



Physical Computing



ครั้งที่ 16. Microcontroller
Internet of Things



What is Internet of Things

นิยามของ Internet of Things ที่เผยแพร่ในเอกสารของ IEEE เมื่อ 27 May 2015

An IoT is a network that connects uniquely identifiable “Things” to the Internet.

The “Things” have sensing/actuation and potential programmability capabilities.

Through the exploitation of unique identification and sensing information about the “Thing” can be collected and the state of the “Things” can be changed from anywhere, anytime, by anything.

IoT คือเครือข่ายที่เชื่อมต่อ “สรรพสิ่ง” ที่สามารถระบุตัวตนได้อย่างเป็นเอกลักษณ์ เข้ากับอินเทอร์เน็ต “สรรพสิ่ง” เหล่านี้มีความสามารถในการรับรู้/ตอบสนอง และอาจมีความสามารถในการตั้งโปรแกรมได้

ด้วยการใช้ประโยชน์จากการระบุตัวตนที่เป็นเอกลักษณ์และข้อมูลการรับรู้ สามารถรวบรวมข้อมูลเกี่ยวกับ “สรรพสิ่ง” นั้นได้ และสามารถเปลี่ยนแปลงสถานะของ “สรรพสิ่ง” เหล่านั้นได้จากที่ไหนก็ได้ เมื่อไหร่ก็ได้ โดยอะไรก็ได้

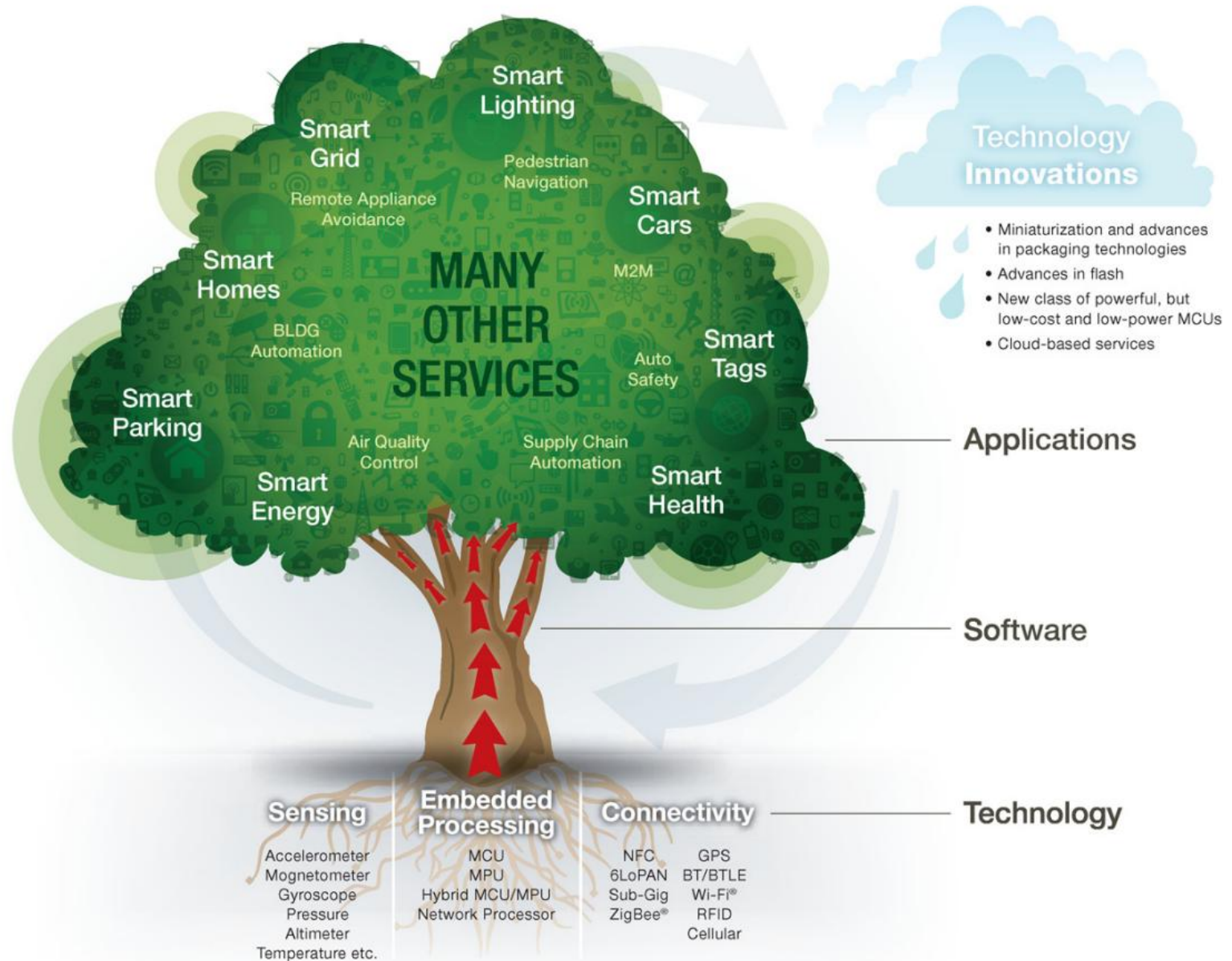
What is Internet of Things

Internet of Things features

- Interconnection of Things
- Connection of Things to the Internet
- Uniquely Identifiable Things
- Ubiquity
- Sensing/Actuation capability
- Embedded intelligence
- Interoperable Communication Capability
- Self configurability
- Programmability

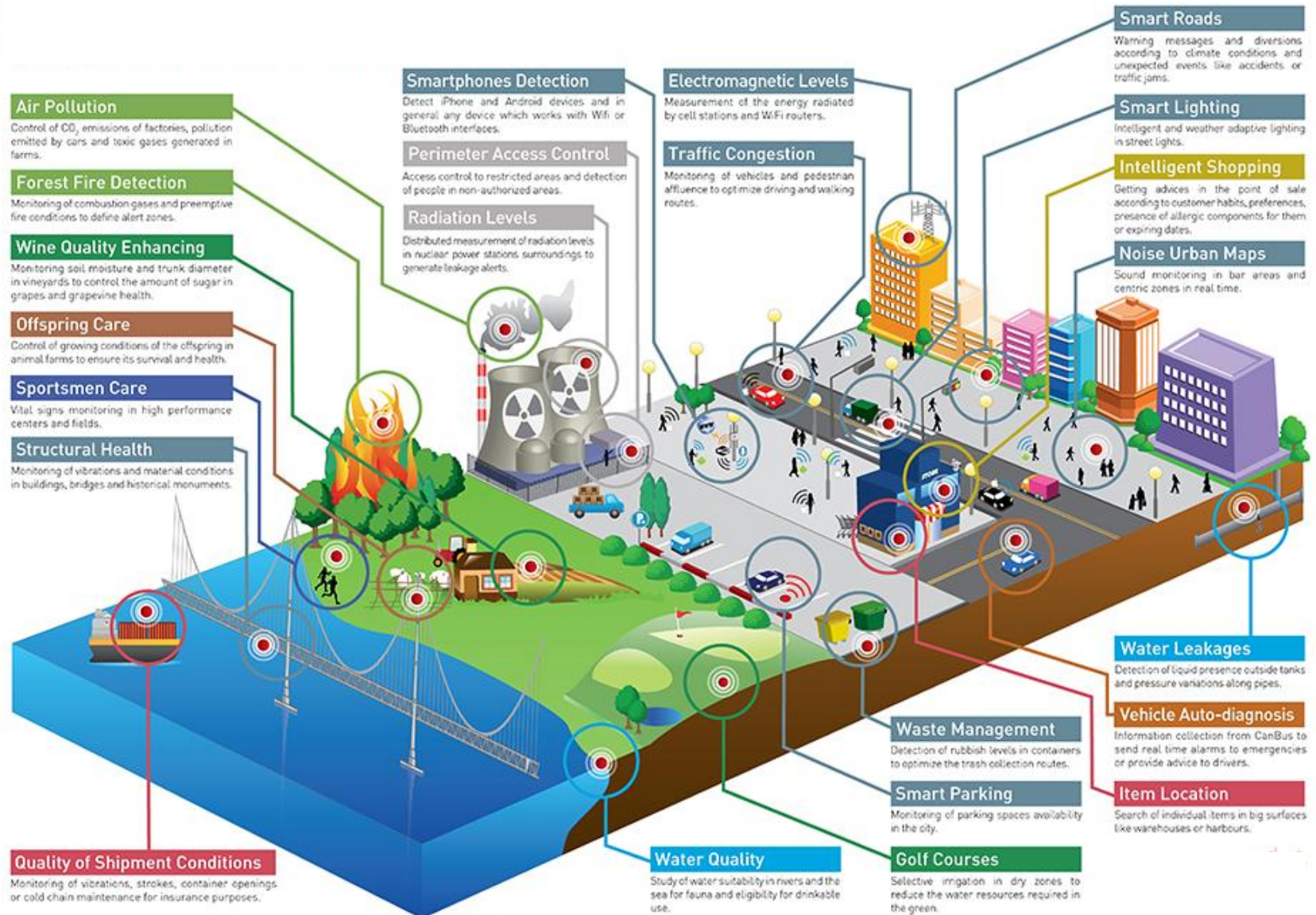
เทคโนโลยีการเชื่อมต่อของสรรพสิ่ง

- Internet of Things หรือ IoT คือ สภาพแวดล้อมอันประกอบด้วยสรรพสิ่งที่สามารถสื่อสารและเชื่อมต่อกันได้ผ่านโพรโทคอลการสื่อสารทั้งแบบใช้สายและไร้สาย โดยสรรพสิ่งต่างๆ มีวิธีการระบุตัวตนได้ รับรู้บริบทของสภาพแวดล้อมได้ และมีปฏิสัมพันธ์โต้ตอบและทำงานรวมกันได้
- IoT จะเปลี่ยนรูปแบบและกระบวนการผลิตในภาคอุตสาหกรรมไปสู่ยุคใหม่ หรือที่เรียกว่า Industry 4.0 ที่จะอาศัยการเชื่อมต่อสื่อสารและทำงานร่วมกันระหว่างเครื่องจักร มนุษย์ และข้อมูล เพื่อเพิ่มอำนาจในการตัดสินใจที่รวดเร็วและมีความถูกต้องแม่นยำสูง
- โดยเทคโนโลยีที่ทำให้ IoT เกิดขึ้นได้จริงและสร้างผลกระทบในวงกว้างได้ แบ่งออกเป็นสามกลุ่มได้แก่
 - 1) **เทคโนโลยีที่ช่วยให้สรรพสิ่งรับรู้ข้อมูล** ในบริบทที่เกี่ยวข้อง เช่น เซ็นเซอร์
 - 2) **เทคโนโลยีที่ช่วยให้สรรพสิ่งมีความสามารถในการสื่อสาร** เช่น ระบบสมองกลฝังตัว รวมถึงการสื่อสารแบบไร้สายที่ใช้พลังงานต่ำ อาทิ Zigbee, 6LowPAN, Low-power Bluetooth
 - 3) **เทคโนโลยีที่ช่วยให้สรรพสิ่งประมวลผลข้อมูล** ในบริบทของตน เช่น เทคโนโลยีการประมวลผลแบบคลาวด์ และเทคโนโลยีการวิเคราะห์ข้อมูลขนาดใหญ่ หรือ Big Data Analytics



The IoT: Different Services, Technologies, Meanings for Everyone [1]

IoT Smart-X Application



IoT Use case : Smart City



- Smart Parking
- Structural health
- Noise Urban Maps
- Smartphone Detection
- Electromagnetic Field Levels
- Traffic Congestion
- Smart Lighting
- Waste Management
- Smart Roads
- Smart Meters

IoT Use case : Smart Environment



- Forest Fire Detection
- Air Pollution
- Snow Level Monitoring
- Landslide Prevention
- Earthquake Early Detection

IoT Use case : Smart water

An aerial photograph showing a river that has overflowed its banks, flooding surrounding green fields and some residential areas. The water is a muddy brown color. The year '2016' is written in the bottom left corner of the image.

