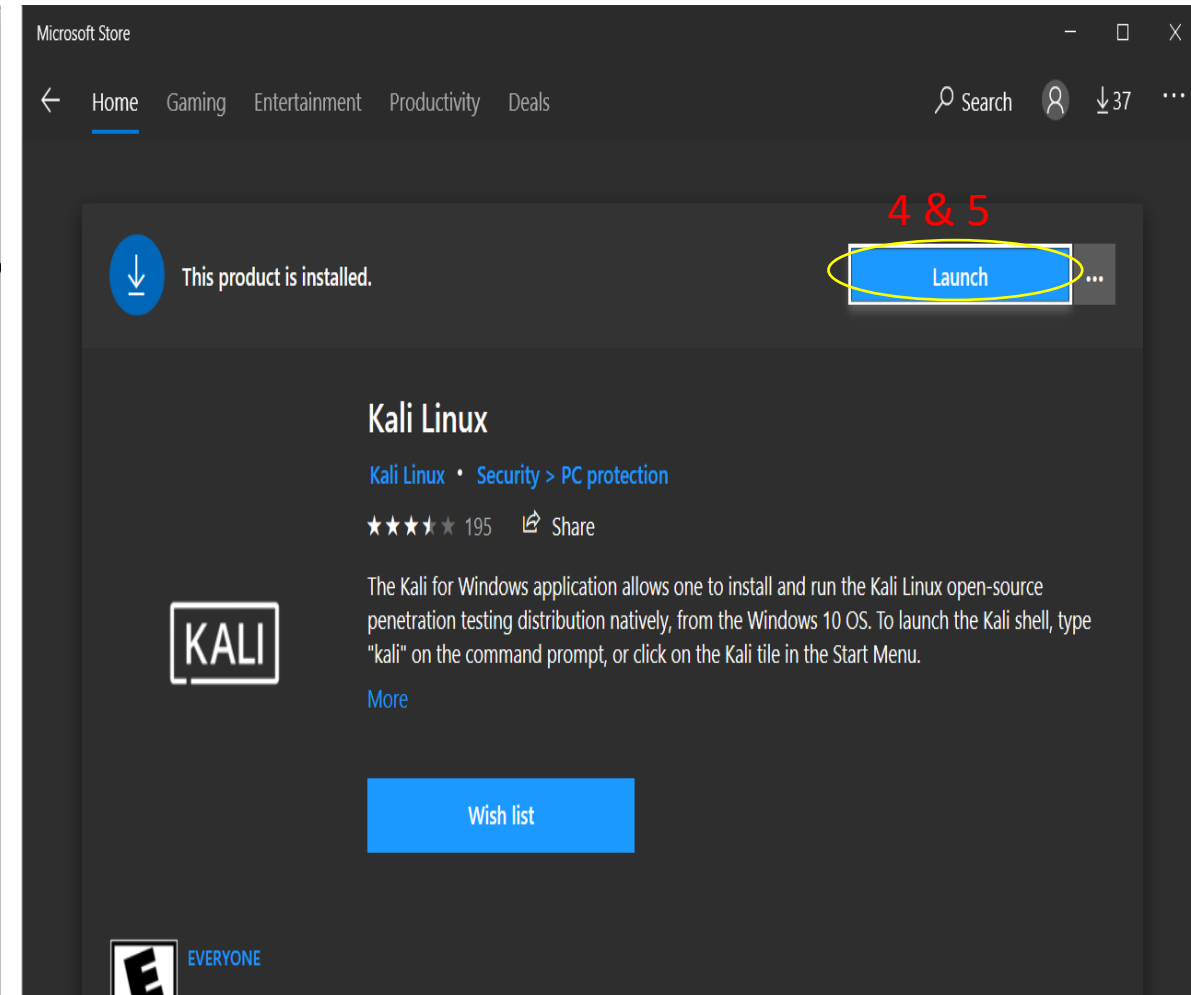
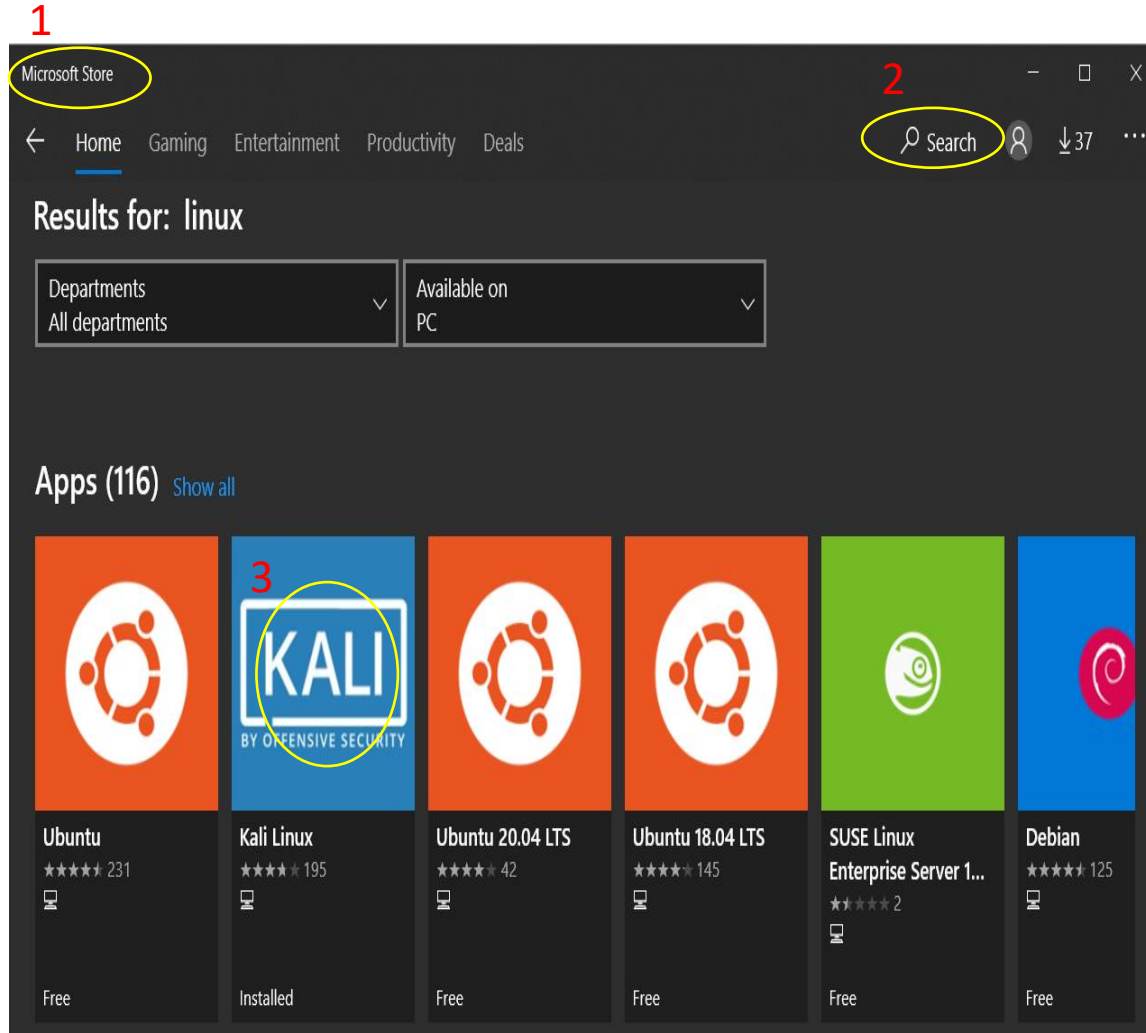
- Step1 : 1 Click on Windows Start Button > 2 Click Settings > 3 Click Apps > 4 Click Programs & Features (Top Right Corner) > 5 Click Turn Windows Features on or off > 6 Tick (✓) on Window Subsystem for Linux



Installation of Kali Linux on Windows (WSL) GUI

- Installation Steps as Follows:

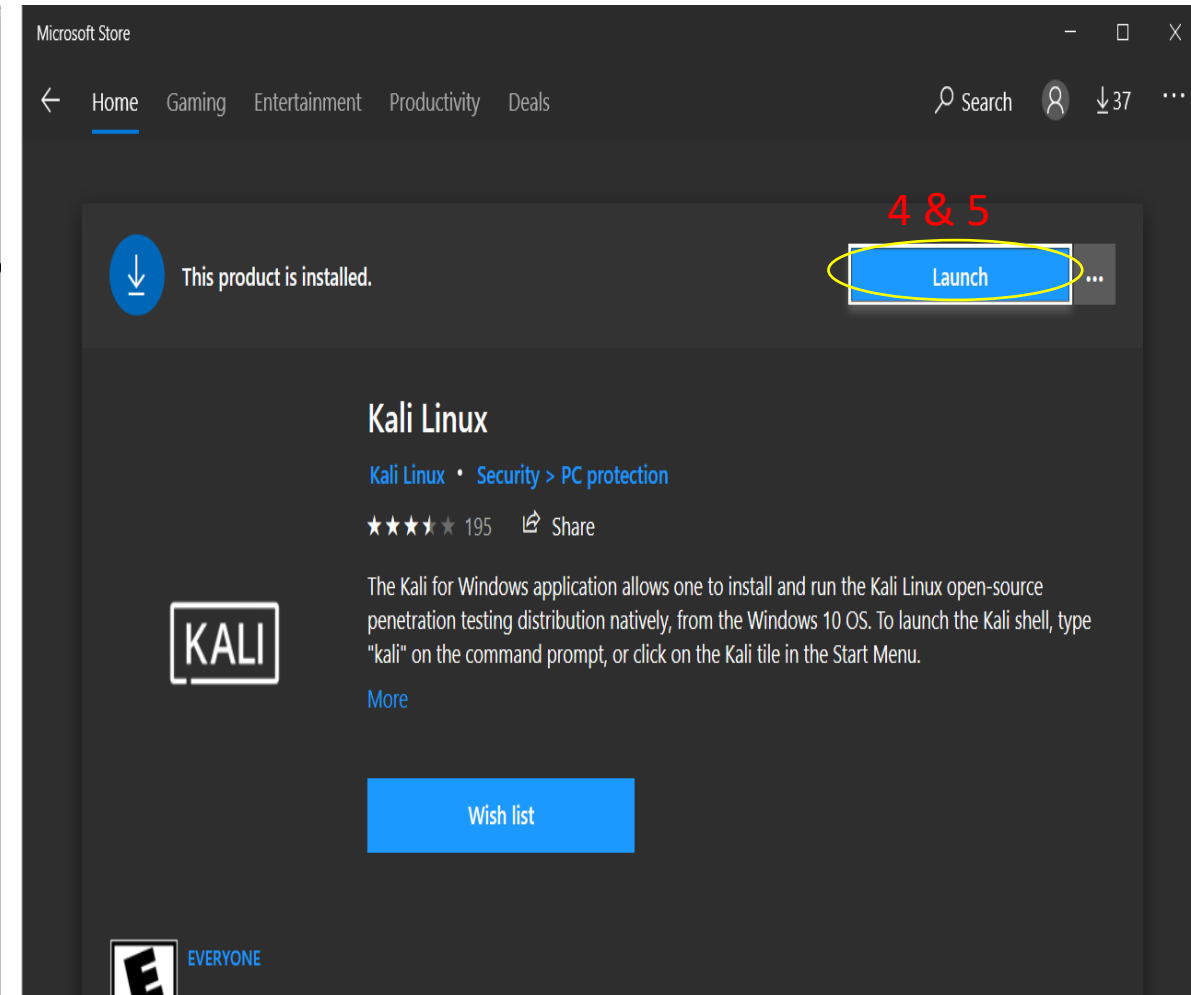
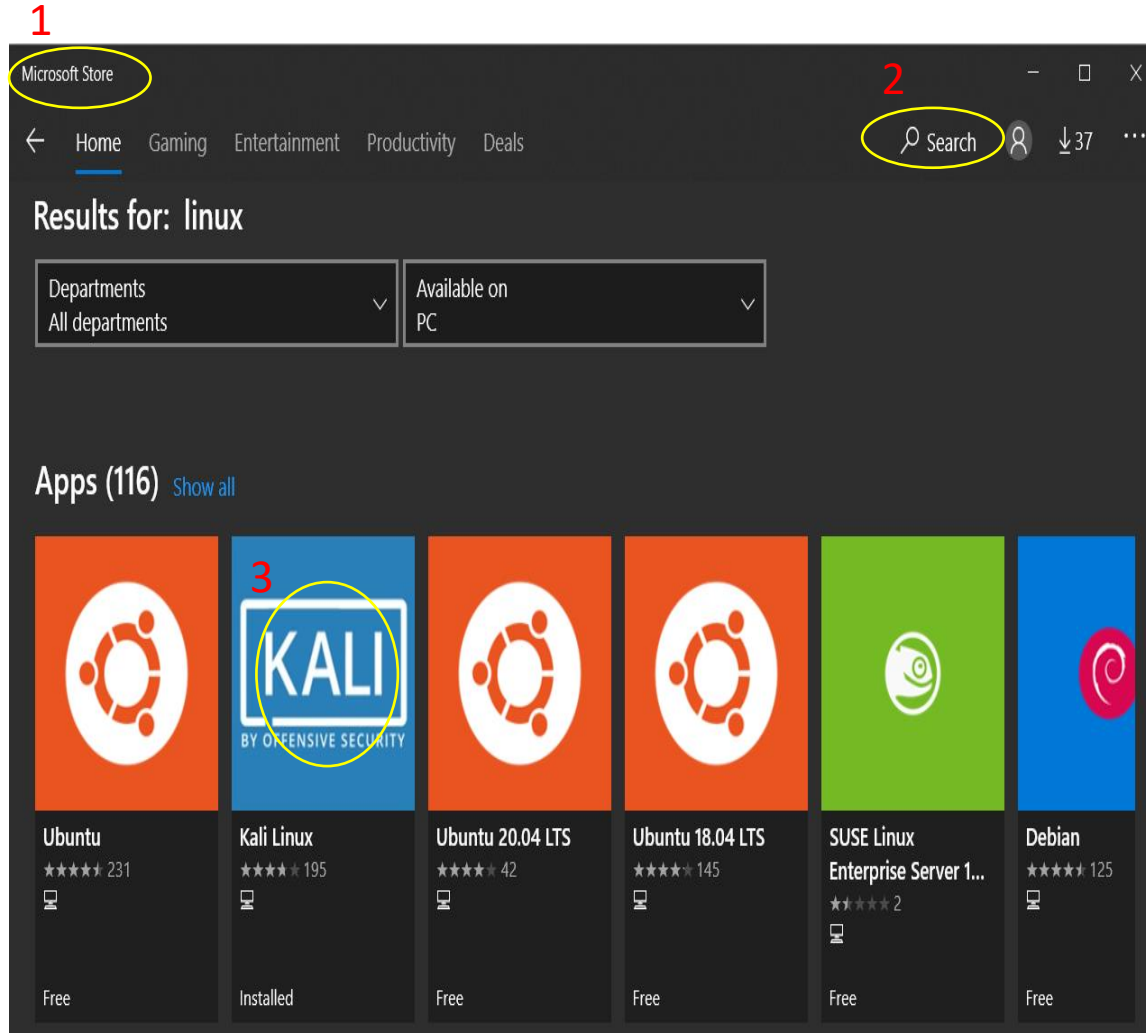
➤ Step2 : 1 Click on Microsoft Store > 2 Search Linux > 3 Click on Kali Linux App > 4 Click on Get for Download App > 5 Click on Launch



Installation of Kali Linux on Windows (WSL) GUI

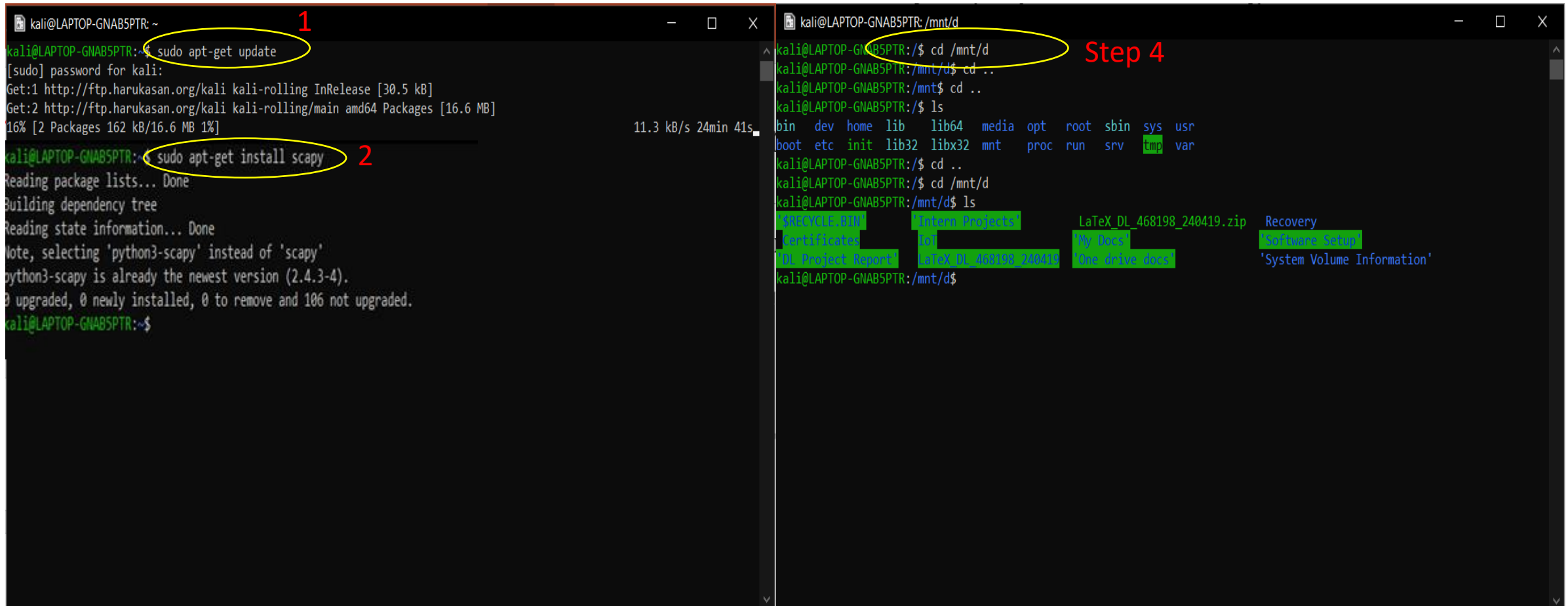
- Installation Steps as Follows:

➤ Step2 : 1 Click on Microsoft Store > 2 Search Linux > 3 Click on Kali Linux App > 4 Click on Get for Download App > 5 Click on Launch



Installation of Kali Linux on Windows (WSL) GUI

- Installation Steps as Follows:
 - Step3 : 1 Update your Kali Linux > 2 Install Scapy & Use installed software
 - Step4: Mount local drives



```
kali@LAPTOP-GNAB5PTR: ~  
kali@LAPTOP-GNAB5PTR:~$ sudo apt-get update  
[sudo] password for kali:  
Get:1 http://ftp.harukasan.org/kali kali-rolling InRelease [30.5 kB]  
Get:2 http://ftp.harukasan.org/kali kali-rolling/main amd64 Packages [16.6 MB]  
16% [2 Packages 162 kB/16.6 MB 1%]  
11.3 kB/s 24min 41s  
  
kali@LAPTOP-GNAB5PTR:~$ sudo apt-get install scapy  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
Note, selecting 'python3-scapy' instead of 'scapy'  
python3-scapy is already the newest version (2.4.3-4).  
0 upgraded, 0 newly installed, 0 to remove and 106 not upgraded.  
kali@LAPTOP-GNAB5PTR:~$  
  
kali@LAPTOP-GNAB5PTR: /mnt/d  
kali@LAPTOP-GNAB5PTR:/$ cd /mnt/d  
kali@LAPTOP-GNAB5PTR:/mnt/d$ cd ..  
kali@LAPTOP-GNAB5PTR:/mnt$ cd ..  
kali@LAPTOP-GNAB5PTR:/$ ls  
bin dev home lib lib64 media opt root sbin sys usr  
boot etc init lib32 libx32 mnt proc run srv tmp var  
kali@LAPTOP-GNAB5PTR:/$ cd ..  
kali@LAPTOP-GNAB5PTR:/$ cd /mnt/d  
kali@LAPTOP-GNAB5PTR:/mnt/d$ ls  
'$RECYCLE.BIN' 'Intern Projects' LaTeX_DL_468198_240419.zip Recovery  
'Certificates' 'lo' 'My Docs' 'Software Setup'  
'DL Project Report' 'LaTeX DL_468198_240419' 'One drive docs' 'System Volume Information'  
kali@LAPTOP-GNAB5PTR:/mnt/d$
```

Pre-processing of IoT Traffic

- Network Traffic Capturing: Using Wireshark/TCPdump capturing the IoT network traffic.
- Splitting: Separate the IoT devices traffic from whole network traffic traces.
- Flow Construction: Construction of two types flows such as TCP & UDP from IoT traffic.
- Feature Extraction: Extract the three types of features such as packet level, flow level & behavior level.

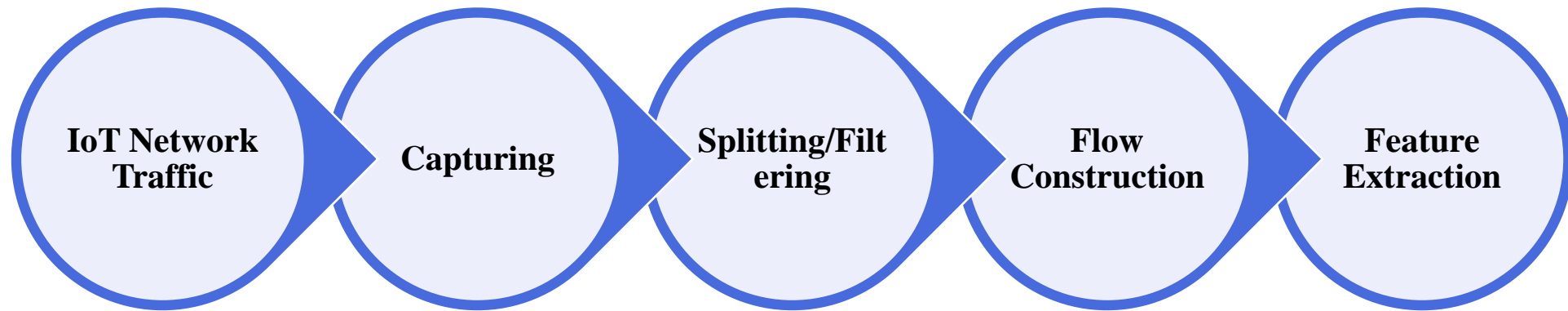


Fig. 6: Preprocessing Steps for IoT Traffic Classification

IoT Network Traffic Capturing

- Network Traffic Capturing: Using Wireshark/TCPdump capturing the IoT network traffic.
 - Download Wireshark for Windows, Linux, MAC OS :
<https://www.wireshark.org/download.html>
 - Wireshark
 - ✓ Free & open-source packet analyzer
 - ✓ Network troubleshooter
 - ✓ Analysis
 - ✓ Software and communications protocol development
 - ✓ Education

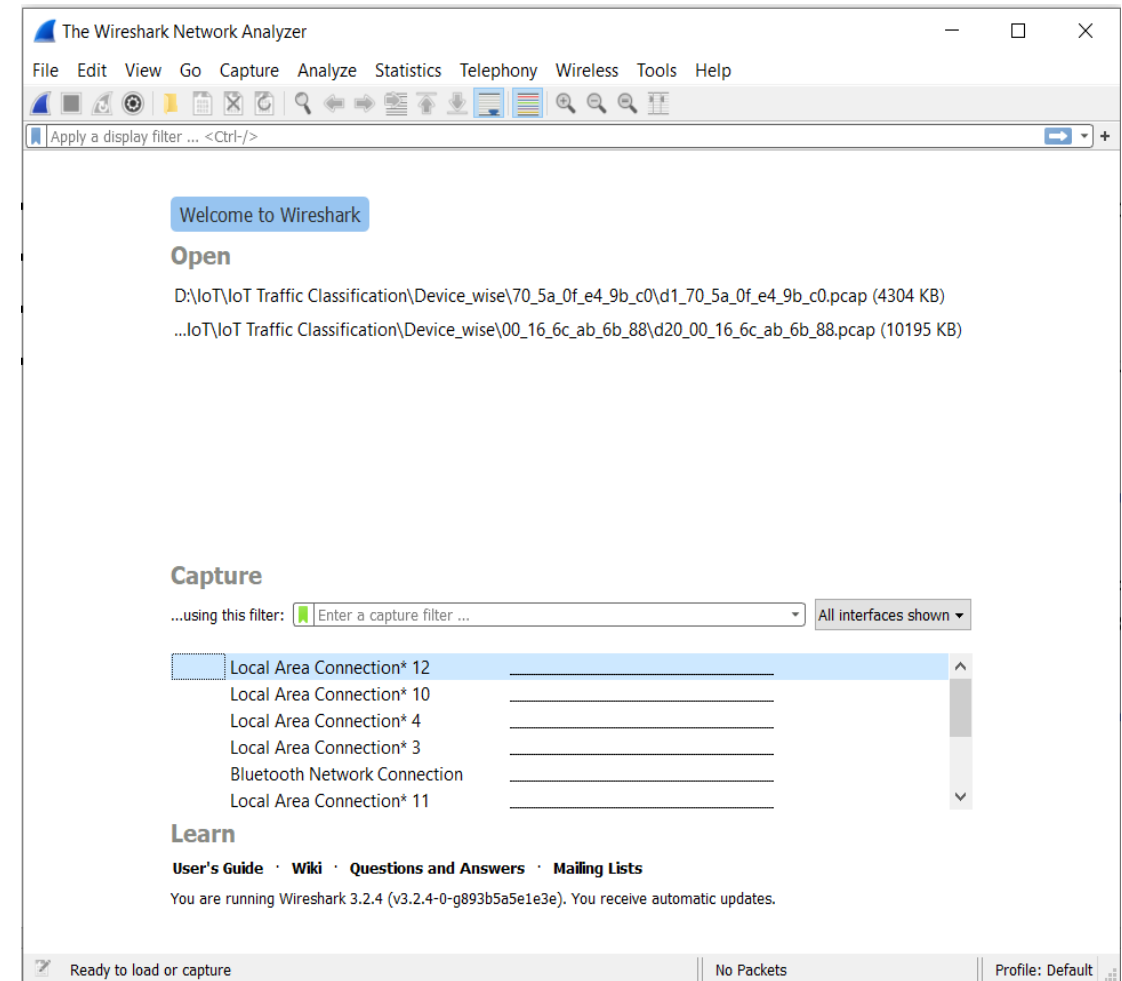


Fig. 7: IoT Traffic Capturing

IoT Network Traffic Capturing

- Network Traffic Capturing: Using Wireshark/TCPdump capturing the IoT network traffic.

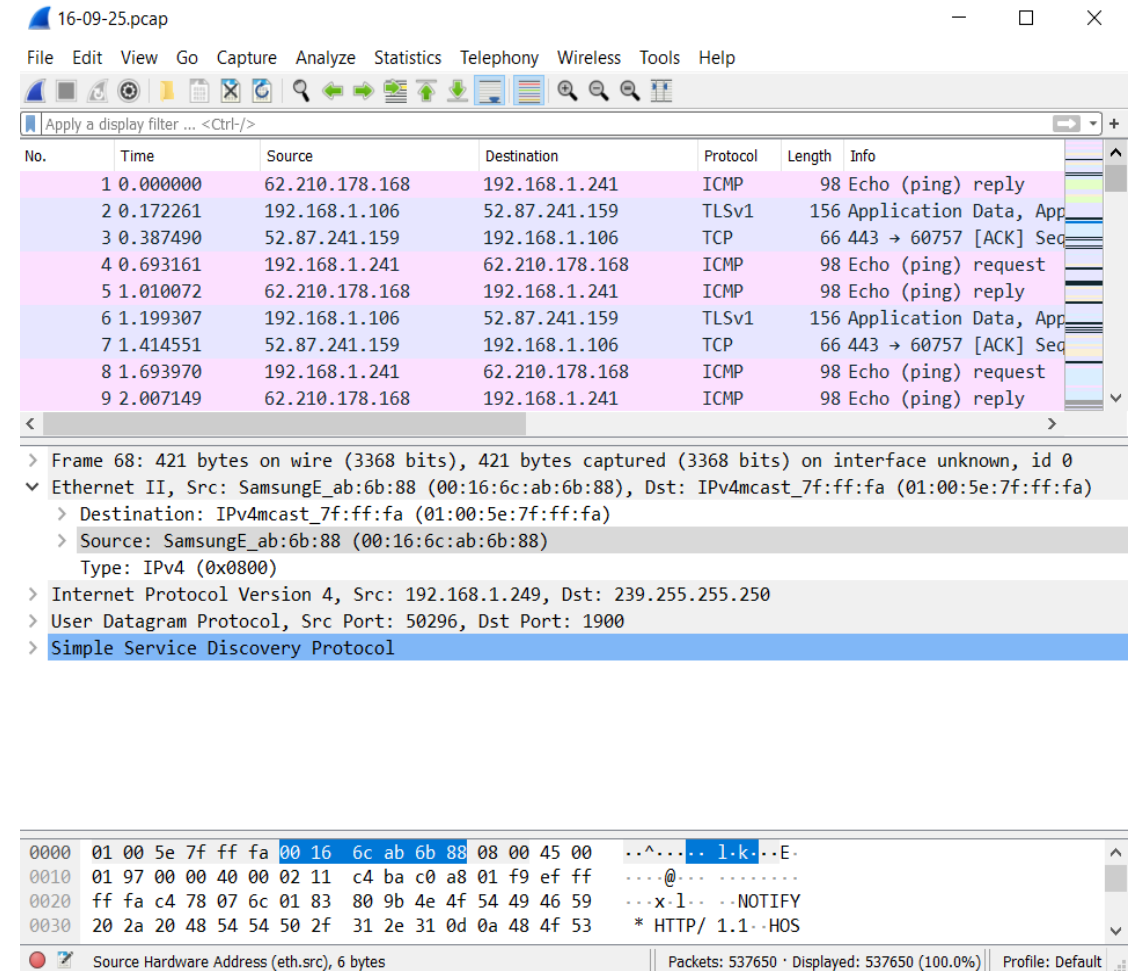
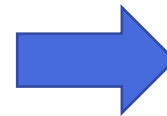
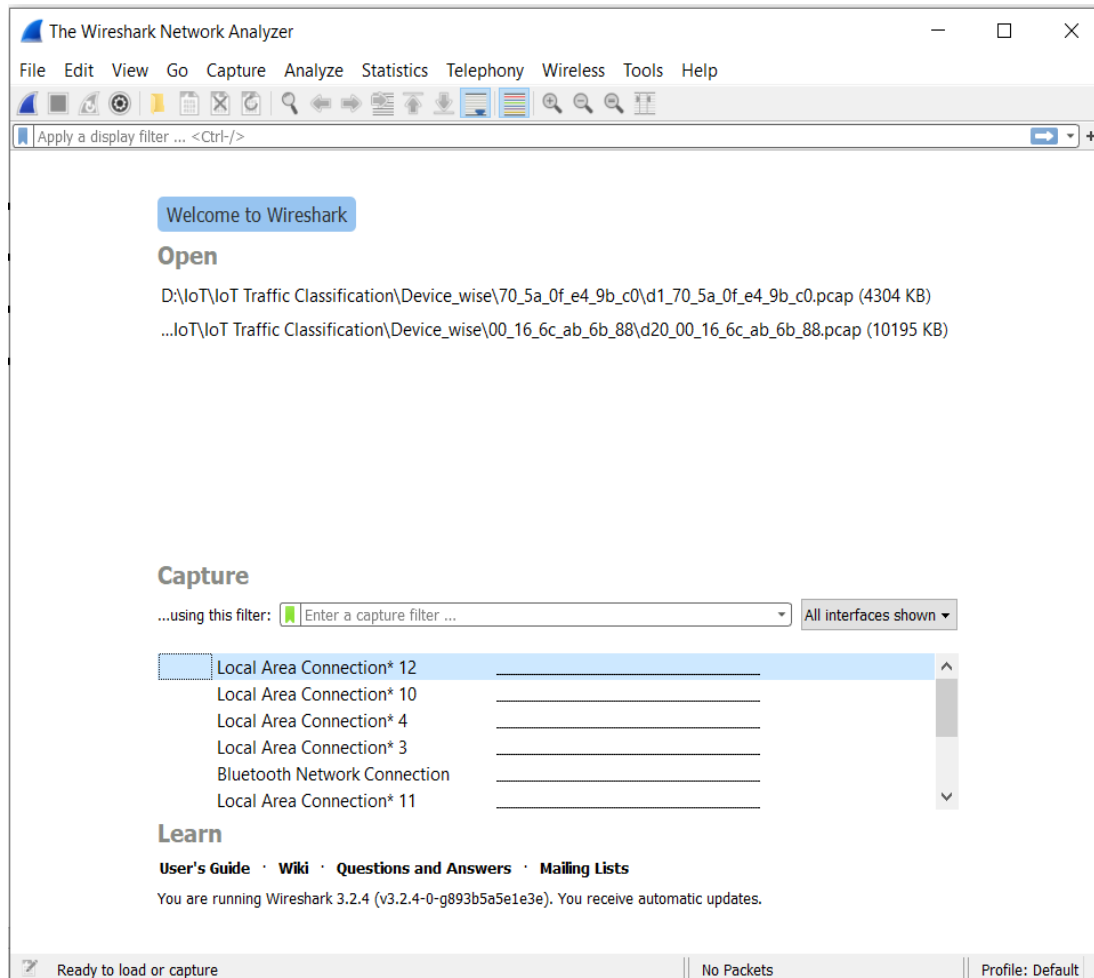