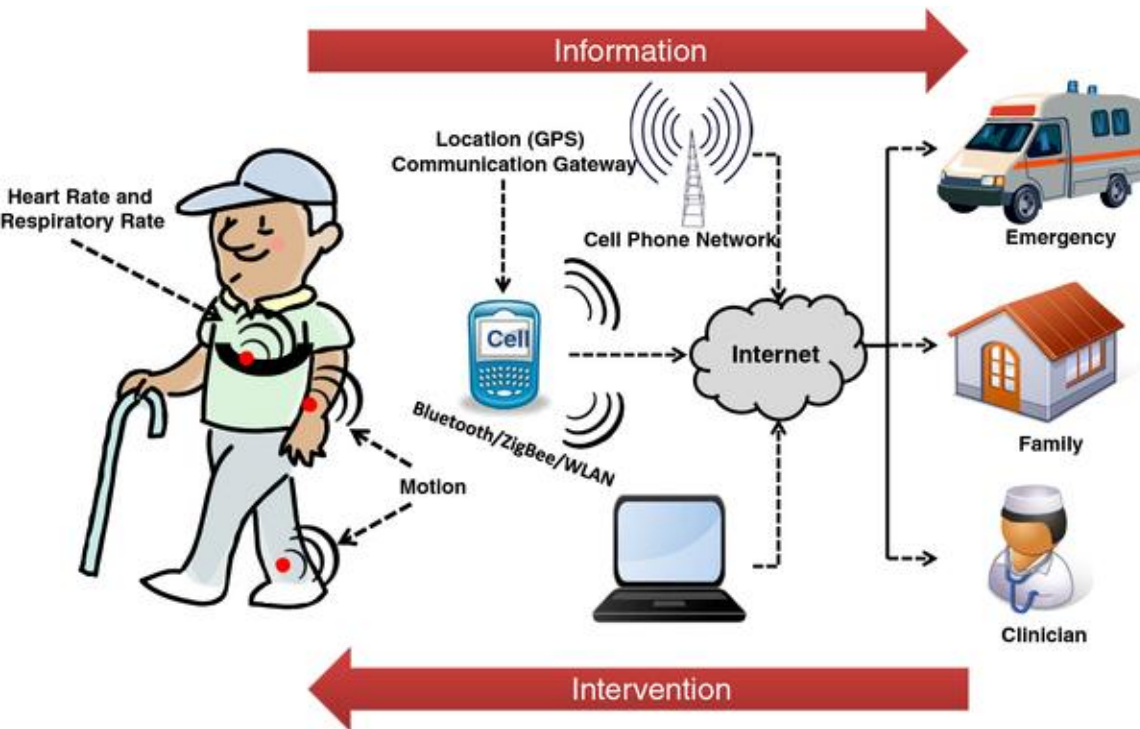
Supporting New England Communities to Become River-Smart

Policies and Programs That Can Help New England Towns Thrive Despite River Floods

- Potable water monitoring
- Chemical leakage detection in rivers
- Pollution levels in the sea
- Water Leakages
- River Floods

*Potable water : น้ำดื่ม

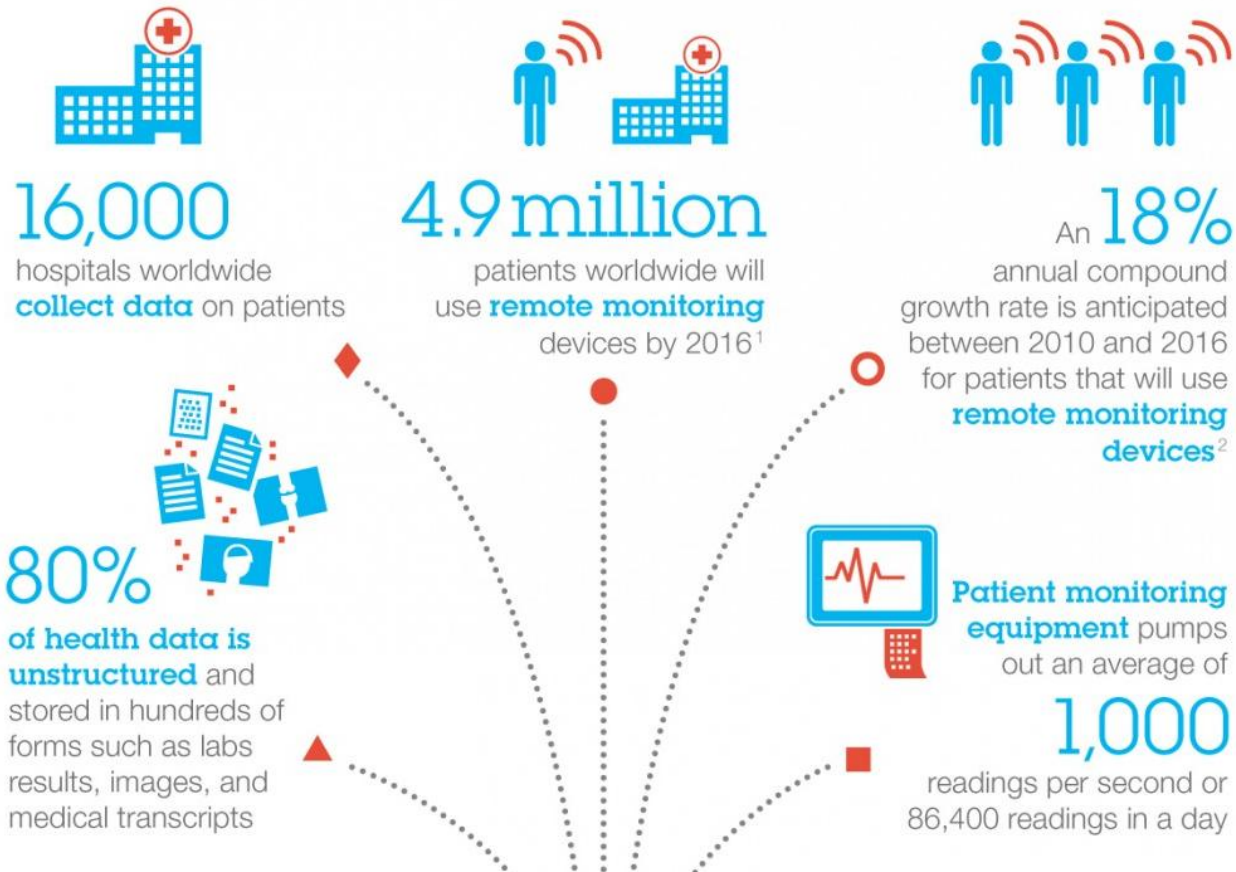
IoT Use case : Smart Health



- Fall Detection
- Medical Fridges
- Vital signs monitoring
- Patients Surveillance
- Ultraviolet Radiation

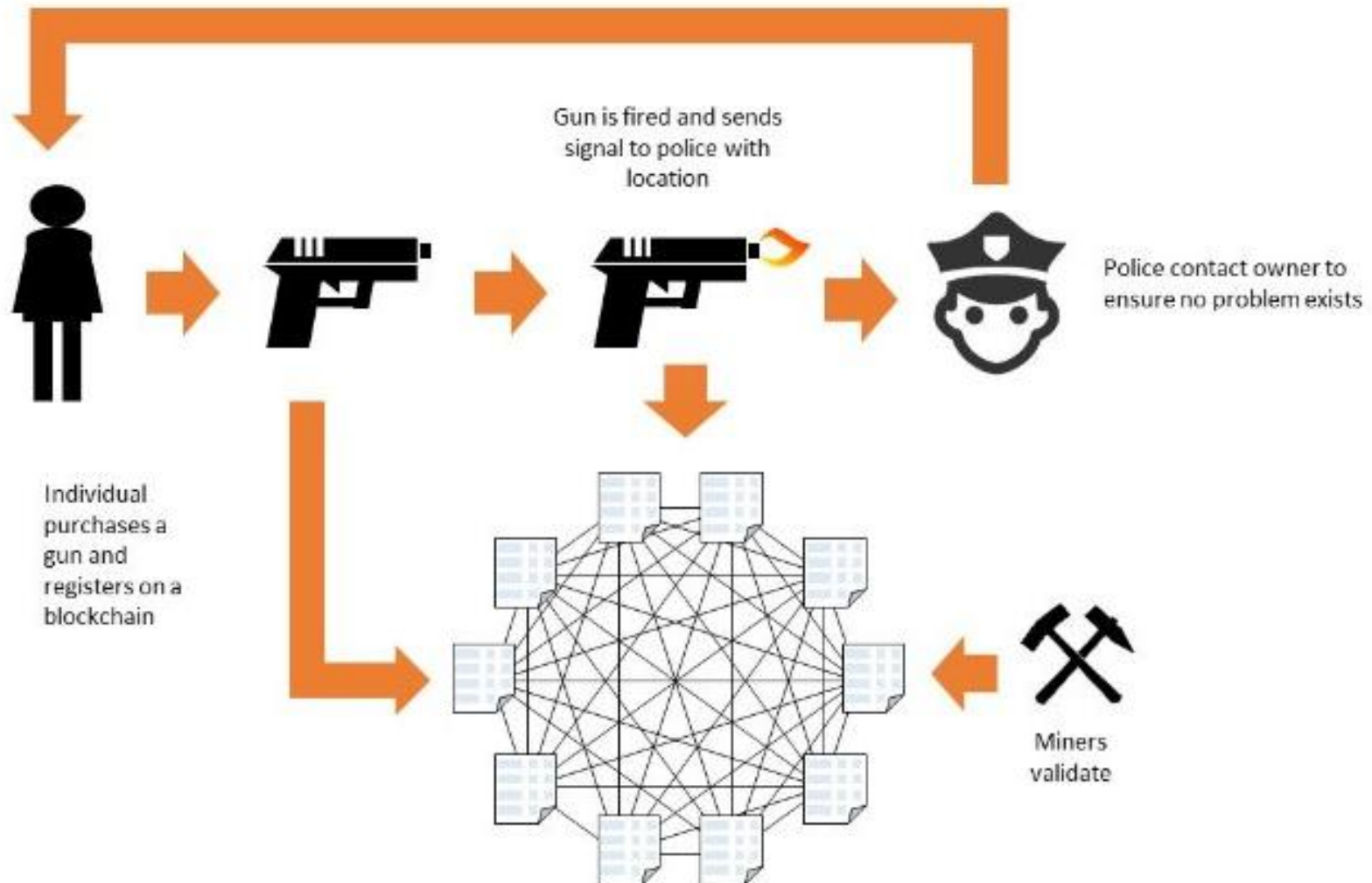
IoT & Data Analytic ➔ Pattern ➔ Alert

ตัวอย่าง IoT และ Big Data ในเรื่อง Health



The ability to analyze big data in motion in real-time as it streams in can help predict the onset of illness and respond instantly from new insight that will help transform healthcare.