Fig. 7: IoT Traffic Capturing

IoT Network Traffic Splitting/Filtering

- Splitting/Filtering: Separate the IoT devices traffic from whole network traffic traces.
 - Method 1: Using Wireshark

16-09-25.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Filter by MAC Address

`eth.addr == 00:16:6c:ab:6b:88`

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	62.210.178.168	192.168.1.241	ICMP	98	Echo (ping) reply
2	0.172261	192.168.1.106	52.87.241.159	TLSv1	156	Application Data, App
3	0.387490	52.87.241.159	192.168.1.106	TCP	66	443 → 60757 [ACK] Seq
4	0.693161	192.168.1.241	62.210.178.168	ICMP	98	Echo (ping) request
5	1.010072	62.210.178.168	192.168.1.241	ICMP	98	Echo (ping) reply
6	1.199307	192.168.1.106	52.87.241.159	TLSv1	156	Application Data, App
7	1.414551	52.87.241.159	192.168.1.106	TCP	66	443 → 60757 [ACK] Seq
8	1.693970	192.168.1.241	62.210.178.168	ICMP	98	Echo (ping) request
9	2.007149	62.210.178.168	192.168.1.241	ICMP	98	Echo (ping) reply

> Frame 68: 421 bytes on wire (3368 bits), 421 bytes captured (3368 bits) on interface unknown, id 0

▼ Ethernet II, Src: SamsungE_ab:6b:88 (00:16:6c:ab:6b:88), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)

> Destination: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)

> Source: SamsungE_ab:6b:88 (00:16:6c:ab:6b:88)

Type: IPv4 (0x0800)

> Internet Protocol Version 4, Src: 192.168.1.249, Dst: 239.255.255.250

> User Datagram Protocol, Src Port: 50296, Dst Port: 1900

> Simple Service Discovery Protocol

0000 01 00 5e 7f ff fa 00 16 6c ab 6b 88 08 00 45 00 ..^... 1.k...E.

0010 01 97 00 00 40 00 02 11 c4 ba c0 a8 01 f9 ef ff ...@.....

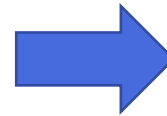
0020 ff fa c4 78 07 6c 01 83 80 9b 4e 4f 54 49 46 59 ...x.l...NOTIFY

0030 20 2a 20 48 54 54 50 2f 31 2e 31 0d 0a 48 4f 53 * HTTP/ 1.1..HOS

Source Hardware Address (eth.src), 6 bytes

Packets: 537650 · Displayed: 537650 (100.0%)

Profile: Default



00_16_6c_ab_6b_88.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.249	54.67.41.98	TCP	188	42284 → 5222 [PSH, AC
2	0.162604	54.67.41.98	192.168.1.249	TCP	204	5222 → 42284 [PSH, AC
3	0.164105	192.168.1.249	54.67.41.98	TCP	66	42284 → 5222 [ACK] Se
4	5.172367	Tp-LinkT_51:33:ea	SamsungE_ab:6b:88	ARP	42	Who has 192.168.1.249
5	5.173419	SamsungE_ab:6b:88	Tp-LinkT_51:33:ea	ARP	42	192.168.1.249 is at 0
6	14.474129	192.168.1.249	239.255.255.250	SSDP	421	NOTIFY * HTTP/1.1
7	14.475600	192.168.1.249	239.255.255.250	SSDP	493	NOTIFY * HTTP/1.1
8	14.480066	192.168.1.249	239.255.255.250	SSDP	489	NOTIFY * HTTP/1.1
9	14.480247	192.168.1.249	239.255.255.250	SSDP	469	NOTIFY * HTTP/1.1

> Frame 1: 188 bytes on wire (1504 bits), 188 bytes captured (1504 bits) on interface unknown, id 0

▼ Ethernet II, Src: SamsungE_ab:6b:88 (00:16:6c:ab:6b:88), Dst: Tp-LinkT_51:33:ea (14:cc:20:51:33:ea)

> Destination: Tp-LinkT_51:33:ea (14:cc:20:51:33:ea)

> Source: SamsungE_ab:6b:88 (00:16:6c:ab:6b:88)

Type: IPv4 (0x0800)

> Internet Protocol Version 4, Src: 192.168.1.249, Dst: 54.67.41.98

> Transmission Control Protocol, Src Port: 42284, Dst Port: 5222, Seq: 1, Ack: 1, Len: 122

0000 14 cc 20 51 33 ea 00 16 6c ab 6b 88 08 00 45 00 ..Q3... 1.k...E.

0010 00 ae 0b 4a 40 00 00 06 0c ba c0 a8 01 f9 36 43 ...J@...6C

0020 29 62 a5 2c 14 66 cc 89 22 c5 4e 1f d5 5c 80 18)b.,-f...".N..\

0030 2b 84 07 1e 00 00 01 01 08 0a 06 eb 16 2d 52 c2 +.....R.

Source Hardware Address (eth.src), 6 bytes

Packets: 31904 · Displayed: 31904 (100.0%)

Profile: Default

Fig. 8: IoT Traffic Splitting/Filtering

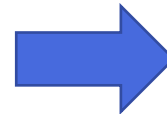
IoT Network Traffic Splitting/Filtering

- Splitting/Filtering: Separate the IoT devices traffic from whole network traffic traces.
 - Method 2: Bash Script

```
Pcap_split - Notepad
File Edit Format View Help
# Pcap Splitter

#macs=(44:65:0d:56:cc:d3 e0:76:d0:3f:00:ae 70:88:6b:10:0f:c6 b4:75:0e:ec:e5:a9 ec:1a:59:83:28:11 ec:1a:59:79:f4:89 74:6a:89:00:2e:25
7c:70:bc:5d:5e:dc 30:8c:fb:2f:e4:b2 6c:ad:f8:5e:e4:61 28:c2:dd:ff:a5:2d 70:5a:0f:e4:9b:c0 74:c6:3b:29:d7:1d d0:73:d5:01:83:08
18:b4:30:25:be:e4 70:ee:50:18:34:43 70:ee:50:03:b8:ac 00:17:88:2b:9a:25 e0:76:d0:33:bb:85 88:4a:ea:31:66:9d 00:16:6c:ab:6b:88
d0:52:a8:00:67:5e f4:f2:6d:93:51:f1 50:c7:bf:00:56:39 18:b7:9e:02:20:44 00:24:e4:10:ee:4c 00:24:e4:1b:6f:96 00:24:e4:20:28:c6)
#ips=( 192.168.202.68 192.168.202.79 192.168.229.153 192.168.23.253 )
macs=(44:65:0d:56:cc:d3 e0:76:d0:3f:00:ae)

for mac in ${macs[*]}
do
    #for ip in ${ips[*]}
    do
        echo "$mac" >&2
        #echo "$ip" >&2
        tshark -r "/mnt/d/IoT/Test1/T/16-09-25.pcap" -Y "eth.addr == $mac" -w "/mnt/d/IoT/Test1/Out/$mac.pcap"
    done
done
```



```
kali@LAPTOP-GNAB5PTR: /mnt/d/IoT/S
kali@LAPTOP-GNAB5PTR:/mnt/d/IoT/S$ chmod +x Pcap_split.sh
kali@LAPTOP-GNAB5PTR:/mnt/d/IoT/S$ ./Pcap_split.sh
44:65:0d:56:cc:d3
e0:76:d0:3f:00:ae
kali@LAPTOP-GNAB5PTR:/mnt/d/IoT/S$
```

Fig. 8: IoT Traffic Splitting/Filtering

IoT Traffic Flow Construction

- Flow Construction: Construction of two types flows such as TCP & UDP from IoT traffic using scapy & python.

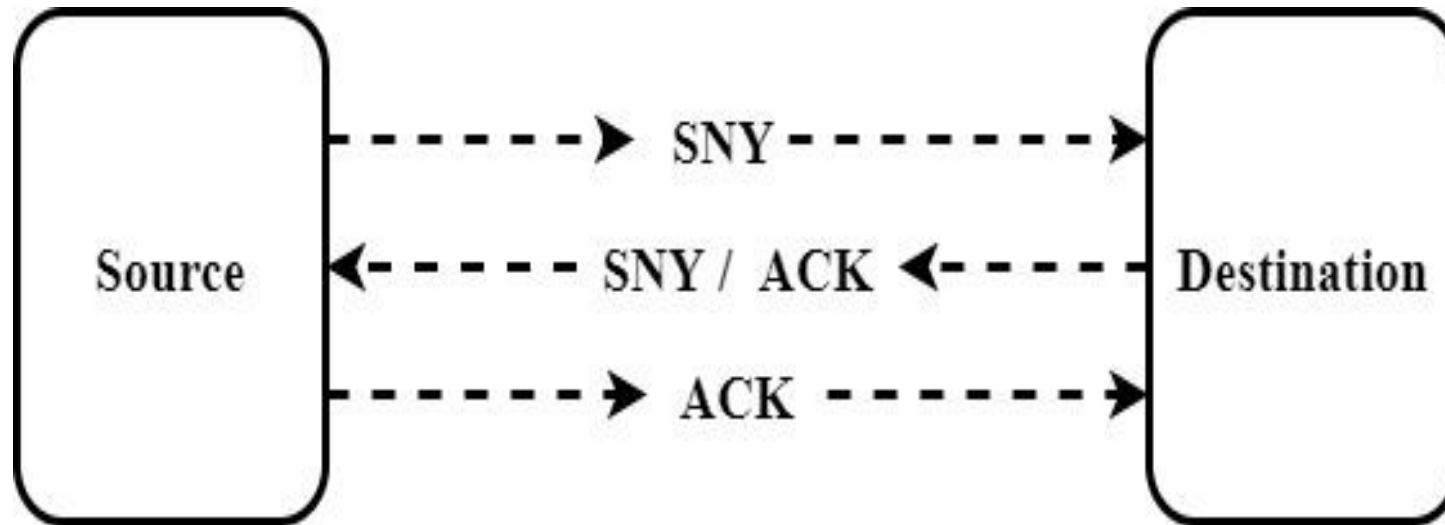


Fig. 9: IoT Traffic Flow Construction

IoT Traffic Flow Construction

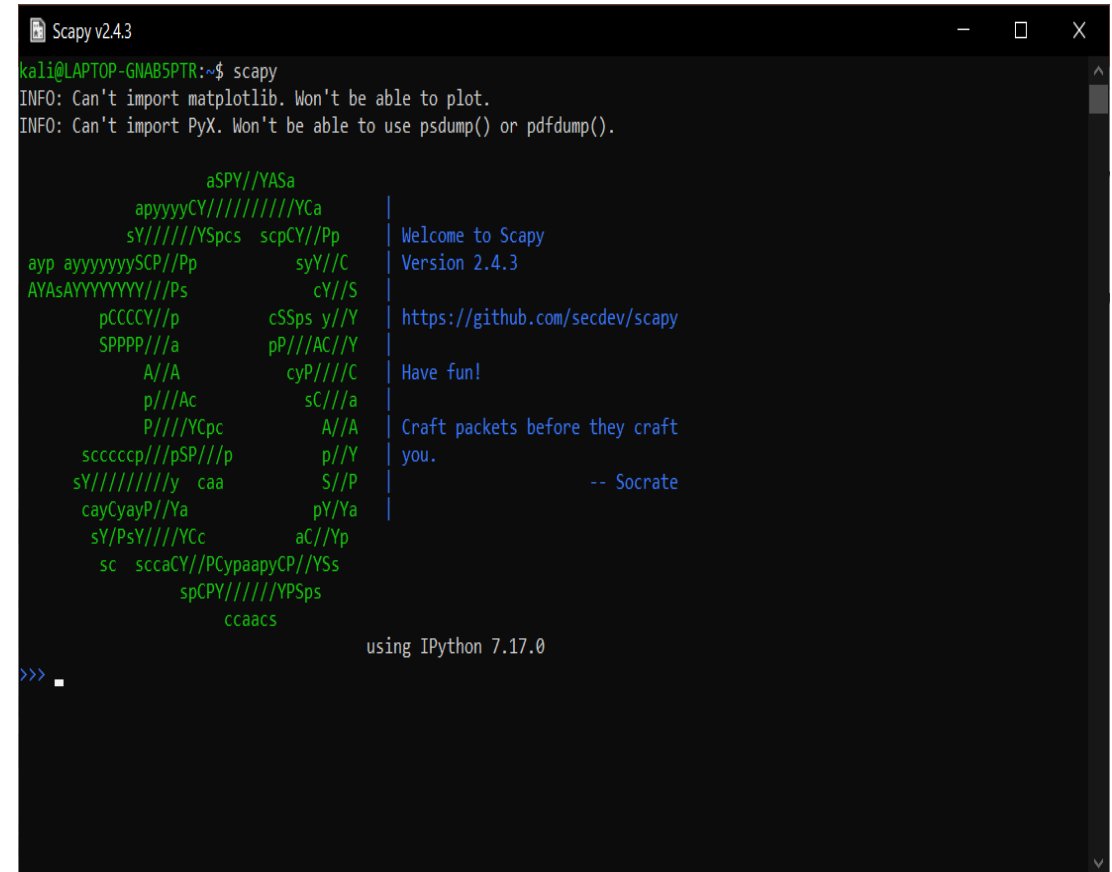
- Flow Construction: Construction of two types flows such as TCP & UDP from IoT traffic using scapy & python.

➤ Download Scapy :

<https://scapy.readthedocs.io/en/latest/installation.html>

➤ Scapy

- ✓ Packet Manipulation Python Tool
- ✓ Flow Construction
- ✓ Forge or Decode Packets
- ✓ Scanning
- ✓ Tracerouting
- ✓ Attacks
- ✓ Network Discovery



```
Scapy v2.4.3
kali@LAPTOP-GNAB5PTR:~$ scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

aSPY//YASa
  apyyyyCY/////////YCa
    sY////////YSpCs  scpCY//Pp
ayp ayyyyyySCP//Pp    syY//C
AYAsAYYYYYYYY//Ps    cY//S
  pCCCCY//p          cSSps y//Y
  SPPPP//a           pP//AC//Y
    A//A             cyP///C
      p///Ac         sC///a
        P///YCpc     A//A
  scccccp///pSP///p   p//Y
sY/////////y caa      S//P
cayCyayP//Ya         pY/Ya
sY/PsY///Ycc         aC//Yp
  sc  sccaCY//PCypaapyCP//YSs
        spCPY////////YPSps
          ccaacs

Welcome to Scapy
Version 2.4.3
https://github.com/secdev/scapy
Have fun!
Craft packets before they craft
you.
-- Socrates

using IPython 7.17.0
>>> .
```

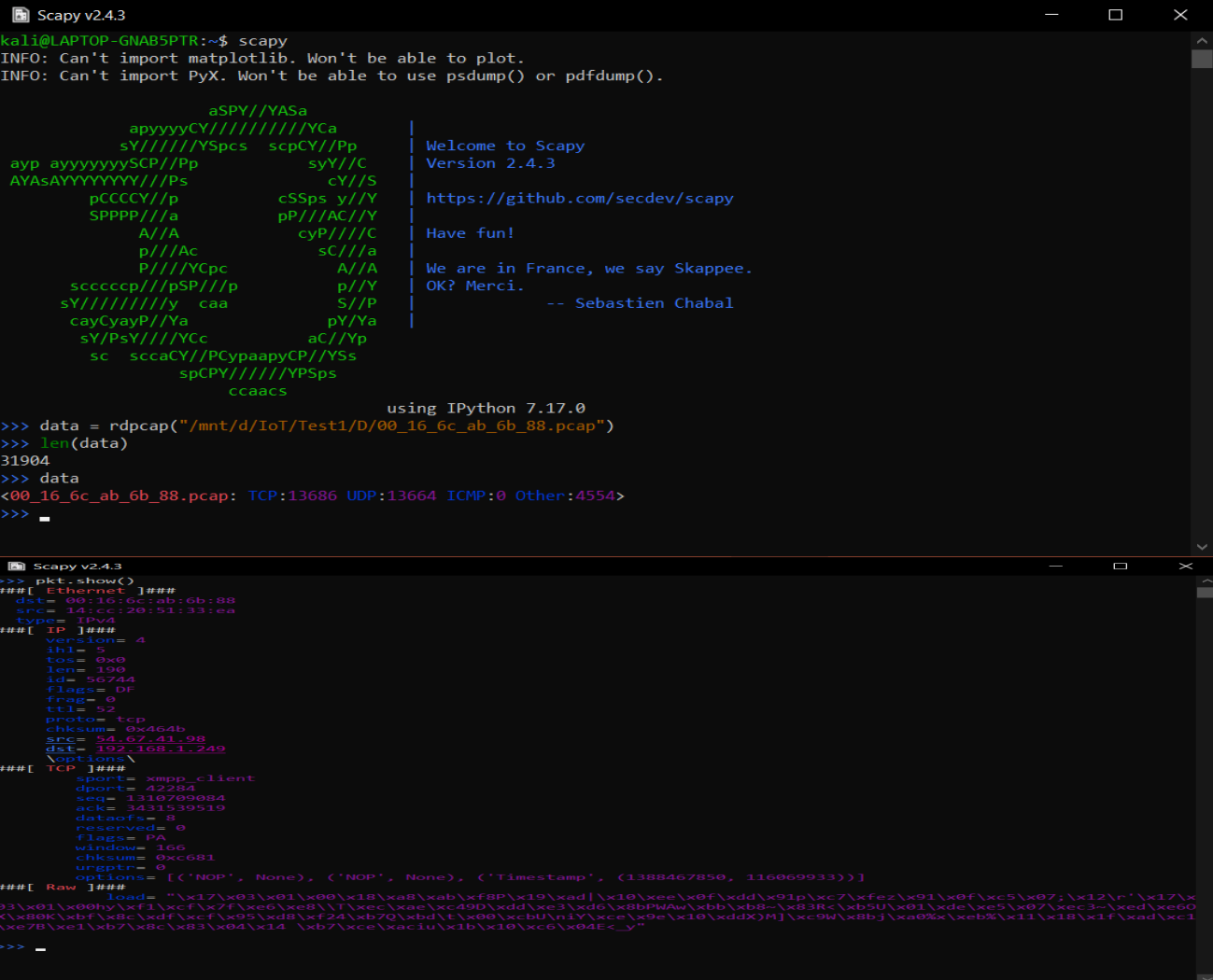
IoT Traffic Flow Construction

- Flow Construction: Construction of two types flows such as TCP & UDP from IoT traffic using scapy & python.

➤ Working with Scapy :

■ Reading PCAP File

- ✓ Open scapy on Terminal
- ✓ Using rdpcap() for reading PCAP file
- ✓ Check Number of packets in PCAP file



```
Scapy v2.4.3
kali@LAPTOP-GNAB5PTR:~$ scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().

      aSPY//YASa
      apyyyyCY/////////YCa
      sY/////////YSpCs  scpCY//Pp
ayp ayyyyyySCP//Pp      syY//C
AYAsAYYYYYYYY//Ps      cY//S
      pCCCCY//p      cSSps y//Y
      SPPPP//a      pP//AC//Y
      A//A      cyP///C
      p///Ac      sC///a
      P///YCpc      A//A
      scccccp///pSP///p      p//Y
      sY/////////y  caa      S//P
      cayCyayP//Ya      pY/Ya
      sY/PsY///YCc      aC//Yp
      sc  sccaCY//PCypaapyCP//YSs
      spCPY/////////YPSps
      ccaacs

Welcome to Scapy
Version 2.4.3

https://github.com/secdev/scapy

Have fun!

We are in France, we say Skappee.
OK? Merci.
-- Sebastien Chabal

using IPython 7.17.0
>>> data = rdpcap("/mnt/d/IoT/Test1/D/00_16_6c_ab_6b_88.pcap")
>>> len(data)
31904
>>> data
<00_16_6c_ab_6b_88.pcap: TCP:13686 UDP:13664 ICMP:0 Other:4554>
>>>

Scapy v2.4.3
>>> pkt.show()
#### Ethernet I####
  dst= 00:16:0c:ab:6b:88
  src= 14:cc:20:51:33:ea
  type= IPv4
#### IP I####
  version= 4
  ihl= 5
  tos= 0x0
  len= 196
  id= 56744
  flags= DF
  frag= 0
  ttl= 52
  proto= tcp
  checksum= 0x464b
  src= 14.07.41.98
  dst= 192.168.1.249
  options=
#### TCP I####
  sport= xmpp_client
  dport= 42284
  seq= 1310709084
  ack= 3431539519
  dataofs= 8
  reserved= 0
  flags= PA
  window= 166
  checksum= 0xc681
  urgptr= 0
  options= [('NOP', None), ('NOP', None), ('Timestamp', (1388467850, 116069933))]
#### Row I####
  load= "\x17\x03\x01\x00\x18\x08\xab\xf8p\x19\xad|\x10\xee\x0f\xdd\x91p\x07\xfez\x91\x0f\x05\x07;\x12\r'\x17\x
3\x01\x09h\xef1\xcf\x2f\x06\x08\x01\x0c\x0c\x0d\x03\x06\x08p\x0a\x0b\x0b\x08\x03R\x0b5U\x01\x0e\x05\x07\x0c3\x0ed\x0e60
\x08K\x0bf\x08\xdf\xcf\x95\x08\x24\x07Q\x0d\x00\x0bU\niY\x0e\x09e\x10\xddX)M]\xc9w\x08bJ\x0a0%\x0eb%\x11\x18\x1f\xad\x0c1
\x07B\x0e1\x0b7\x08c\x03\x04\x14 \x07\x0e\x0aciU\x1b\x10\x0c6\x04E<_y"
>>>
```

IoT Traffic Flow Construction

- Flow Construction: Construction of two types flows such as TCP & UDP from IoT traffic using scapy & python.

➤ Working with Scapy :

- Analyze Single Packet

```
Scapy v2.4.3
>>> pkt.show()
###[ Ethernet ]###
  dst= 00:16:6c:ab:6b:88
  src= 14:cc:20:51:33:ea
  type= IPv4
###[ IP ]###
  version= 4
  ihl= 5
  tos= 0x0
  len= 190
  id= 56744
  flags= DF
  frag= 0
  ttl= 52
  proto= tcp
  checksum= 0x464b
  src= 54.67.41.98
  dst= 192.168.1.249
  \options\
###[ TCP ]###
  sport= xmpp_client
  dport= 42284
  seq= 1310709084
  ack= 3431539519
  dataofs= 8
  reserved= 0
  flags= PA
  window= 166
  checksum= 0xc681
  urgptr= 0
  options= [('NOP', None), ('NOP', None), ('Timestamp', (1388467850, 116069933))]
###[ Raw ]###
  load= "\x17\x03\x01\x00\x18\xa8\xab\xf8P\x19\xad|\x10\xee\x0f\xdd\x91p\xc7\xfez\x91\x0f\xc5\x07;\x12\r'\x17\x03\x01\x00hy\xf1\xcf\x7f\xe6\xe8\\T\xec\xae\xc49D\xdd\xe3\xd6\x8bPWAw\xbb\x8~\x83R<\xb5U\x01\xde\xe5\x07\xec3~\xed\xe60(\x80K\xbf\x8c\xdf\xcf\x95\xd8\xf24\xb7Q\xbd\t\x00\xcbU\niY\xce\x9e\x10\xddX)M]\xc9W\x8bj\xa0%\xeb%\x11\x18\x1f\xad\xc1\xe7B\xe1\xb7\x8c\x83\x04\x14 \xb7\xce\xaciu\x1b\x10\xc6\x04E<_y"
>>> _
```

IoT Traffic Flow Construction

- Flow Construction: Construction of two types flows such as TCP & UDP from IoT traffic using scapy & python.

- Working with Scapy :
 - Construction of Flow

```
Scapy v2.4.3
>>> sessions = data.sessions()
>>> for k,v in sessions.items():
...     print(k,v)
...
TCP 192.168.1.249:42284 > 54.67.41.98:5222 <PacketList: TCP:3491 UDP:0 ICMP:0 Other:0>
TCP 54.67.41.98:5222 > 192.168.1.249:42284 <PacketList: TCP:2253 UDP:0 ICMP:0 Other:0>
ARP 192.168.1.1 > 192.168.1.249 <PacketList: TCP:0 UDP:0 ICMP:0 Other:1173>
ARP 192.168.1.249 > 192.168.1.1 <PacketList: TCP:0 UDP:0 ICMP:0 Other:1173>
UDP 192.168.1.249:44976 > 239.255.255.250:1900 <PacketList: TCP:0 UDP:12159 ICMP:0 Other:0>
IPv6 fe80::216:6cff:feab:6b88 > ff02::1:ffab:6b88 nh=Hop-by-Hop Option Header <PacketList: TCP:0 UDP:0 ICMP:0 Other:414>
IPv6 fe80::216:6cff:feab:6b88 > ff02::fb nh=Hop-by-Hop Option Header <PacketList: TCP:0 UDP:0 ICMP:0 Other:414>
IP 192.168.1.249 > 239.255.255.250 proto=igmp <PacketList: TCP:0 UDP:0 ICMP:0 Other:413>
UDP 192.168.1.249:1900 > 192.168.1.193:3080 <PacketList: TCP:0 UDP:648 ICMP:0 Other:0>
TCP 192.168.1.193:3916 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:3918 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:3916 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:3918 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
IP 192.168.1.249 > 224.0.0.251 proto=igmp <PacketList: TCP:0 UDP:0 ICMP:0 Other:413>
TCP 192.168.1.193:4442 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4442 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4444 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4444 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4451 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4451 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4455 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4455 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4637 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4637 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4640 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4640 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
Ethernet type=888e <PacketList: TCP:0 UDP:0 ICMP:0 Other:86>
TCP 192.168.1.193:4752 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4752 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4754 > 192.168.1.249:49152 <PacketList: TCP:6 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.249:49152 > 192.168.1.193:4754 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
TCP 192.168.1.193:4212 > 192.168.1.249:49152 <PacketList: TCP:5 UDP:0 ICMP:0 Other:0>
```

Feature Extraction from IoT Traffic

- Feature Extraction: Extract the three types of features such as packet level, flow level & behavior level.

Packet Level Attributes	Flow Level Attributes	Behavior Level Attributes
Packet Length	Flow Length	DNS Interval
Packet Source Port No.	Flow Duration	NTP Interval
Packet Destination Port No.	Flow Ratio	Cipher Suites
Packet Payload Length	Flow Payload Length	Domain Names

Training & Testing of IoT Traffic

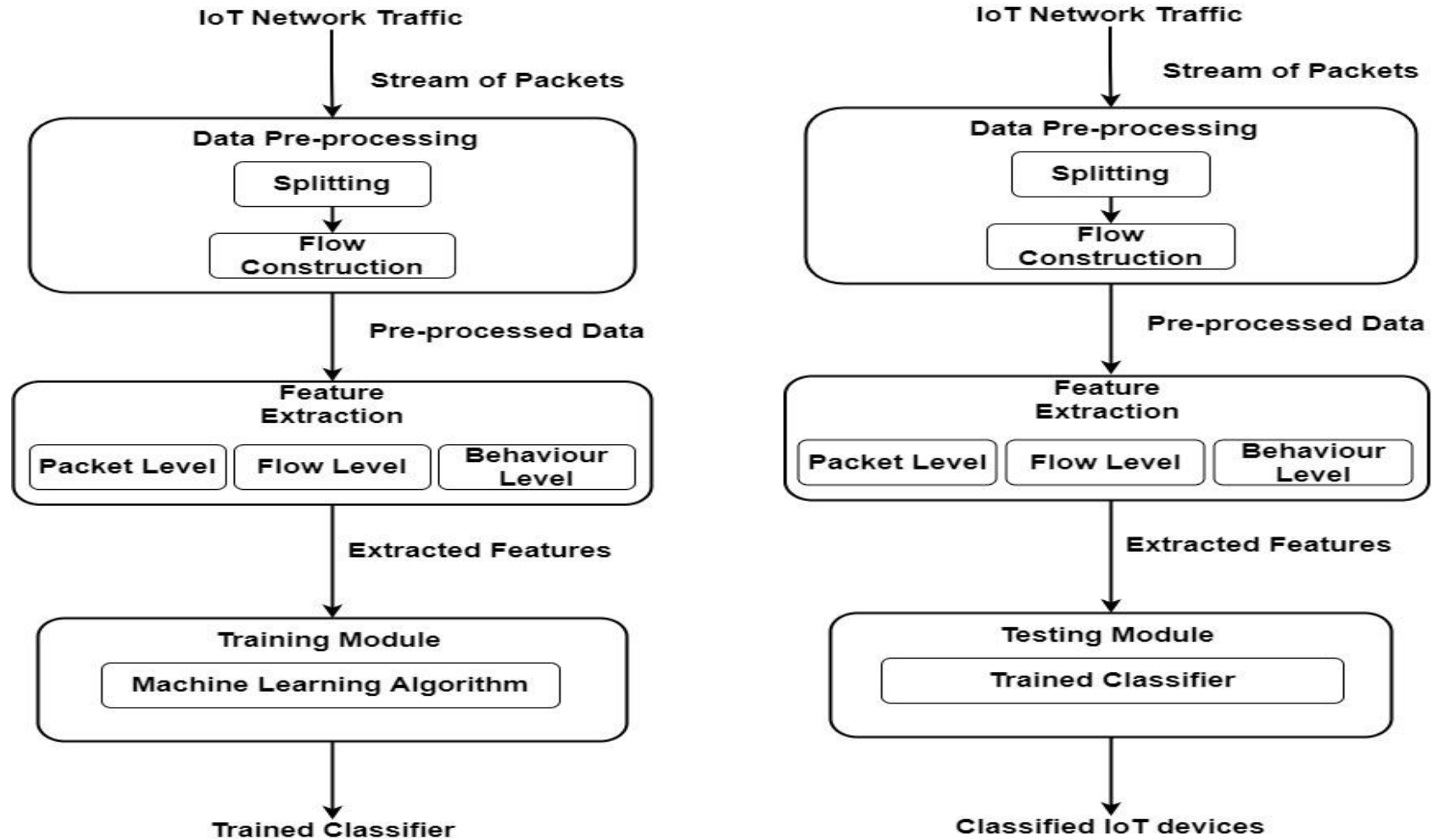


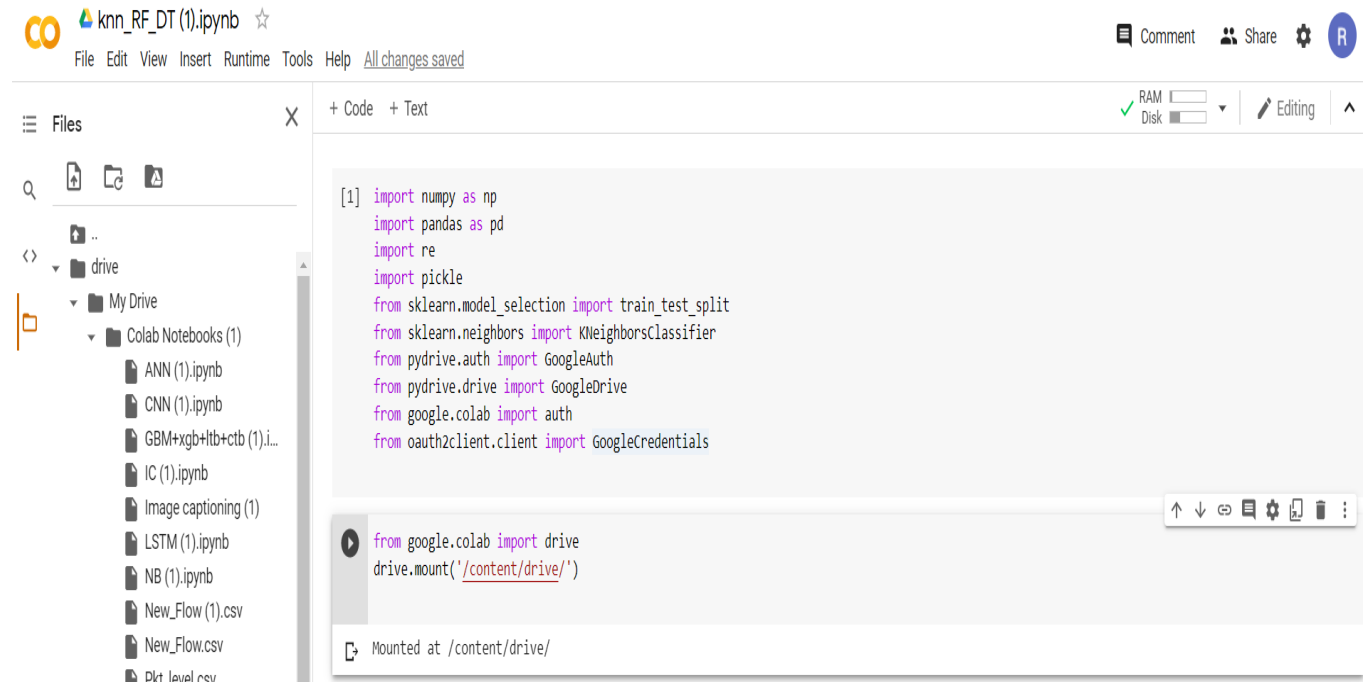
Fig. 10: Training Module & Testing Module

Training & Testing of IoT Traffic

- Training a machine learning classifier simply learning a certain type of patterns from a labeled input IoT traffic.