ตัวอย่าง IoT และ Blockchain

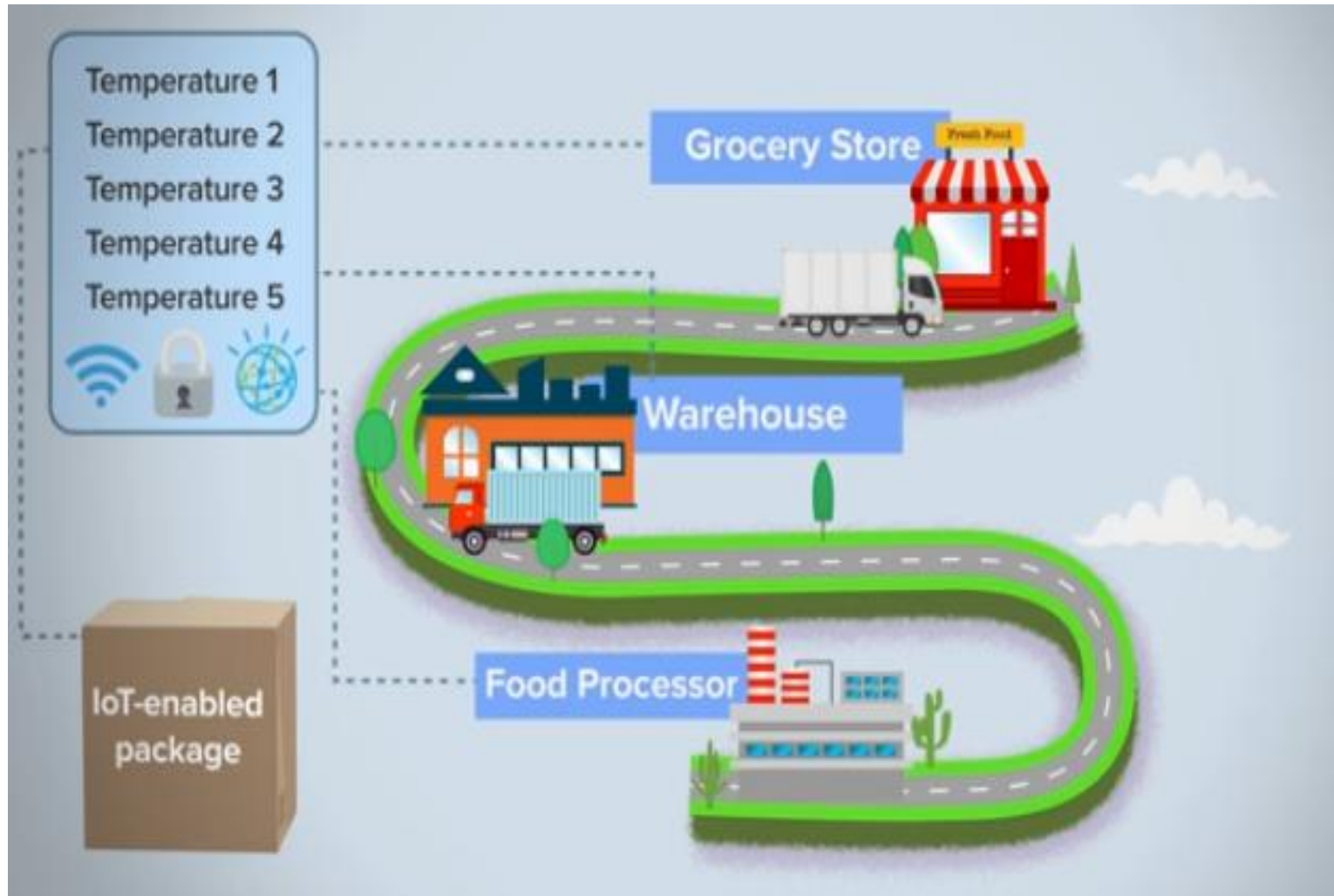


ตัวอย่าง IoT และ Blockchain



ตัวอย่าง IoT และ Blockchain

การควบคุมอุณหภูมิสินค้า การตรวจสอบ

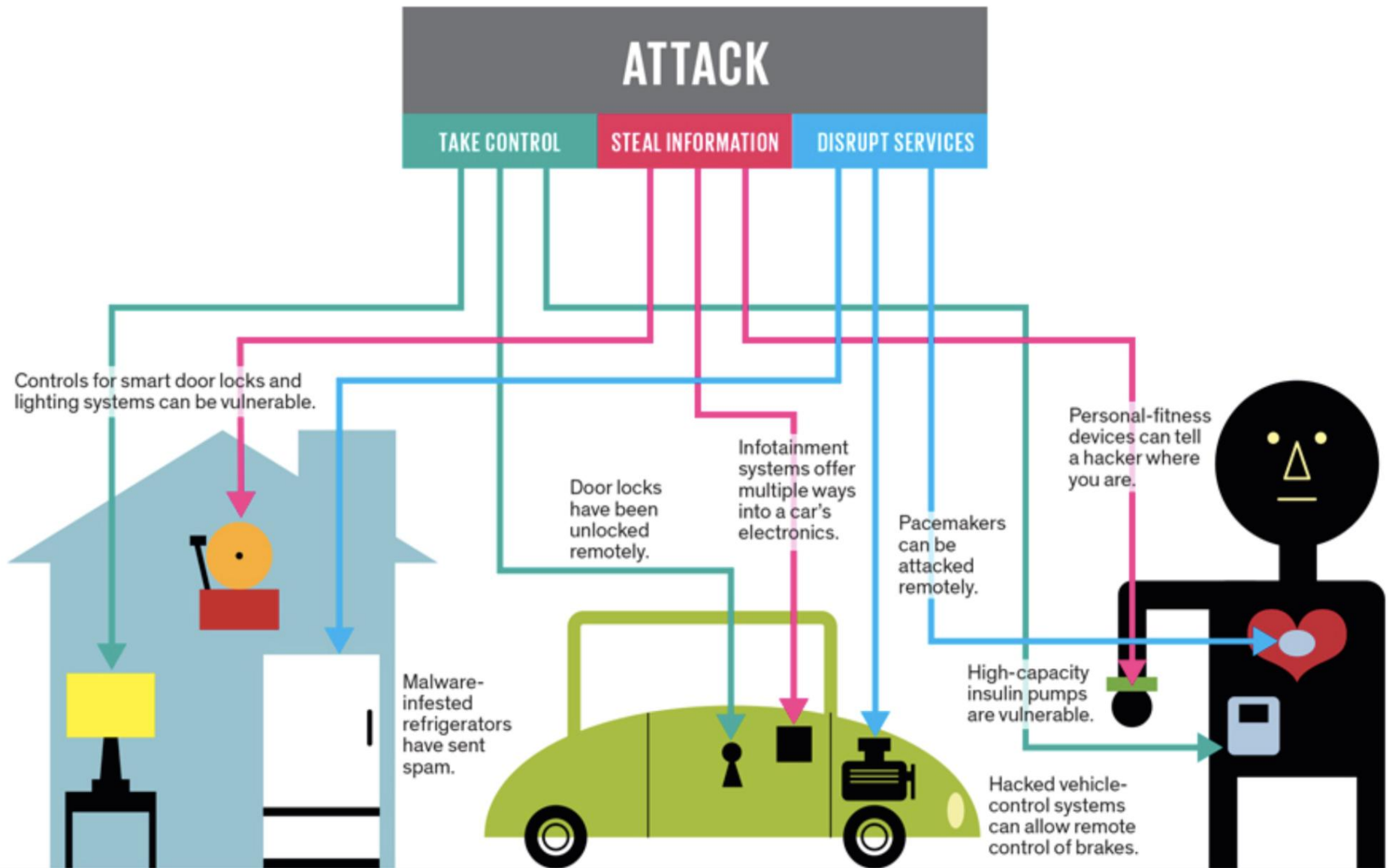


ตัวอย่าง IoT และ Blockchain

การประกัน การเดินเรือ การเดินเรือนอกเส้นทาง การรักษาสภาพตู้ container



Security ของ IoT



Short Range



Long Range Unlicensed



Long Range Licensed



connectivity options for things

Low-power, Long-range WAN for IoT

Overview Wireless Technologies for IoT

PAN

Short Range
Communicating Devices



Well established standards

Good for:

- Mobile
- In-home
- Short range

Not good for:

- Battery life
- Long range

Cellular

Long Range w/ Power
Traditional M2M



Well established standards

Good for:

- Long range
- High data-rate
- Coverage

Not good for:

- Battery life

Low-Power WAN

Long Range w/ Battery
Internet of Objects



Emerging PHY standards

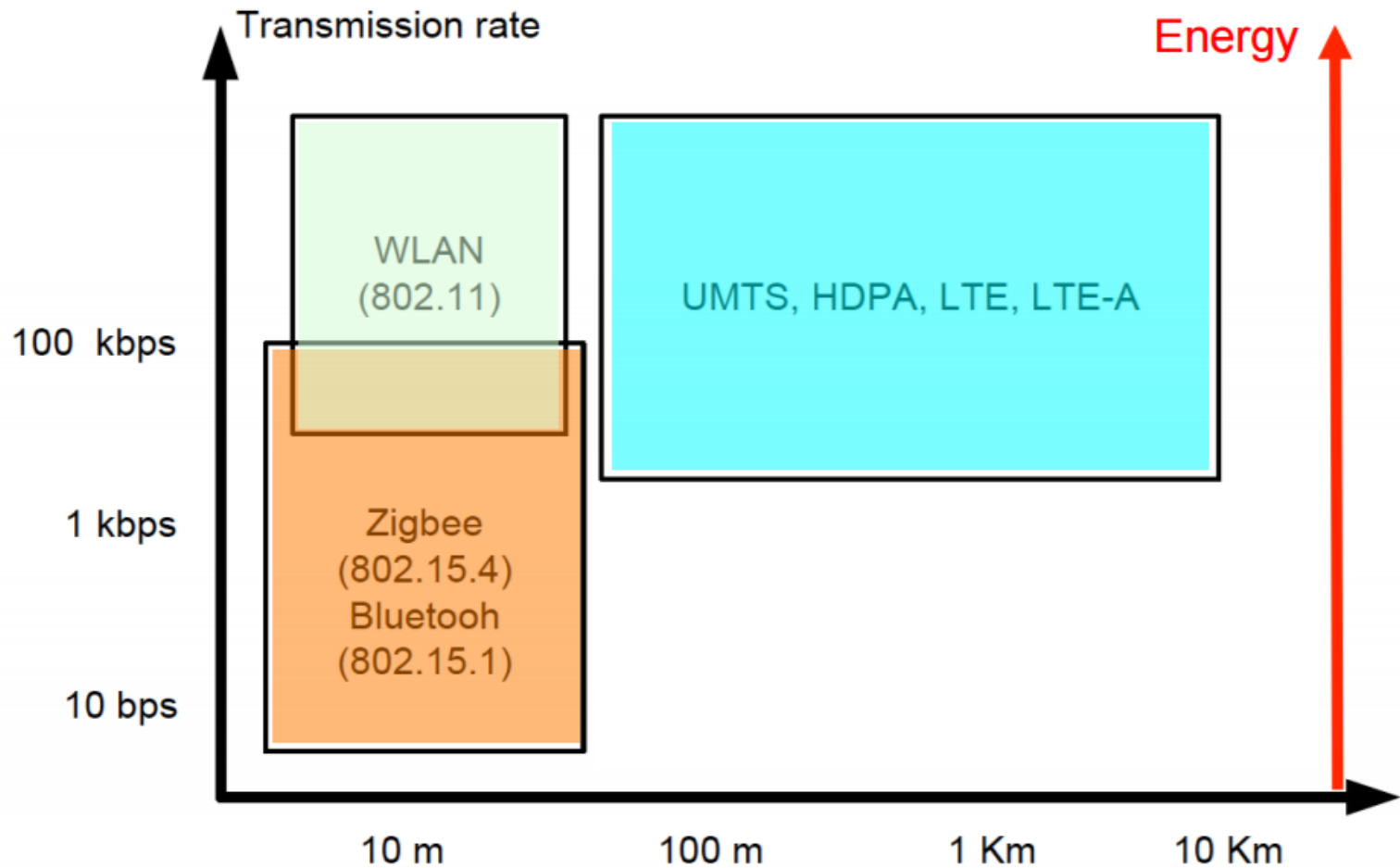
Good for:

- Long range
- Long battery
- Low cost

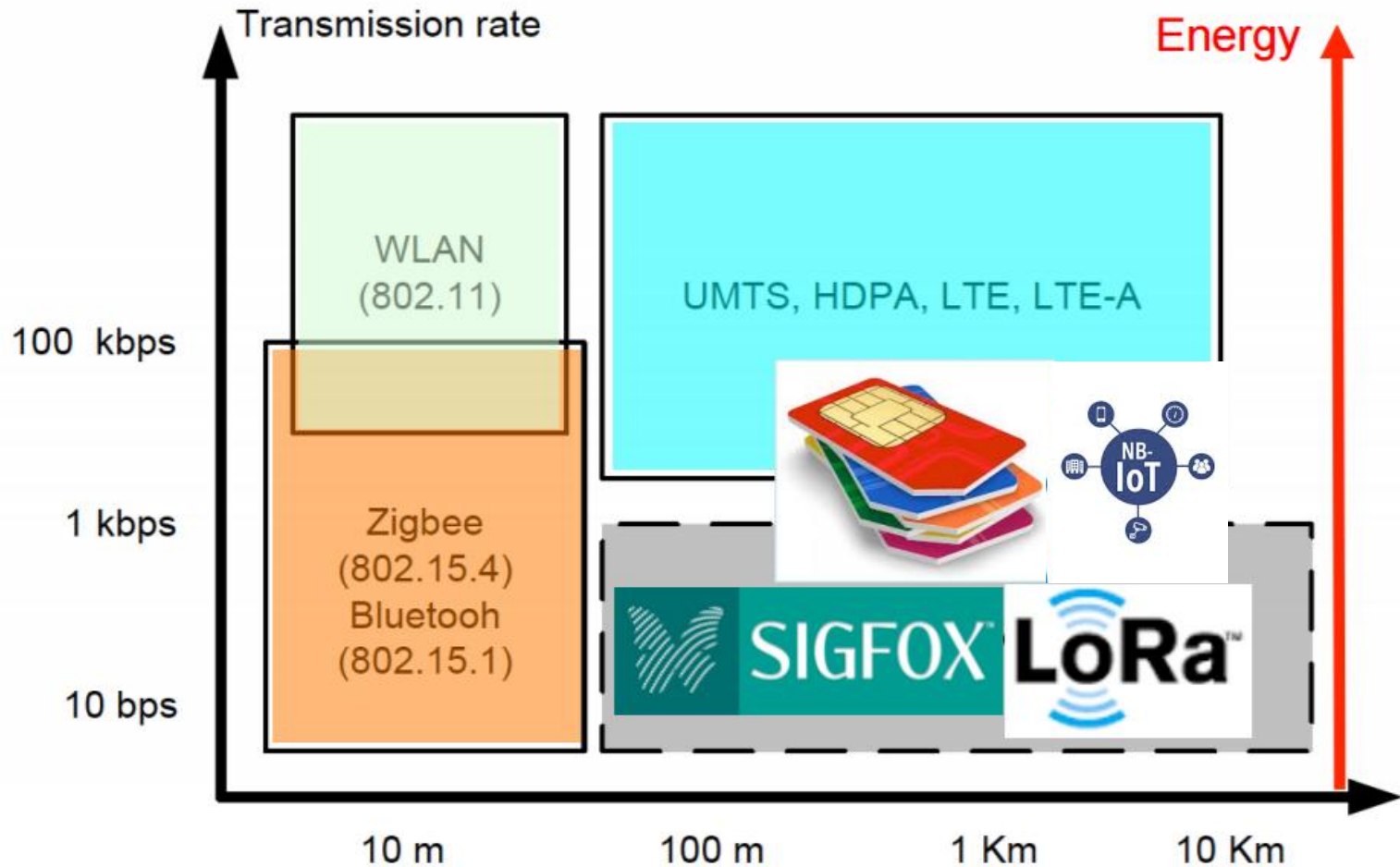
Not good for:

- High data-rate

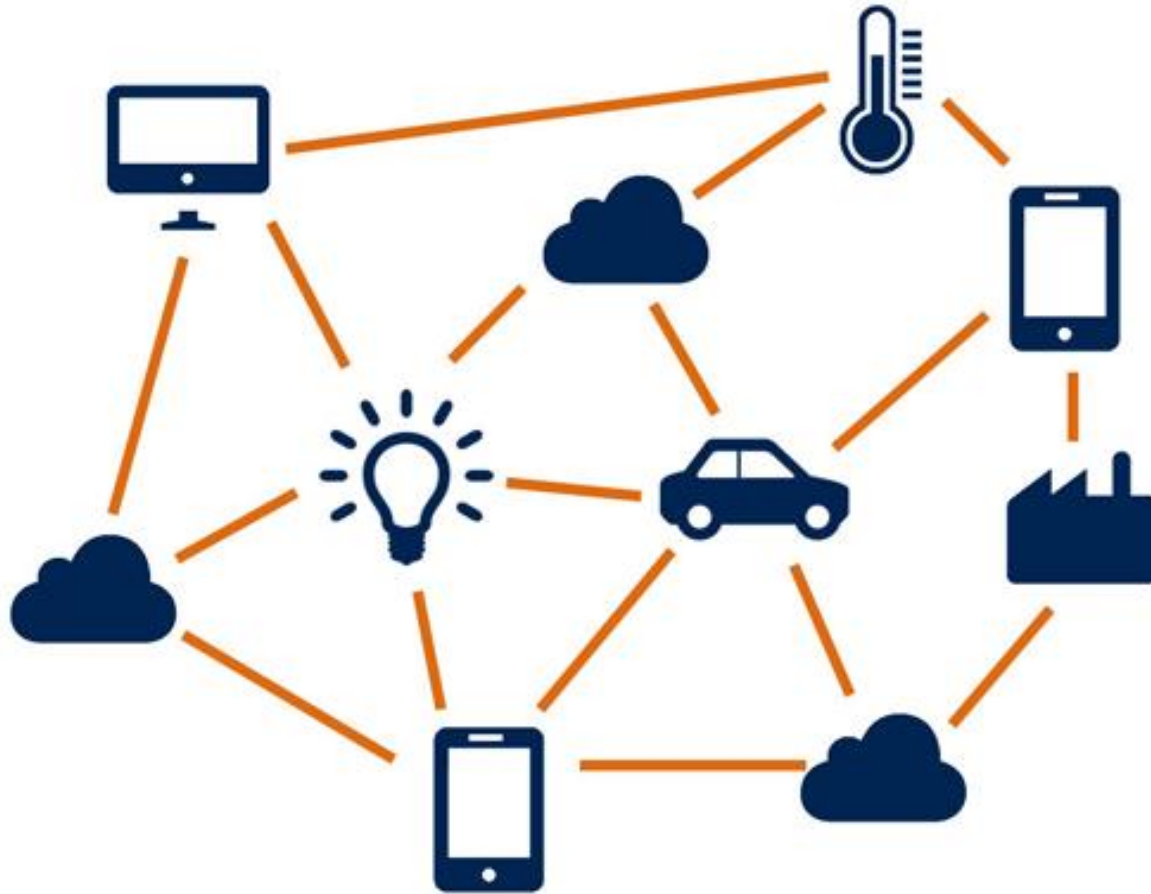
Low-power, Long-range WAN for IoT



Low-power, Long-range WAN for IoT



Internet of Things Protocol



MQTT Protocol

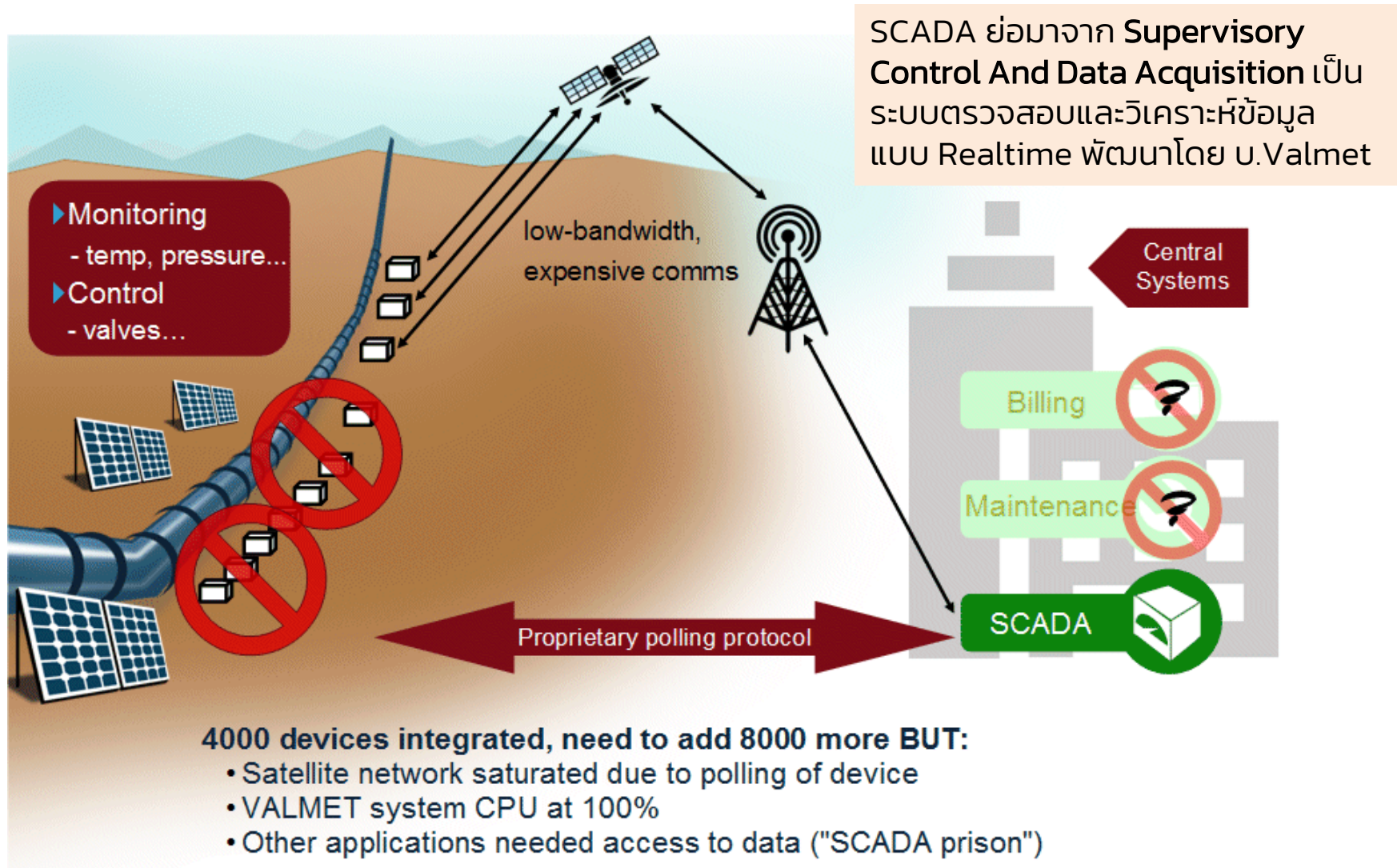
- MQTT : Message Queuing Telemetry Transport
- Invented by Andy Stanford Clark (IBM) and Arlen Nipper (Eurotech) in 1999
- Originally envisioned for use over Satellite link from an oil pipeline



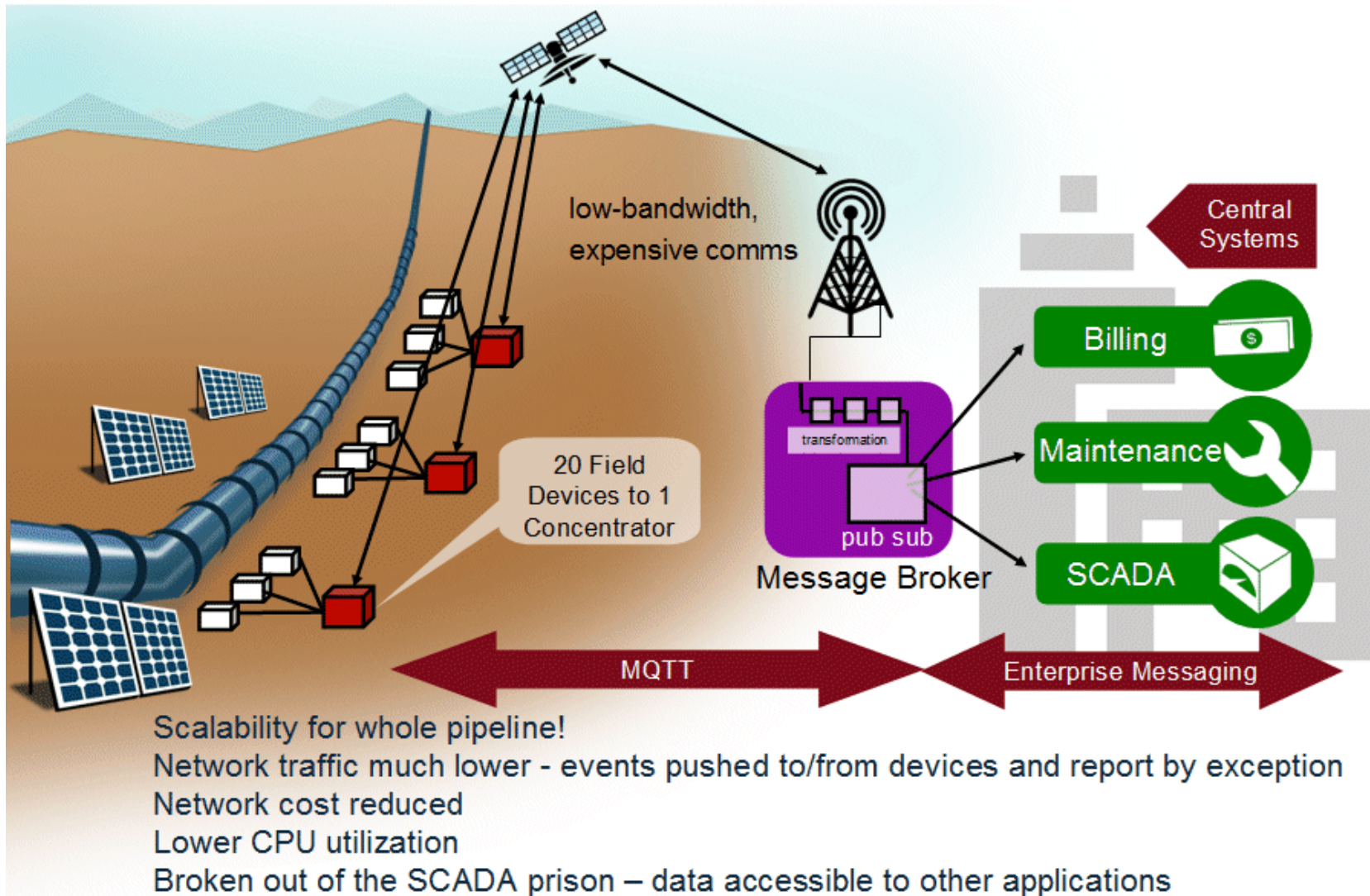
Use case:

- 30.000 devices
- 17.000 km pipeline
- Remote monitoring
- Remote control
- Uses satellite links
- Bandwidth is **very** expensive