➤ Steps to Train a Classifier:

- **Using Google Colab**
 - ✓ Free online cloud-based Jupyter notebook environment
 - ✓ To train machine learning and deep learning models on CPUs, GPUs, and TPUs.
- **Importing Packages**
- **Import Google Drive**



```
[1] import numpy as np
import pandas as pd
import re
import pickle
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

from google.colab import drive
drive.mount('/content/drive/')
```

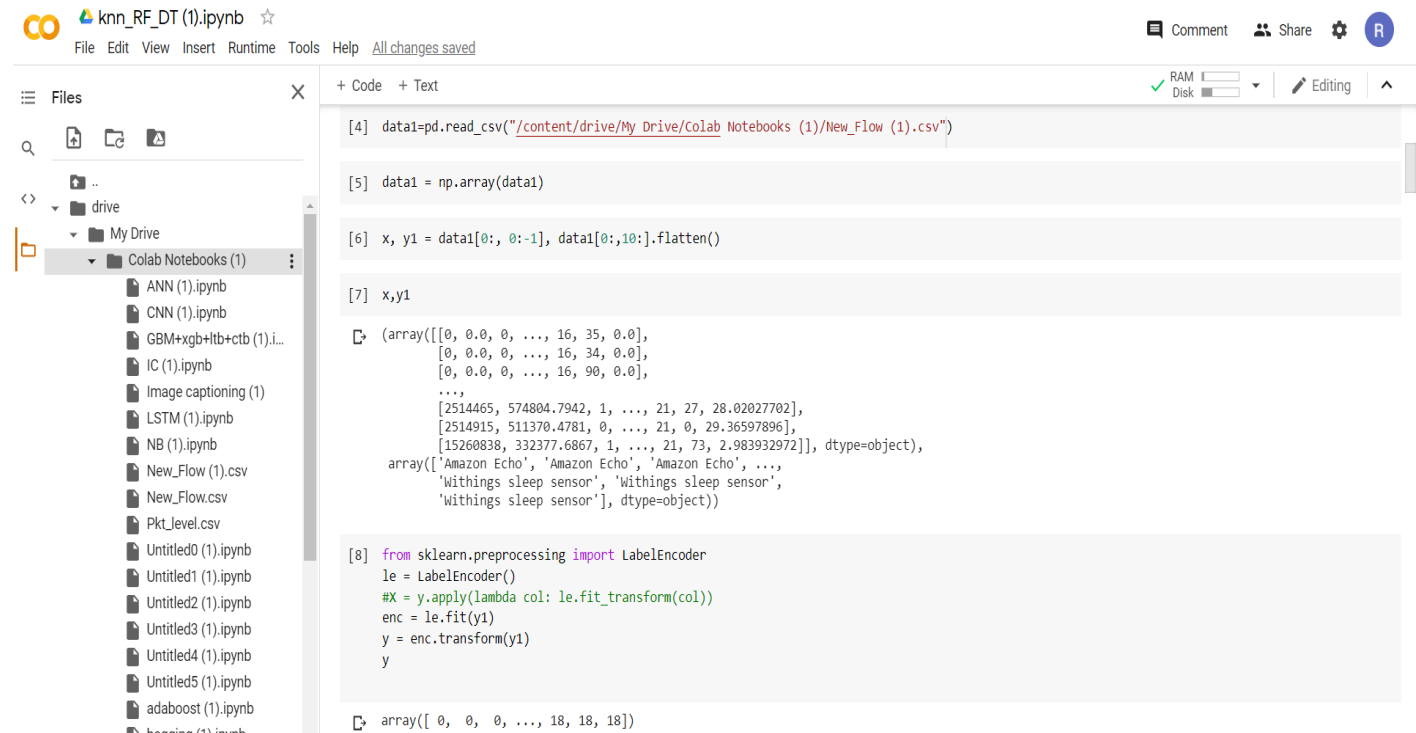
Mounted at /content/drive/

Training & Testing of IoT Traffic

- Training a machine learning classifier simply learning a certain type of patterns from a labeled input IoT traffic.

➤ Steps to Train a Classifier:

- **Using Google Colab**
 - ✓ Free online cloud-based Jupyter notebook environment
 - ✓ To train machine learning and deep learning models on CPUs, GPUs, and TPUs.
- **Importing Packages**
- **Import Google Drive**
- **Data Type Selection and Upload**
- **Data Mapping**
- **Label Encoding (If needed)**



The screenshot displays a Google Colab notebook interface. The top bar shows the notebook title 'knn_RF_DT (1).ipynb' and various menu options like File, Edit, View, Insert, Runtime, Tools, and Help. A status bar on the right indicates 'All changes saved' and shows RAM and Disk usage. The left sidebar shows a file explorer with a directory structure including 'drive' and 'My Drive'. The main code area contains the following Python code:

```
[4] data1=pd.read_csv("/content/drive/My Drive/Colab Notebooks (1)/New_Flow (1).csv")

[5] data1 = np.array(data1)

[6] x, y1 = data1[0:, 0:-1], data1[0:,10:].flatten()

[7] x,y1

[8] (array([[0, 0.0, 0, ..., 16, 35, 0.0],
          [0, 0.0, 0, ..., 16, 34, 0.0],
          [0, 0.0, 0, ..., 16, 90, 0.0],
          ...,
          [2514465, 574804.7942, 1, ..., 21, 27, 28.02027702],
          [2514915, 511370.4781, 0, ..., 21, 0, 29.36597896],
          [15260838, 332377.6867, 1, ..., 21, 73, 2.983932972]]], dtype=object),
  array(['Amazon Echo', 'Amazon Echo', 'Amazon Echo', ...,
        'Withings sleep sensor', 'Withings sleep sensor',
        'Withings sleep sensor'], dtype=object))

[8] from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#X = y.apply(lambda col: le.fit_transform(col))
enc = le.fit(y1)
y = enc.transform(y1)
y

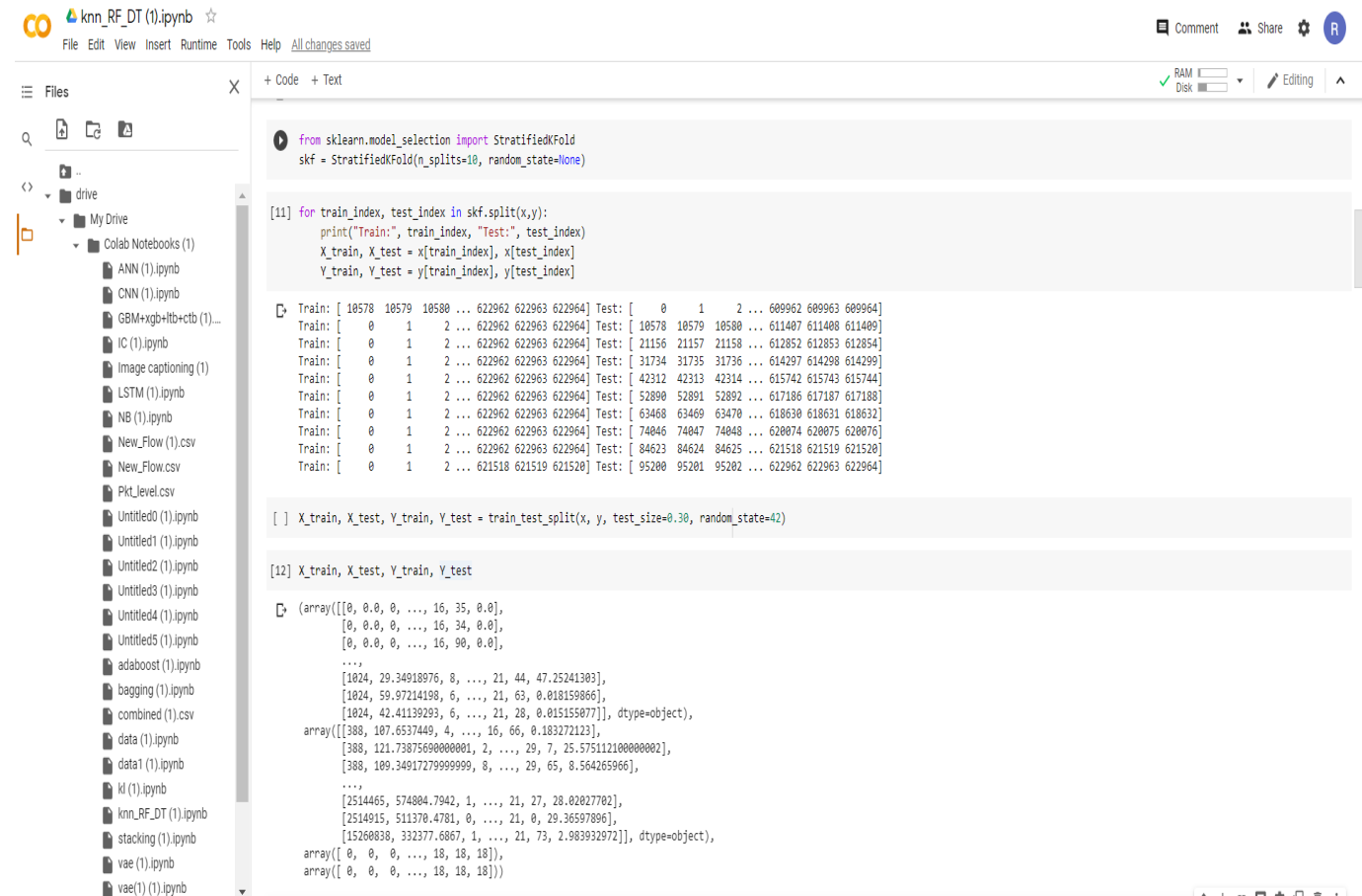
array([ 0,  0,  0, ..., 18, 18, 18])
```

Training & Testing of IoT Traffic

- Training a machine learning classifier simply learning a certain type of patterns from a labeled input IoT traffic.

➤ Steps to Train a Classifier:

- Using Google Colab
 - ✓ Free online cloud-based Jupyter notebook environment
 - ✓ To train machine learning and deep learning models on CPUs, GPUs, and TPUs.
- Importing Packages
- Import Google Drive
- Data Type Selection and Upload
- Data Mapping
- Label Encoding (If needed)
- K-fold Cross validation or Splitting of Data (Choice)



```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10, random_state=None)

[11] for train_index, test_index in skf.split(x,y):
    print("Train:", train_index, "Test:", test_index)
    X_train, X_test = x[train_index], x[test_index]
    Y_train, Y_test = y[train_index], y[test_index]

Train: [ 10578 10579 10580 ... 622962 622963 622964] Test: [ 0 1 2 ... 609962 609963 609964]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 10578 10579 10580 ... 611407 611408 611409]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 21156 21157 21158 ... 612852 612853 612854]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 31734 31735 31736 ... 614297 614298 614299]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 42312 42313 42314 ... 615742 615743 615744]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 52890 52891 52892 ... 617186 617187 617188]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 63468 63469 63470 ... 618630 618631 618632]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 74046 74047 74048 ... 620074 620075 620076]
Train: [ 0 1 2 ... 622962 622963 622964] Test: [ 84623 84624 84625 ... 621518 621519 621520]
Train: [ 0 1 2 ... 621518 621519 621520] Test: [ 95200 95201 95202 ... 622962 622963 622964]

[ ] X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.30, random_state=42)

[12] X_train, X_test, Y_train, Y_test

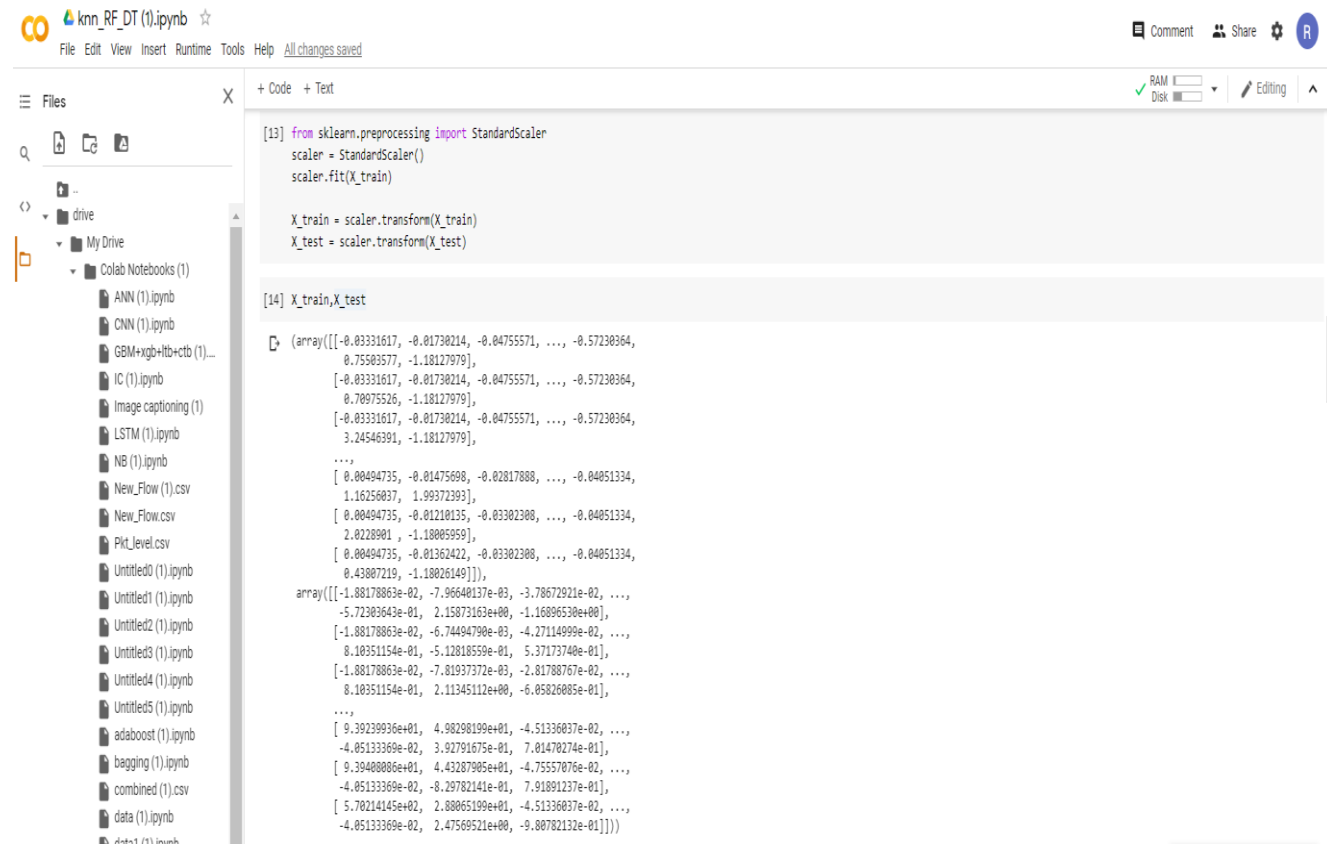
array([[0, 0, 0, ..., 16, 35, 0, 0],
       [0, 0, 0, ..., 16, 34, 0, 0],
       [0, 0, 0, ..., 16, 90, 0, 0],
       ...,
       [1024, 29.34918976, 8, ..., 21, 44, 47.25241303],
       [1024, 59.97214198, 6, ..., 21, 63, 0.018159866],
       [1024, 42.41139293, 6, ..., 21, 28, 0.015155077]], dtype=object),
array([[388, 107.6537440, 4, ..., 16, 66, 0.183272123],
       [388, 121.738756900000001, 2, ..., 29, 7, 25.575112100000002],
       [388, 109.34917279999999, 8, ..., 29, 65, 8.564265966],
       ...,
       [2514465, 574084.7942, 1, ..., 21, 27, 28.02027702],
       [2514915, 511370.4781, 0, ..., 21, 0, 29.36597896],
       [15260838, 332377.6867, 1, ..., 21, 73, 2.983932972]], dtype=object),
array([ 0, 0, 0, ..., 18, 18, 18]),
array([ 0, 0, 0, ..., 18, 18, 18])
```

Training & Testing of IoT Traffic

- Training a machine learning classifier simply learning a certain type of patterns from a labeled input IoT traffic.

➤ Steps to Train a Classifier:

- Using Google Colab
 - ✓ Free online cloud-based Jupyter notebook environment
 - ✓ To train machine learning and deep learning models on CPUs, GPUs, and TPUs.
- Importing Packages
- Import Google Drive
- Data Type Selection and Upload
- Data Mapping
- Label Encoding (If needed)
- K-fold Cross validation or Splitting of Data (Choice)
- Normalization of Data



```
[13] from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

[14] X_train, X_test

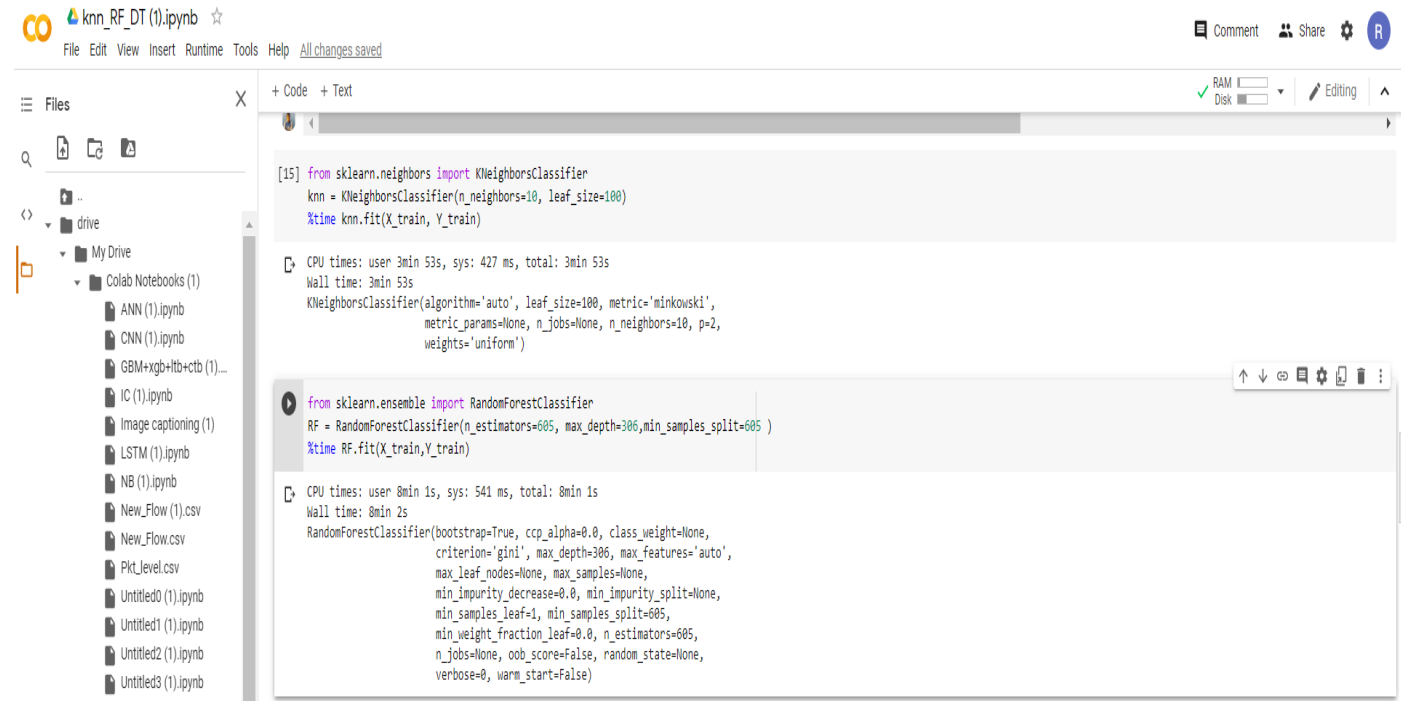
Out[14]: (array([[ -0.03331617, -0.01730214, -0.04755571, ..., -0.57230364,
 0.75503577, -1.18127979],
 [ -0.03331617, -0.01730214, -0.04755571, ..., -0.57230364,
 0.70975526, -1.18127979],
 [ -0.03331617, -0.01730214, -0.04755571, ..., -0.57230364,
 3.24546391, -1.18127979],
 ...,
 [ 0.00494735, -0.01475698, -0.02817888, ..., -0.04051334,
 1.16256037, 1.99372393],
 [ 0.00494735, -0.01210135, -0.03302308, ..., -0.04051334,
 2.0220901, -1.10005959],
 [ 0.00494735, -0.01362422, -0.03302308, ..., -0.04051334,
 0.43807219, -1.18026149]]),
 array([[ -1.88178863e-02, -7.96640137e-03, -3.78672921e-02, ...,
 -5.72303643e-01, 2.15873163e+00, -1.16096530e+00],
 [ -1.88178863e-02, -6.74494790e-03, -4.27114099e-02, ...,
 8.10351154e-01, -5.12810559e-01, 5.37173740e-01],
 [ -1.88178863e-02, -7.01937372e-03, -2.81788767e-02, ...,
 8.10351154e-01, 2.11345112e+00, -6.05826085e-01],
 ...,
 [ 9.39239936e+01, 4.90290199e+01, -4.51336037e-02, ...,
 -4.05133369e-02, 3.92791675e-01, 7.01470274e-01],
 [ 9.39400006e+01, 4.43207905e+01, -4.75557076e-02, ...,
 -4.05133369e-02, -8.29782141e-01, 7.91091237e-01],
 [ 5.70214145e+02, 2.80865199e+01, -4.51336037e-02, ...,
 -4.05133369e-02, 2.47569521e+00, -9.80782131e-01]]))
```

Training & Testing of IoT Traffic

- Training a machine learning classifier simply learning a certain type of patterns from a labeled input IoT traffic.

➤ Steps to Train a Classifier:

- **Using Google Colab**
 - ✓ Free online cloud-based Jupyter notebook environment
 - ✓ To train machine learning and deep learning models on CPUs, GPUs, and TPUs.
- **Importing Packages**
- **Import Google Drive**
- **Data Type Selection and Upload**
- **Data Mapping**
- **Label Encoding (If needed)**
- **K-fold Cross validation or Splitting of Data (Choice)**
- **Normalization of Data**
- **Training of Classifier**



```
[15] from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10, leaf_size=100)
%time knn.fit(X_train, Y_train)

CPU times: user 3min 53s, sys: 427 ms, total: 3min 53s
Wall time: 3min 53s
KNeighborsClassifier(algorithm='auto', leaf_size=100, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                    weights='uniform')
```

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators=605, max_depth=306, min_samples_split=605 )
%time RF.fit(X_train, Y_train)

CPU times: user 8min 1s, sys: 541 ms, total: 8min 1s
Wall time: 8min 2s
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=306, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=605,
                       min_weight_fraction_leaf=0.0, n_estimators=605,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

Training & Testing of IoT Traffic

- Training a machine learning classifier simply learning a certain type of patterns from a labeled input IoT traffic.

➤ Steps to Train a Classifier:

- Using Google Colab
 - ✓ Free online cloud-based Jupyter notebook environment
 - ✓ To train machine learning and deep learning models on CPUs, GPUs, and TPUs.
- Importing Packages
- Import Google Drive
- Data Type Selection and Upload
- Data Mapping
- Label Encoding (If needed)
- K-fold Cross validation or Splitting of Data (Choice)
- Normalization of Data
- Training of Classifier

```
knn_RF_DT (1).ipynb
File Edit View Insert Runtime Tools Help All changes saved

Files
-
drive
  My Drive
    Colab Notebooks (1)
      ANN (1).ipynb
      CNN (1).ipynb
      GBM+rgb+hb+ctb (1)...
      IC (1).ipynb
      Image captioning (1)
      LSTM (1).ipynb
      NB (1).ipynb
      New_Flow (1).csv
      New_Flow.csv
      Pkt_Level.csv
      Untitled0 (1).ipynb
      Untitled1 (1).ipynb
      Untitled2 (1).ipynb
      Untitled3 (1).ipynb

+ Code + Text
[15] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors=10, leaf_size=100)
     %time knn.fit(X_train, Y_train)

CPU times: user 3min 53s, sys: 427 ms, total: 3min 53s
Wall time: 3min 53s
KNeighborsClassifier(algorithm='auto', leaf_size=100, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=10, p=2,
                    weights='uniform')

from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators=605, max_depth=306, min_samples_split=605)
%time RF.fit(X_train, Y_train)

CPU times: user 8min 1s, sys: 541 ms, total: 8min 1s
Wall time: 8min 2s
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                      criterion='gini', max_depth=306, max_features='auto',
                      max_leaf_nodes=None, max_samples=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=605,
                      min_weight_fraction_leaf=0.0, n_estimators=605,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

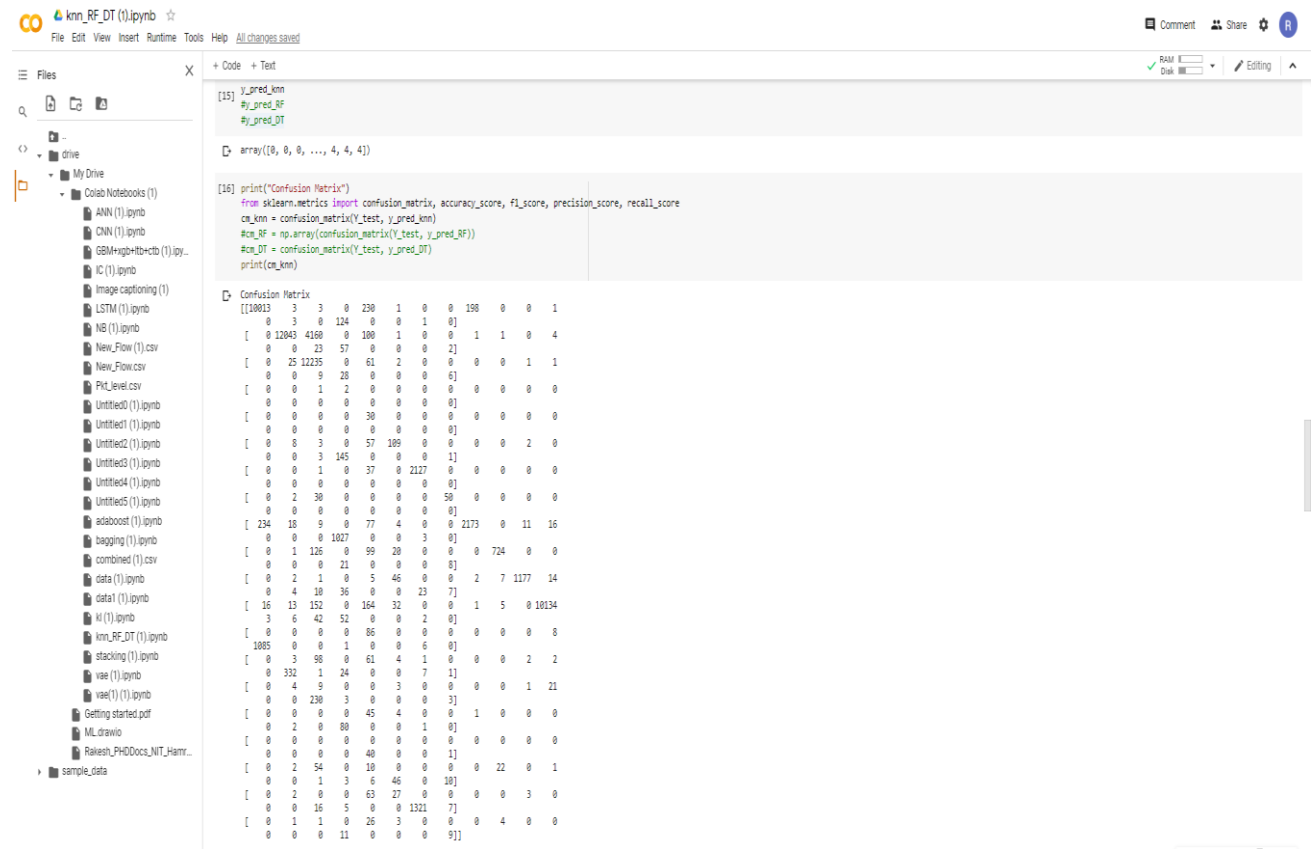
Training & Testing of IoT Traffic

- Testing a machine learning classifier for algorithmic correctness and assuring the quality of newly build model.

➤ Steps to Testing a Classifier:

- Testing Classifier with test dataset
- Making Confusion Matrix

✓ Easy way to measure the performance of Classifier



The screenshot shows a Jupyter Notebook titled 'knn_RF_DT (1).ipynb'. The left sidebar displays a file explorer with various notebooks and data files. The main area contains the following code:

```
[15]: y_pred_knn
      #y_pred_RF
      #y_pred_DT

array([0, 0, 0, ..., 4, 4, 4])