MQTT Protocol



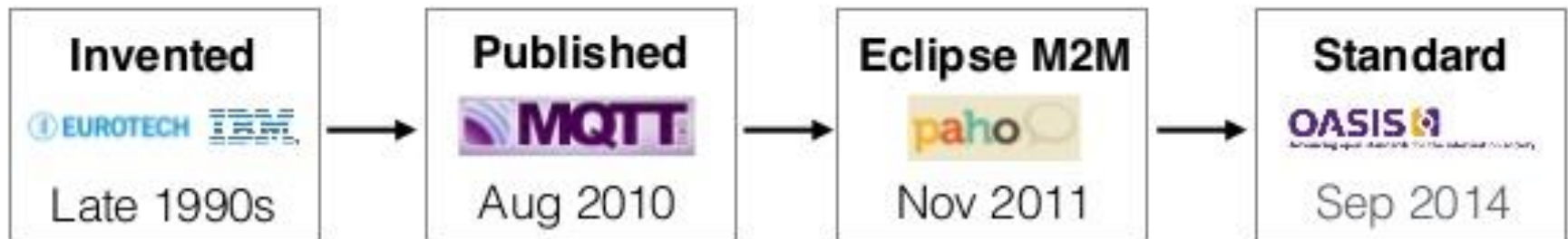
MQTT Protocol



MQTT Protocol

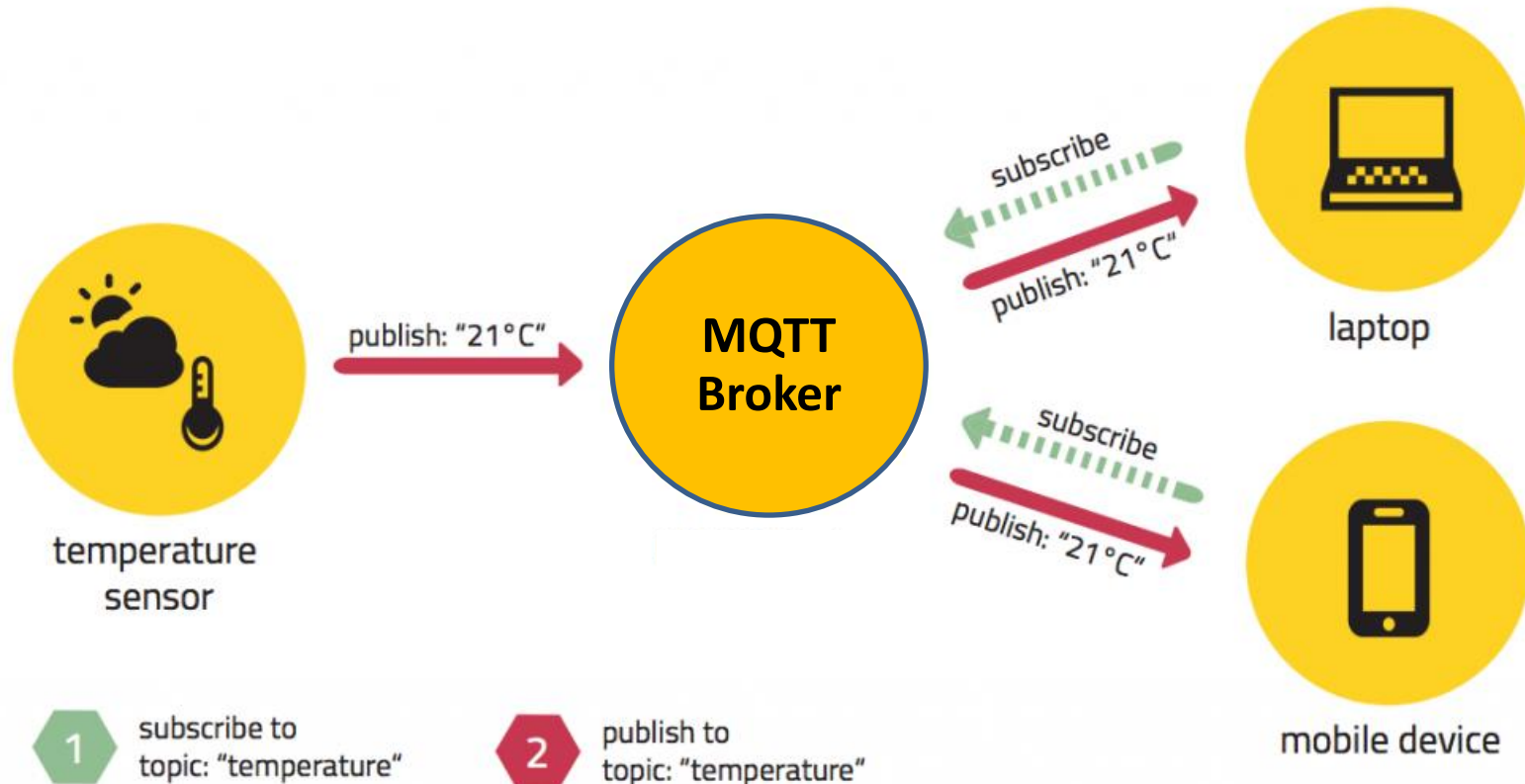
MQTT a lightweight protocol for IoT **messaging**

- **open** open spec, standard 40+ client implementations
- **lightweight** minimal overhead efficient format tiny clients (kb)
- **reliable** QoS for reliability on unreliable networks
- **simple** 43-page spec connect + publish + subscribe



The publish/subscribe pattern

Bi-directional , Asynchronous "**Push**" Communication



MQTT Protocol

MQTT

simple to implement

Connect

Subscribe

Publish

Unsubscribe

Disconnect

```
client = new Messaging.Client(hostname, port, clientId)
client.onMessageArrived = messageArrived;
client.onConnectionLost = connectionLost;
client.connect({ onSuccess: connectionSuccess });

function connectionSuccess() {
    client.subscribe("planets/earth");
    var msg = new Messaging.Message("Hello world!");
    msg.destinationName = "planets/earth";
    client.publish(msg);
}

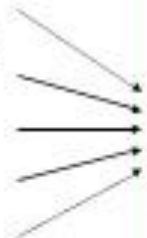
function messageArrived(msg) {
    console.log(msg.payloadString);
    client.unsubscribe("planets/earth");
    client.disconnect();
}
```


Subscribe

MQTT

allows **wildcard** subscriptions

scores/football/big12/Texas
scores/football/big12/TexasTech
scores/football/big12/Oklahoma
scores/football/big12/IowaState
scores/football/big12/TCU
scores/football/big12/OkState
scores/football/big12/Kansas
scores/football/SEC/TexasA&M
scores/football/SEC/LSU
scores/football/SEC/Alabama



scores/football/big12/Texas



Texas Fan

scores/football/big12/+



Big 12 Fan

scores/#

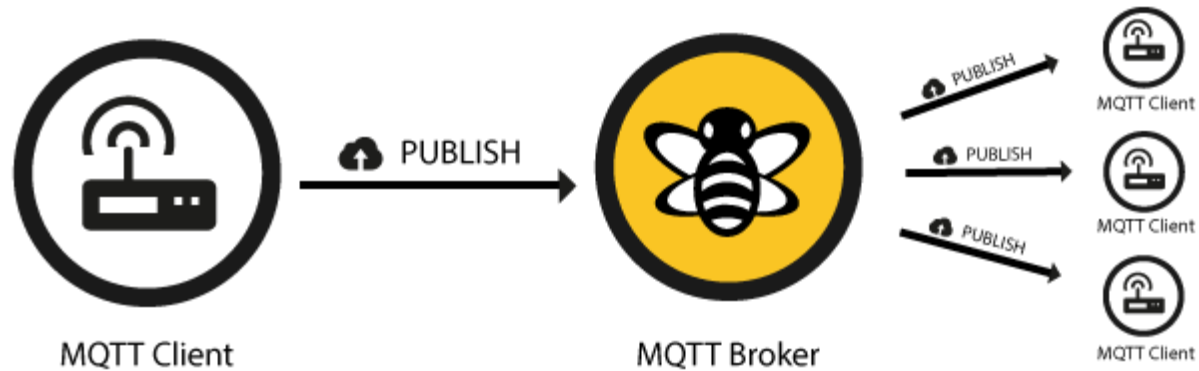


ESPN

single level wildcard: +

multi-level wildcard: #

Publish



MQTT-Packet:

PUBLISH



contains:

`packetId` (always 0 for qos 0)
`topicName`
`qos`
`retainFlag`
`payload`
`dupFlag`

Example

4314
"topic/1"
1
false
"temperature:32.5"
false

Publish

Topics

A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).



QoS

A Quality of Service Level (QoS) for this message. The level (0,1 or 2) determines the guarantee of a message reaching the other end

Retain-Flag

This flag determines if the message will be saved by the broker for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing. (ให้ Broker เก็บ Payload ล่าสุด ที่ส่งขึ้นไปไว้ด้วย ถ้ามีคนใหม่เข้ามา Subscribe ที่หลัง ก็จะได้รับข้อมูลนี้ด้วย)

Payload

This is the actual content of the message.

DUP flag

The duplicate flag indicates, that this message is a duplicate and is resent because the other end didn't acknowledge the original message. This is only relevant for QoS greater than 0

Packet Identifier

The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero.

QoS

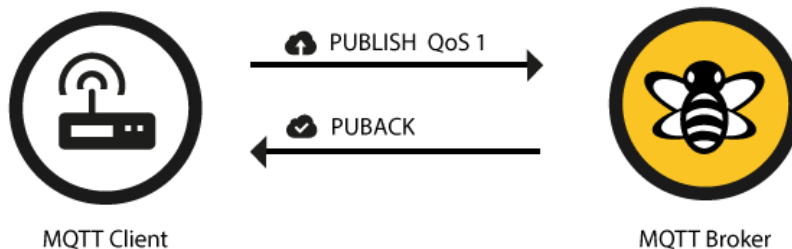
- **QoS 0 – at most once**

The minimal level is zero and it guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored and redelivered by the sender.



- **QoS 1 – at least once**

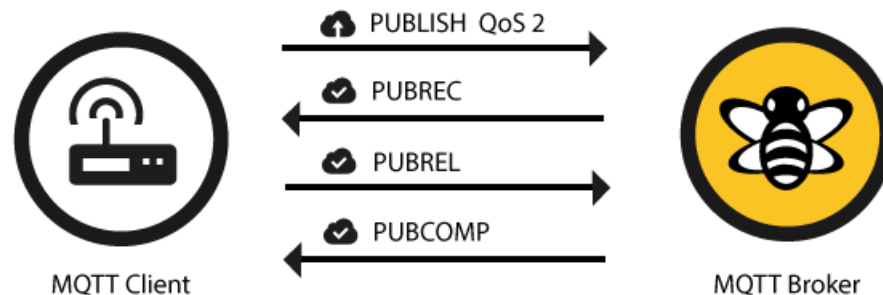
it is guaranteed that a message will be delivered at least once to the receiver. The sender will store the message until it gets an acknowledgement in form of a [PUBACK](#) command message from the receiver.



QoS

- **QoS 2 – Exactly once**

The highest QoS is 2, it guarantees that each message is received only once by the counterpart. It is the safest and also the slowest quality of service level. The guarantee is provided by two flows there and back between sender and receiver.



Retain messages

MQTT

retained messages for last value caching



CONNECT ID=thing1 →
PUBLISH thing1/battery {"value":95} RETAIN →
PUBLISH thing1/battery {"value":94} RETAIN →
PUBLISH thing1/battery {"value":93} RETAIN →
DISCONNECT →



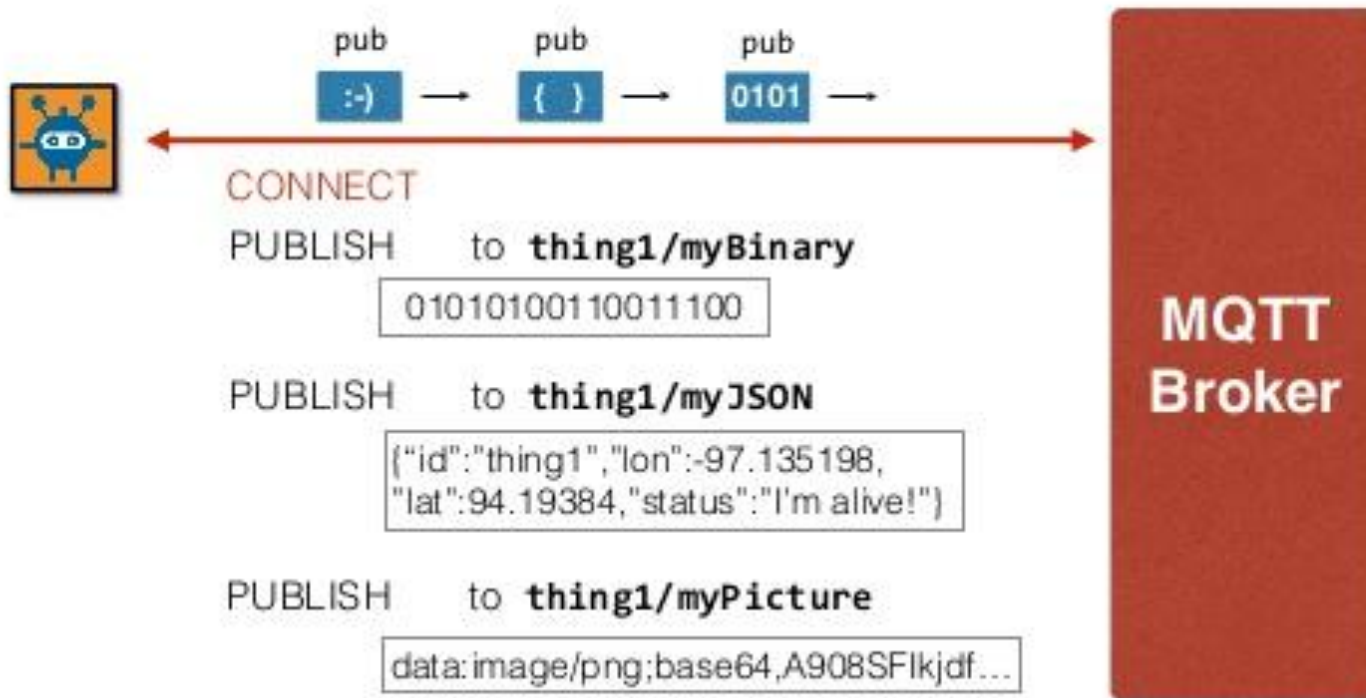
CONNECT ID=thing2 →
SUBSCRIBE thing1/battery →
← RETAIN thing1/battery {"value":93} PUBLISH

MQTT
Broker

Payload

MQTT

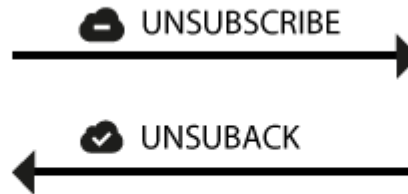
agnostic payload for flexible delivery



Unsubscribe



MQTT Client



MQTT Broker

MQTT-Packet:

UNSUBSCRIBE



contains:

`packetId`
`topic1` } (list of topics)
`topic2`
...