[16]: print("Confusion Matrix")
      from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
      cm_knn = confusion_matrix(Y_test, y_pred_knn)
      #cm_RF = np.array(confusion_matrix(Y_test, y_pred_RF))
      #cm_DT = confusion_matrix(Y_test, y_pred_DT)
      print(cm_knn)
```

The output of the code is a Confusion Matrix for the knn classifier:

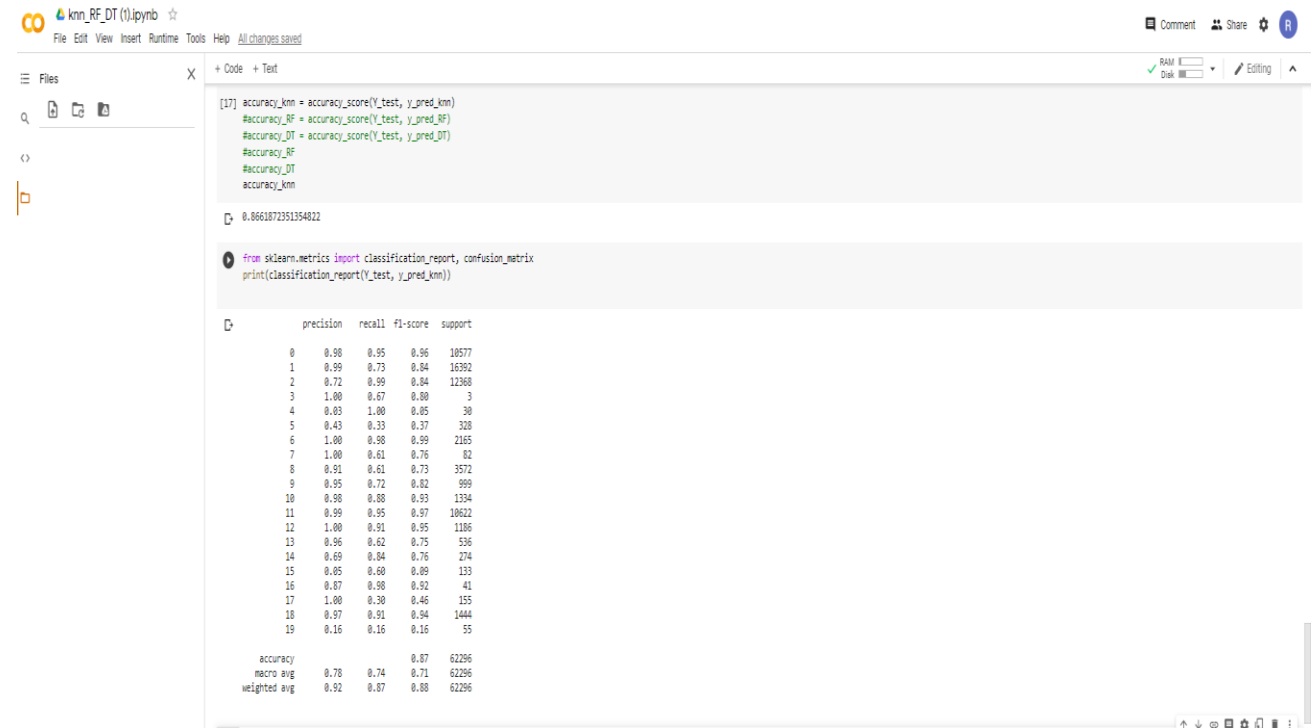
```
Confusion Matrix
[[10013  3  3  0  230  1  0  0  198  0  0  1
  0  3  0 124  0  0  1  0]
 [ 0 12043 4160  0 100  1  0  0  1  1  0  4
  0  0  23  57  0  0  0  2]
 [ 0  25 12235  0  61  2  0  0  0  0  1  1
  0  0  9  28  0  0  0  6]
 [ 0  0  1  2  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  30  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0]
 [ 0  0  3  0  57 100  0  0  0  0  2  0
  0  0  3 145  0  0  0  1]
 [ 0  0  1  0  37  0 2127  0  0  0  0  0
  0  0  0  0  0  0  0  0]
 [ 0  2  30  0  0  0  0  50  0  0  0  0
  0  0  0  0  0  0  0  0]
 [ 234 10  9  0  77  4  0  0 2173  0 11 16
  0  0  0 1027  0  0  3  0]
 [ 0  1 126  0  99 20  0  0  0 724  0  0
  0  0  0  21  0  0  0  0]
 [ 0  2  1  0  5  46  0  0  2  7 1177 14
  0  4 10  36  0  0 23  7]
 [ 16 13 152  0 164 32  0  0  1  5  0 10134
  3  6  42  52  0  0  2  0]
 [ 0  0  0  0  86  0  0  0  0  0  0  0
 1005  0  1  0  0  6  0]
 [ 0  3  98  0  61  4  1  0  0  0  2  2
 0 332  1  24  0  0  7 1]
 [ 0  4  9  0  0  3  0  0  0  0  1  21
 0  0 230  3  0  0  0  3]
 [ 0  0  0  0  45  4  0  0  1  0  0  0
 0  2  0  80  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  40  0  0 1]
 [ 0  2  54  0 10  0  0  0  0 22  0  1
 0  0  1  3  6  46  0 10]
 [ 0  2  0  0  63 27  0  0  0  0  3  0
 0  0 16  5  0  0 1321 7]
 [ 0  1  1  0 26  3  0  0  0  4  0  0
 0  0  0 11  0  0  0 9]]
```


Training & Testing of IoT Traffic

- Testing a machine learning classifier for algorithmic correctness and assuring the quality of newly build model.

➤ Steps to Testing a Classifier:

- Testing Classifier with test dataset
- Making Confusion Matrix
 - ✓ Easy way to measure the performance of Classifier
- Accuracy
- Evaluation Metrics
 - ✓ Precision
 - ✓ Recall
 - ✓ F1 Score



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[17]: accuracy_knn = accuracy_score(Y_test, y_pred_knn)
      #accuracy_RF = accuracy_score(Y_test, y_pred_RF)
      #accuracy_DT = accuracy_score(Y_test, y_pred_DT)
      #accuracy_RF
      #accuracy_DT
      accuracy_knn
```

Output: 0.8661872351354822

```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(Y_test, y_pred_knn))
```

	precision	recall	f1-score	support
0	0.98	0.95	0.96	10577
1	0.99	0.73	0.84	16392
2	0.72	0.99	0.84	12368
3	1.00	0.67	0.80	3
4	0.03	1.00	0.05	30
5	0.43	0.33	0.37	308
6	1.00	0.98	0.99	2165
7	1.00	0.61	0.76	82
8	0.91	0.61	0.73	3572
9	0.95	0.72	0.82	999
10	0.98	0.88	0.93	1334
11	0.99	0.95	0.97	10622
12	1.00	0.91	0.95	1186
13	0.96	0.62	0.75	536
14	0.69	0.84	0.76	274
15	0.05	0.88	0.09	133
16	0.67	0.98	0.82	41
17	1.00	0.30	0.46	355
18	0.97	0.91	0.94	1444
19	0.16	0.16	0.16	55
accuracy			0.87	62296
macro avg	0.78	0.74	0.71	62296
weighted avg	0.92	0.87	0.88	62296

Live Testing Procedure

We first train the DT classifier with 24 hours data.

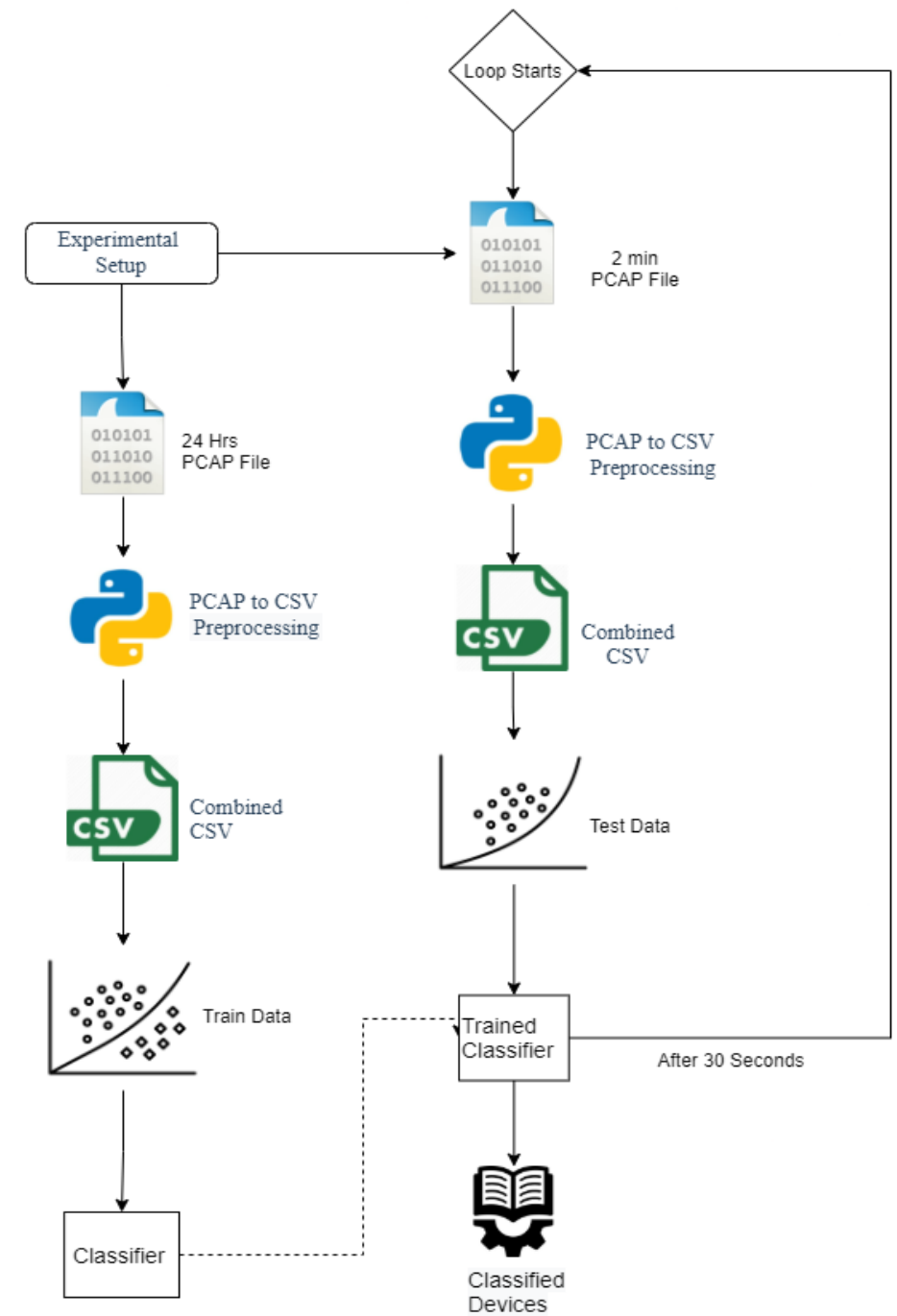
Then a loop starts that captures a PCAP file, comprised of the data of the last 2 minutes.

This PCAP file is processed into a CSV.

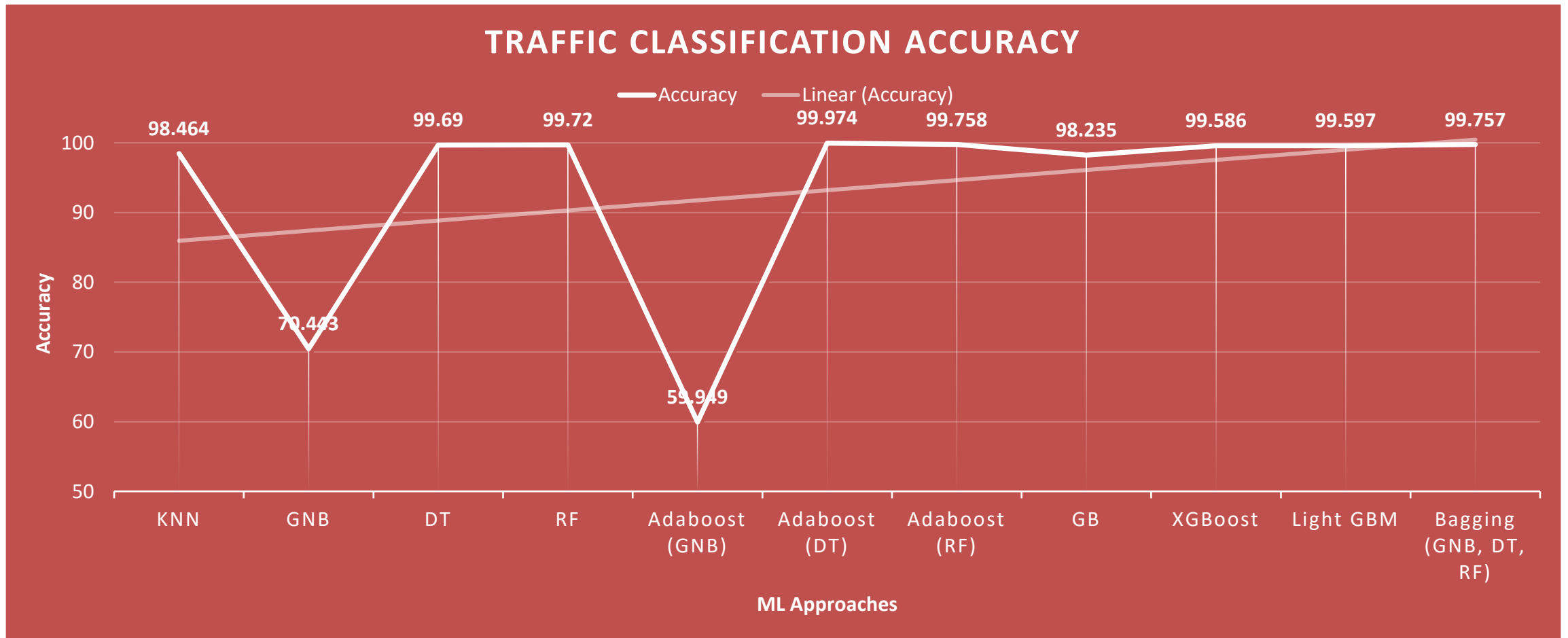
The CSV is used as a Test Data in the trained DT classifier.

Results are obtained and the loop starts again, after every 30 seconds.

Live Testing Flowchart



Trained Model Accuracy for Device classification



Existing Datasets

- A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," in *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745-1759, 1 Aug. 2019, doi: 10.1109/TMC.2018.2866249.
- Dataset Link: <https://iotanalytics.unsw.edu.au/>



Attacks on IoTs

SYN Flood Attack

- A SYN flood (half-open attack) is a type of **denial-of-service (DDoS) attack** which aims to make a server unavailable to legitimate traffic by consuming all available server resources.
- By repeatedly sending initial connection request (SYN) packets, the attacker is able to overwhelm all available ports on a targeted server machine, causing the targeted device to respond to legitimate traffic sluggishly or not at all.

Steps of SYN Flood

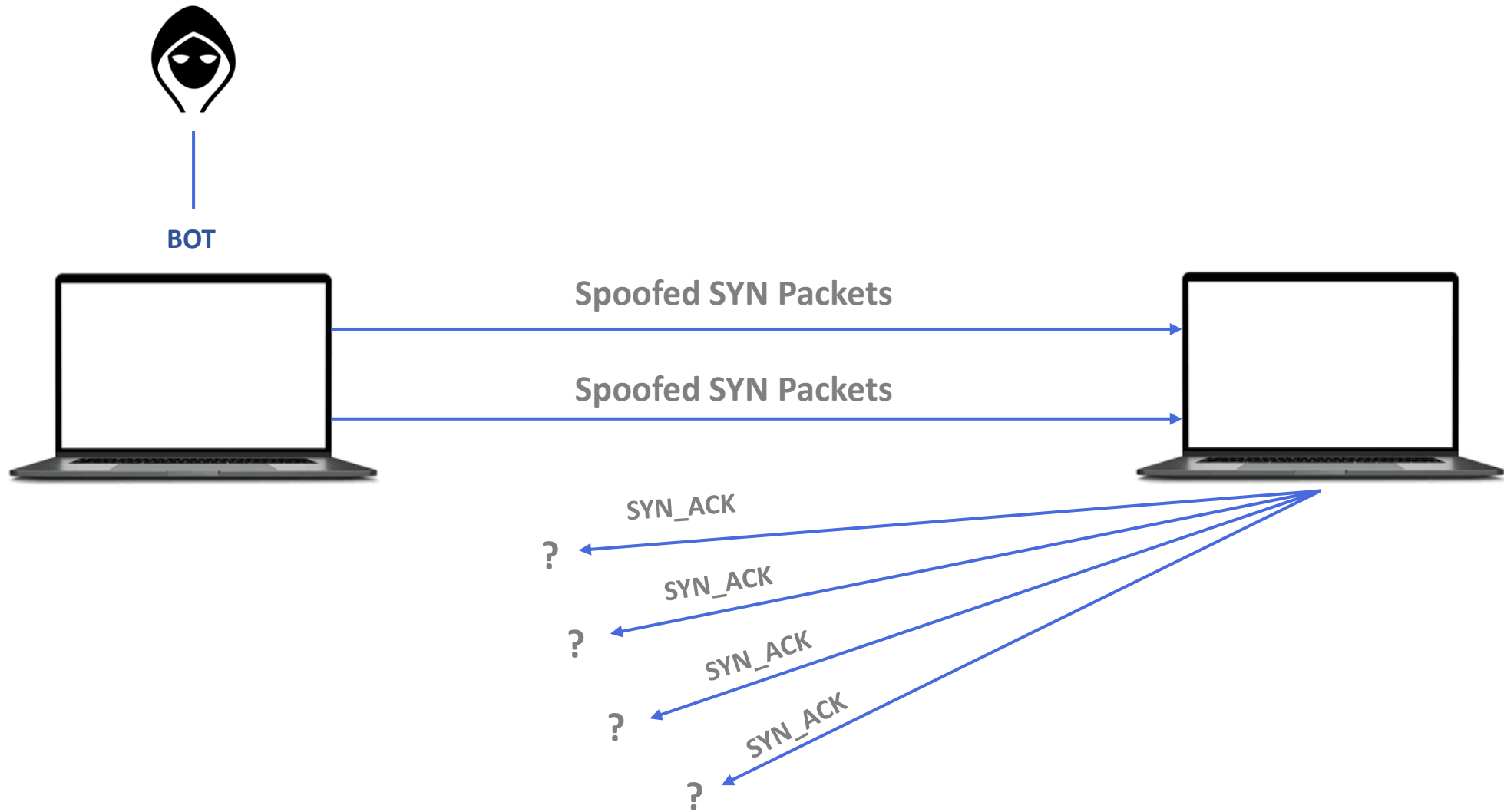
- SYN flood attacks work by exploiting the handshake process of a **TCP** connection. Under normal conditions, TCP connection exhibits three distinct processes in order to make a connection.
 - First, the client sends a SYN packet to the server in order to initiate the connection.
 - The server then responds to that initial packet with a SYN/ACK packet, in order to acknowledge the communication.
 - Finally, the client returns an ACK packet to acknowledge the receipt of the packet from the server. After completing this sequence of packet sending and receiving, the TCP connection is open and able to send and receive data.