Example

4315
"topic/1"
"topic/2"
...

MQTT-Packet:

UNSUBACK



contains:

`packetId`

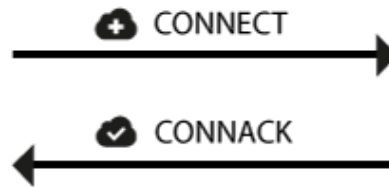
Example

4316

Connect & Disconnect



MQTT Client



MQTT Broker

MQTT-Packet:

CONNECT



contains:	Example
<code>clientId</code>	<code>"client-1"</code>
<code>cleanSession</code>	<code>true</code>
<code>username</code> (optional)	<code>"hans"</code>
<code>password</code> (optional)	<code>"letmein"</code>
<code>lastWillTopic</code> (optional)	<code>"/hans/will"</code>
<code>lastWillQos</code> (optional)	<code>2</code>
<code>lastWillMessage</code> (optional)	<code>"unexpected exit"</code>
<code>keepAlive</code>	<code>60</code>

MQTT-Packet:

CONNACK



contains:	Example
<code>sessionPresent</code>	<code>true</code>
<code>returnCode</code>	<code>0</code>

MQTT-Packet:

DISCONNECT

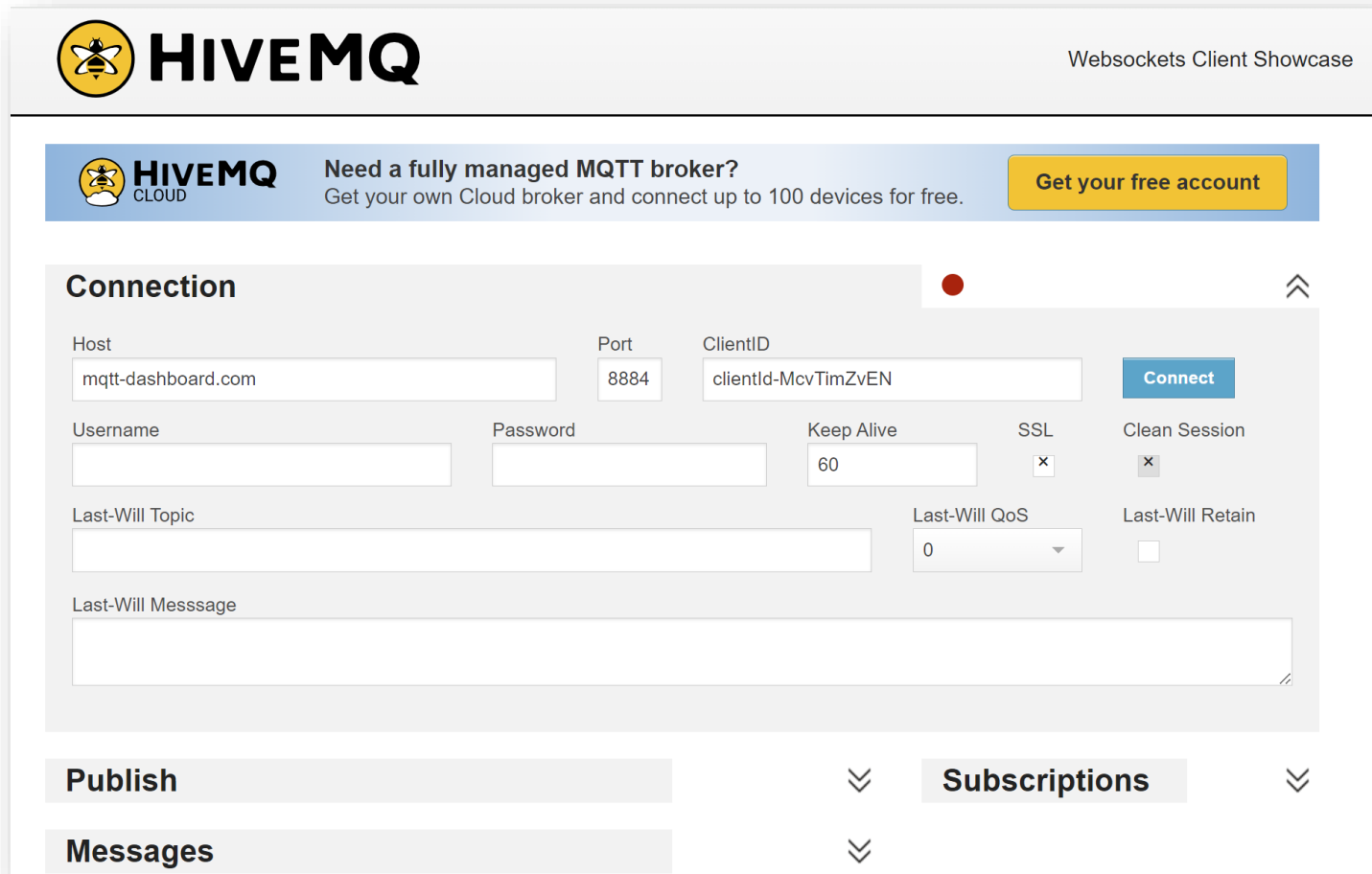


no payload

การทดลองที่ 1. Public MQTT Broker

- เข้า Website ที่ URL

<http://www.hivemq.com/demos/websocket-client/>



The screenshot shows the HiveMQ Websockets Client Showcase interface. At the top, there is a header with the HiveMQ logo and the text "Websockets Client Showcase". Below the header, there is a blue banner with the HiveMQ Cloud logo, the text "Need a fully managed MQTT broker? Get your own Cloud broker and connect up to 100 devices for free.", and a yellow button labeled "Get your free account".

The main section is titled "Connection" and contains a form for connecting to an MQTT broker. The form has the following fields and controls:

- Host:** A text input field containing "mqtt-dashboard.com".
- Port:** A text input field containing "8884".
- ClientID:** A text input field containing "clientId-McvTimZvEN".
- Connect:** A blue button.
- Username:** A text input field.
- Password:** A text input field.
- Keep Alive:** A text input field containing "60".
- SSL:** A checkbox with an "x" icon, indicating it is checked.
- Clean Session:** A checkbox with an "x" icon, indicating it is checked.
- Last-Will Topic:** A text input field.
- Last-Will QoS:** A dropdown menu with "0" selected.
- Last-Will Retain:** A checkbox.
- Last-Will Message:** A large text area.

At the bottom of the interface, there are three tabs: "Publish", "Subscriptions", and "Messages". The "Publish" and "Subscriptions" tabs are currently selected, and the "Messages" tab is collapsed.

<http://www.hivemq.com/demos/websocket-client/>



การทดลองที่ 1. Public MQTT Broker

- **Connect**

- กดปุ่ม Connect

- **Subscribe**

- กด Add Topic Subscription เพื่อ add หัวข้อ ที่ต้องการ subscribe
- กด Subscribe

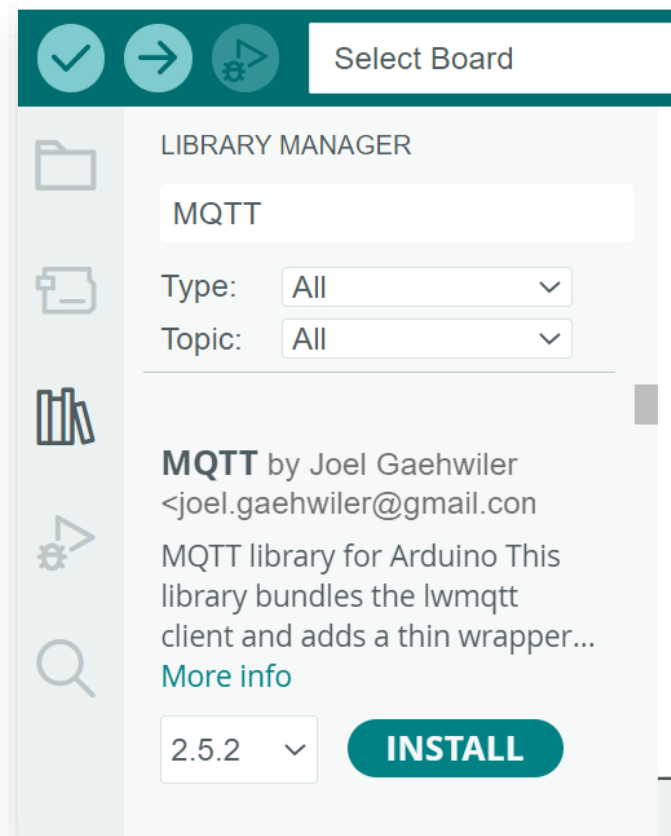
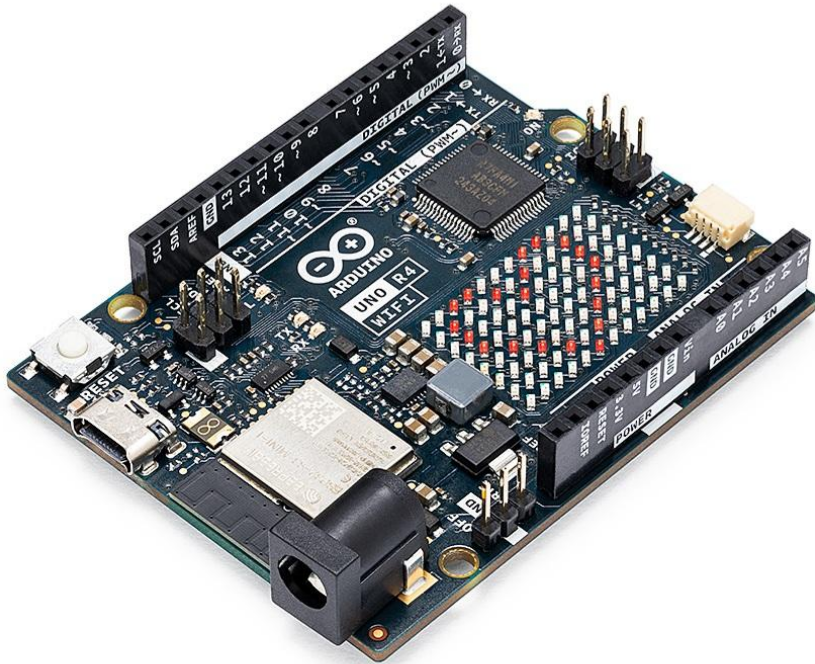
- **Publish**