Three Way Handshaking (TCP)



DoS SYN Flood Working

- The attacker sends a high volume of SYN packets to the targeted server, often with **spoofed** IP addresses.
- The server then responds to each one of the connection requests and leaves an open port ready to receive the response.
- While the server waits for the final ACK packet, which never arrives, the attacker continues to send more SYN packets. The arrival of each new SYN packet causes the server to temporarily maintain a new open port connection for a certain length of time, and once all the available ports have been utilized the server is unable to function normally.



ARP Protocol

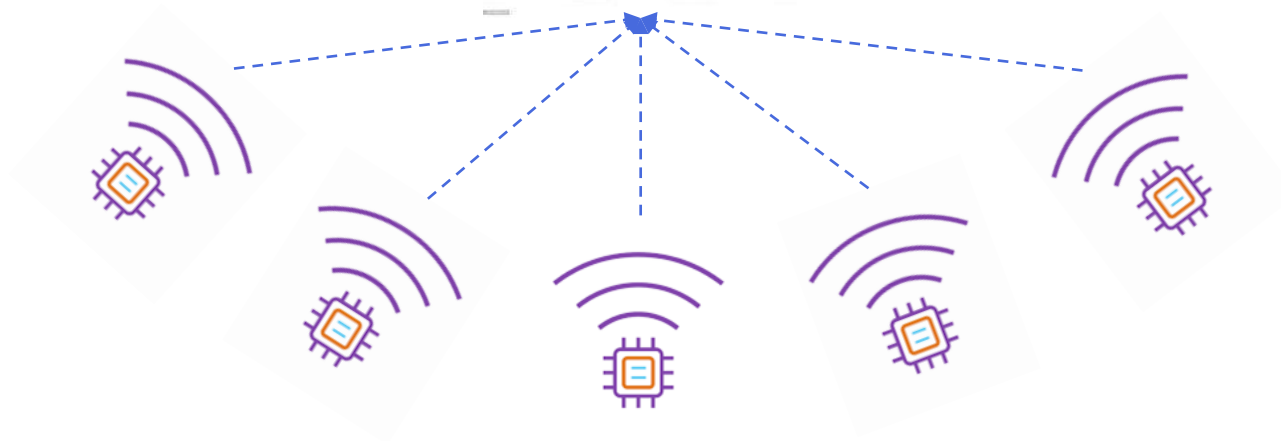
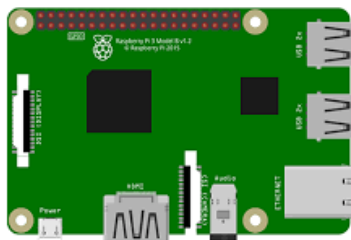
- Address Resolution Protocol (ARP) is a protocol that enables network communications to reach a specific device on the network.
- ARP translates Internet Protocol (IP) addresses to a Media Access Control (MAC) address, and vice versa.
- Most commonly, devices use ARP to contact the router or gateway that enables them to connect to the Internet.
- Hosts maintain an ARP cache, a mapping table between IP addresses and MAC addresses, and use it to connect to destinations on the network. If the host doesn't know the MAC address for a certain IP address, it sends out an ARP request packet, asking other machines on the network for the matching MAC address.

ARP Spoofing

- ARP Spoofing also known as **ARP Poisoning**, is a **Man in the Middle Attack (MitM)** that allows attackers to intercept communication between network devices.
- The two devices update their ARP cache entries and from that point onwards, communicate with the attacker instead of directly with each other.

Working

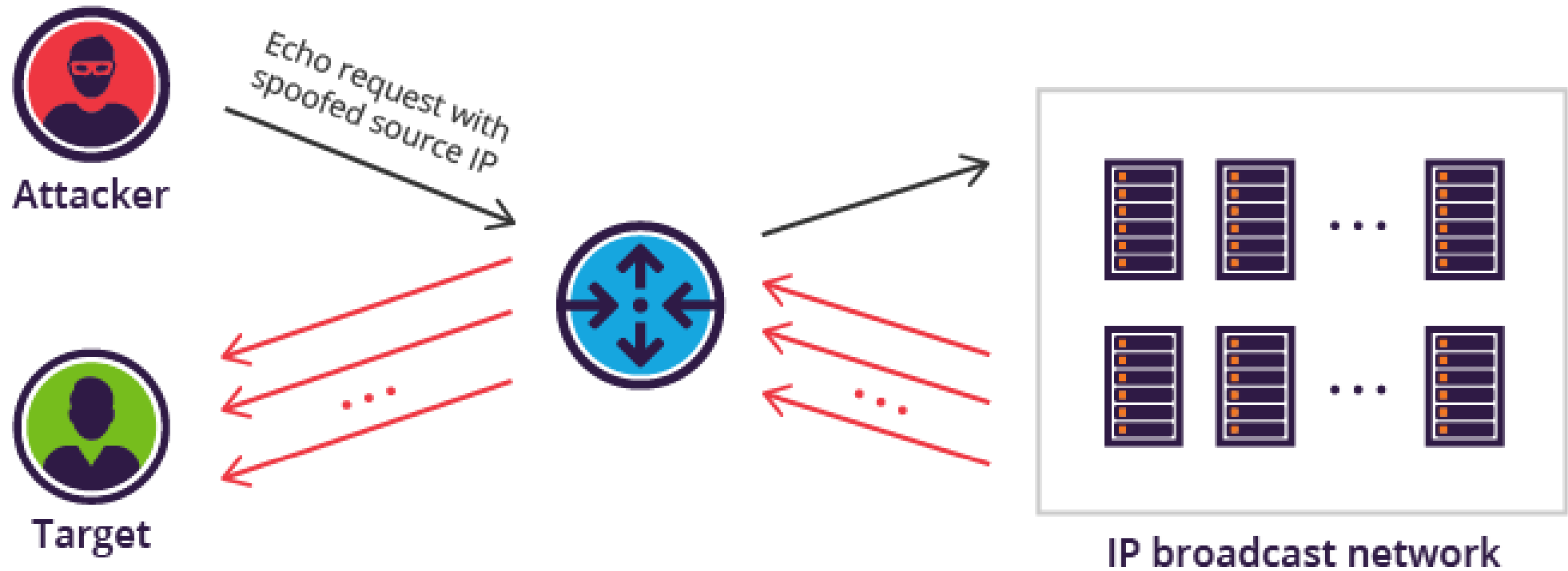
- Must have access to the network.
- Scanning the network to determine the IP addresses of connected device network.
- Attacker uses spoofing tool (i.e. Arpspoof) to forged ARP responses.
- The forged responses advertise that the correct MAC address for both IP addresses, belonging to the router and workstation, is the attacker's MAC address. This fools both router and workstation to connect to the attacker's machine, instead of to each other.
- The two devices update their ARP cache entries and from that point onwards, communicate with the attacker instead of directly with each other.
- The attacker is now secretly in the middle of all communications.



Smurf Attack

- It is a **distributed denial-of-service attack** in which large numbers of **Internet Control Message Protocol (ICMP)** packets with the intended victim's spoofed source IP are broadcast to a computer network using an IP broadcast address.
- Most devices on a network will, by default, respond to this by sending a reply to the source IP address.
- If the number of machines on the network that receive and respond to these packets is very large, the victim's computer will be flooded with traffic.
- This can slow down the victim's computer to the point where it becomes impossible to work on.

Working



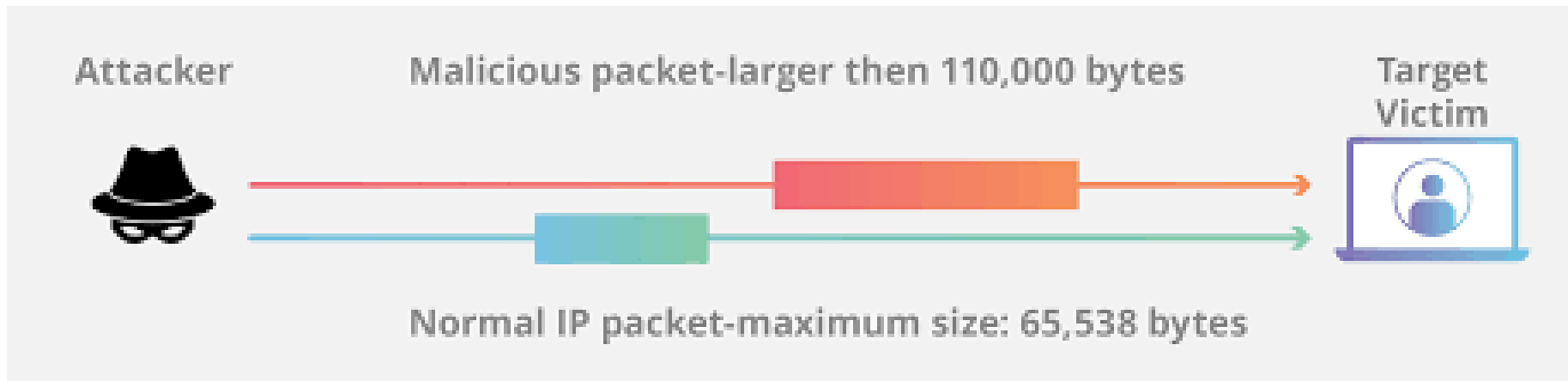
Ping of Death

- A Ping of Death attack is a **denial-of-service (DoS)** attack, in which the attacker aims to disrupt a targeted machine by sending a packet larger than the maximum allowable size, causing the target machine to freeze or crash.
- The original Ping of Death attack is less common today. A related attack known as an **ICMP flood attack** is more prevalent.
- An **Internet Control Message Protocol (ICMP)** echo-reply message or “ping”, is a network utility used to test a network connection, and it works much like sonar – a “pulse” is sent out and the “echo” from that pulse tells the operator information about the environment.

Working

- If the connection is working, the source machine receives a reply from the targeted machine.
- While some ping packets are very small, IP4 ping packets are much larger, and can be as large as the maximum allowable packet size of 65,535 bytes.
- Some **TCP/IP** systems were never designed to handle packets larger than the maximum, making them vulnerable to packets above that size.

Working



References

1. Bai, Lei, Lina Yao, Salil S. Kanhere, Xianzhi Wang, and Zheng Yang. "Automatic device classification from network traffic streams of internet of things." in Proceedings of the 43rd International Conference on Local Computer Networks (LCN'18), 2018, pp. 1-9.
2. Yao, Haipeng, Pengcheng Gao, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Zhu Han. "Capsule Network Assisted IoT Traffic Classification Mechanism for Smart Cities." IEEE Internet of Things Journal, vol. 6, pp. 7515-7525, 2019.
3. N. Chaabouni, M. Mosbah, A. Zemmari, C. Sauvignac, and P. Faruki, "Network intrusion detection for iot security based on learning techniques," IEEE Communications Surveys & Tutorials, vol. 21, pp. 2671– 2701, 2019.
4. Koliass, Constantinos, et al. "DDoS in the IoT: Mirai and other botnets." Computer 50.7 (2017): 80-84.
5. Sivanathan, Arunan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. "Classifying IoT devices in smart environments using network traffic characteristics." IEEE Transactions on Mobile Computing, vol. 18, pp. 1745-1759, 2019.
6. M. Frustaci, P. Pace, G. Aloia, and G. Fortino, "Evaluating critical security issues of the iot world: Present and future challenges," IEEE Internet of Things Journal, vol. 5, pp. 2483–2495, 2017.
7. Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," IEEE Internet of Things Journal, vol. 4, pp. 1250–1258, 2017.
8. Q. Yan, W. Huang, X. Luo, Q. Gong, and F. R. Yu, "A multi-level ddos mitigation framework for the industrial internet of things," IEEE Communications Magazine, vol. 56, pp. 30–36, 2018.

References

9. I. Makhdoom, M. Abolhasan, J. Lipman, R. P. Liu, and W. Ni, “Anatomy of threats to the internet of things,” *IEEE Communications Surveys & Tutorials*, vol. 21, pp. 1636–1675, 2018.
10. J. Granjal, E. Monteiro, and J. S. Silva, “Security for the internet of things: a survey of existing protocols and open research issues,” *IEEE Communications Surveys & Tutorials*, vol. 17, pp. 1294–1312, 2015.
11. N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, and N. Ghani, “Demystifying iot security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations,” *IEEE Communications Surveys & Tutorials*, pp. 2702–2733, 2019.
12. S. Murali and A. Jamalipour, “A lightweight intrusion detection for sybil attack under mobile rpl in the internet of things,” *IEEE Internet of Things Journal (Early Access)*, vol. 6.
13. <https://blackarch.org/spoof.html>
14. <https://www.networkworld.com/article/2272520/six-worst-internet-routing-attacks.html>
15. <https://www.infosecurity-magazine.com/news/massive-bruteforce-attack-on/>
16. <https://canadiandimension.com/articles/view/web-exclusive-author-calls-on-ccla-board-members-to-repudiate-attack-on-dis>
17. S. Li, L. Da Xu, and S. Zhao, “The internet of things: a survey,” *Information Systems Frontiers*, vol. 17, pp. 243–259, 2015.
18. <https://posts.specterops.io/cve-2018-8414-a-case-study-in-responsible-disclosure-ff74c39615ba>
19. <https://www.cvedetails.com/cve/CVE-2019-0735/>

References

20. A. Mosenia and N. K. Jha, “A comprehensive study of security of internet-of-things,” IEEE Transactions on Emerging Topics in Computing, vol. 5, pp. 586–602, 2016
21. <https://www.kaspersky.com/blog/five-most-notorious-cyberattacks/24506/>
22. <https://medium.com/ledger-on-security-and-blockchain/introducing-rainbow-donjons-side-channel-analysis-simulation-tool-2f23fa1f11b3>
23. Pinheiro, Antônio J., Jeandro de M. Bezerra, Caio AP Burgardt, and Divanilson R. Campelo, "Identifying IoT devices and events based on packet length from encrypted traffic" Computer Communications, vol. 144, pp. 8-17, 2019.
24. A. Sivanathan, H. H. Gharakheili, and V. Sivaraman, “Managing iotcyber-security using programmable telemetry and machine learning,”IEEE Transactions on Network and Service Management, vol. 17, pp.60–74, 2020.
25. J. Ortiz, C. Crawford, and F. Le, “Devicemien: network device behaviour modelling for identifying unknown iot devices,” in Proceedings of the 2nd International Conference on Internet of Things Design and Implementation(IOTDI’19, 2019, pp. 106–117.

References

26. Lopez-Martin, Manuel, Belen Carro, and Antonio Sanchez-Esguevillas. "Neural network architecture based on gradient boosting for IoT traffic prediction." *Future Generation Computer Systems* , vol. 100, pp. 656-673, 2019.
27. M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Iot type-of-traffic forecasting method based on gradient boosting neural networks." *Future Generation Computer Systems*, vol. 105, pp. 331–345, 2020.
28. <https://www.3pillarglobal.com/insights/approaches-tools-techniques-for-security-testing>
29. <https://resources.infosecinstitute.com/popular-tools-for-brute-force-attacks/#gref>
30. <https://www.itpro.co.uk/security/innovation-at-work/29577/the-10-best-or-should-that-be-worst-malware>
31. Kolias, Constantinos, et al. "DDoS in the IoT: Mirai and other botnets." *Computer* 50.7, pp. 80-84, 2017
32. Nguyen, Thanh Thi, and Vijay Janapa Reddi. "Deep reinforcement learning for cyber security." *arXiv preprint arXiv:1906.05799* (2019).

Thank You

For more information, please visit the following links:

gauravsingal789@gmail.com

gaurav.singal@nsut.ac.in

<https://www.linkedin.com/in/gauravsingal789/>

<http://www.gauravsingal.in>