- กำหนด topic ที่ต้องการจะ publish
- พิมพ์ ข้อความ ที่จะ publish
- กด publish

Laboratory : Library MQTT

ติดตั้ง Library

ค้นหา MQTT จากนั้นกด **INSTALL**



<https://github.com/256dpi/arduino-mqtt>

Library MQTT

- `MQTTClient client;` สร้าง object จาก class `MQTTClient`
ค่า default buffer คือ 128 bytes
- `client.begin(MQTT_BROKER_ADDRESS, MQTT_PORT, network);`
เริ่มต้นการเชื่อมต่อ (initialize) ระหว่าง MQTT client
กับ MQTT broker โดยใช้ network client
- `client.onMessage(messageReceived);`
คือการ ตั้ง **Callback Function** เมื่อมีข้อความ (message)
เข้ามาจาก MQTT Broker — ให้เรียกฟังก์ชัน
`messageReceived` เพื่อจัดการข้อความนั้น
- `client.connect(clientID, username, password);`
ให้ client พยายามเชื่อมต่อกับ MQTT Broker โดยใช้ชื่อ
`clientID`, ชื่อผู้ใช้ (username), และรหัสผ่าน (password)

Library MQTT

- `client.connected()` ตรวจสอบการเชื่อมต่อ เชื่อมต่อ : `true`, หาดู : `False`
- `client.subscribe("some/topic", qos);`
- `client.unsubscribe("some/topic");`
- `client.publish("some/topic", payload, retained, qos);`
- `client.loop();`
- `client.disconnect();`

Wi-Fi®

The ESP32 onboard the UNO R4 WiFi is used to give the board Wi-Fi® capabilities. The Wi-Fi® module has a bitrate of up to 150 Mbps. The ESP32 module has a built in trace-antenna, meaning that you do not need an external one to use the connectivity features of the board. However, this trace antenna is shared with the Bluetooth® module, which means that you cannot use Bluetooth® and Wi-Fi® at the same time.

To use the Wi-Fi® features of the UNO R4 WiFi, use the **WiFiS3** library that is built in to the UNO R4 Board Package.

Wi-Fi®

- Wi-Fi® support with 802.11 b/g/n standard (Wi-Fi® 4)
- Bit rate at up to 150 Mbps
- 2.4 GHz band

<https://docs.arduino.cc/tutorials/uno-r4-wifi/wifi-examples/>

Library WiFiS3

เป็นไลบรารีสำหรับควบคุมการเชื่อมต่อ WiFi บนบอร์ด Arduino

- `WiFiClient network;` สร้าง object จาก class `WiFiClient`

การเชื่อมต่อ WiFi

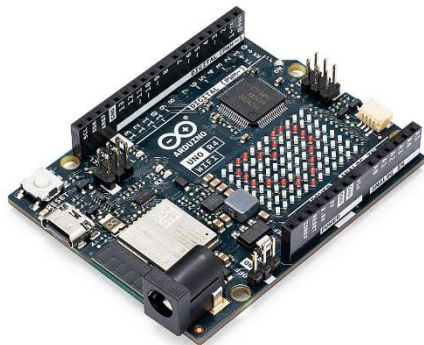
- `WiFi.begin(ssid, pass);` เริ่มต้นการเชื่อมต่อ WiFi
- `WiFi.disconnect()`
- `WiFi.status()` ตรวจสอบสถานะการเชื่อมต่อ ผลลัพธ์ เช่น `WL_CONNECTED` , `WL_IDLE_STATUS` , `WL_NO_SSID_AVAIL`
- `WiFi.reconnect()`

การจัดการข้อมูล

- `WiFi.localIP()` ค่า IP ที่บอร์ดได้รับจาก router
- `WiFi.gatewayIP()` ค่า IP ของ gateway (router)
- `WiFi.SSID()` ชื่อ Wi-Fi ที่กำลังเชื่อมต่ออยู่
- `WiFi.RSSI()` ค่าความแรงสัญญาณ (RSSI) หน่วย dBm

การทดลองที่ 2. MQTT with UNO R4 – WiFi

- นำมือถือของ นศ เปิด Hot Spot ตั้งค่า SSID, Password
- แกะค่าต่างๆ ในโปรแกรม ให้ถูกต้อง เช่น SSID, Password ,ClientID , Broker Address , Topic
- ทำการ Upload Code ที่แก้ค่าแล้ว ลงบนบอร์ด
- เปิด Serial Monitor เพื่อดูค่า
- เปิด Website hiveMQ ใส่ค่า Topic ที่ publish, Subscribe ให้ถูกต้อง
- สังเกตค่าที่ส่งมา ลองทำการ Pub/Sub ไปยังบอร์ด



Code ตัวอย่าง Publish/Subscribe MQTT

- https://github.com/ObeOne-KMITL/lab_iot/blob/main/arduino_R4/mqtt.ino

แก้ Code ในส่วนนี้

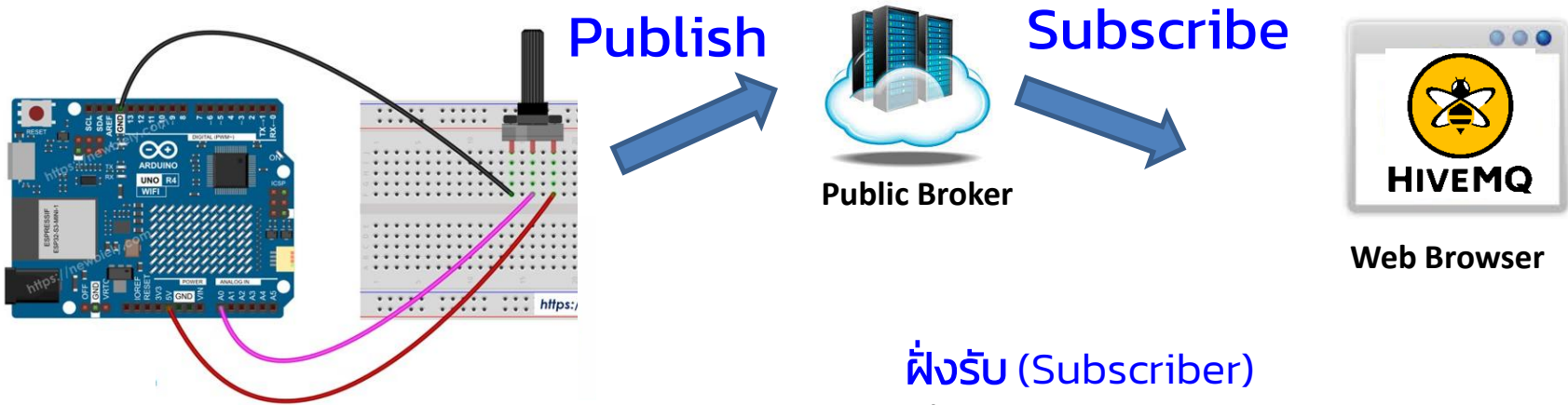
```
const char WIFI_SSID[] = "kmitl-wifi";      // CHANGE TO YOUR WIFI SSID
const char WIFI_PASSWORD[] = "kmitl";      // CHANGE TO YOUR WIFI PASSWORD

const char MQTT_BROKER_ADDRESS[] = "mqtt-dashboard.com"; // CHANGE TO MQTT BROKER'S ADDRESS
//const char MQTT_BROKER_ADDRESS[] = "192.168.0.11"; // CHANGE TO MQTT BROKER'S IP ADDRESS
const int MQTT_PORT = 1883;
const char MQTT_CLIENT_ID[] = "arduino-uno-r4-panwit"; // CHANGE IT AS YOU DESIRE
const char MQTT_USERNAME[] = "";                // CHANGE IT IF REQUIRED, empty if not required
const char MQTT_PASSWORD[] = "";                // CHANGE IT IF REQUIRED, empty if not required

// The MQTT topics that Arduino should publish/subscribe
const char PUBLISH_TOPIC[] = "arduino-uno-r4-panwit/send"; // CHANGE IT AS YOU DESIRE
const char SUBSCRIBE_TOPIC[] = "arduino-uno-r4-panwit/receive"; // CHANGE IT AS YOU DESIRE
```

การทดลองที่ 3. Board : Publish -> Web : Subscribe

AnalogRead ตัวต้านทานปรับค่าได้ โดยขากลางต่อเข้ากับขา A0



Note!!!

ฝั่งส่ง (Publisher)

ส่วนของโปรแกรมที่ใช้ส่งคือ

```
int val = analogRead(A0);  
String val_str = String(val);  
char messageBuffer[10];  
val_str.toCharArray(messageBuffer, 10);  
mqtt.publish(PUBLISH_TOPIC, messageBuffer);
```

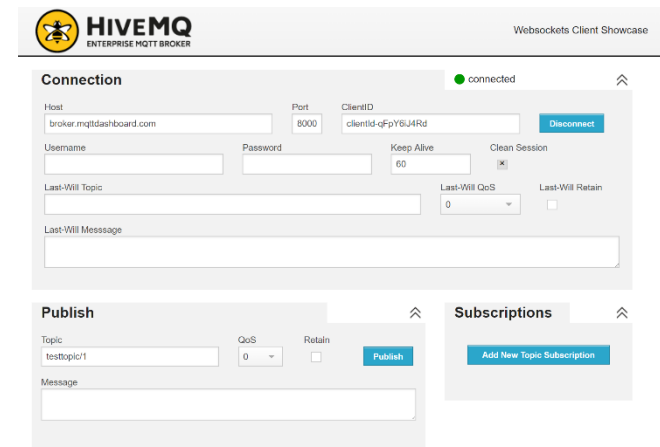
หลักการคือ แปลค่า Integer ให้เป็น String ก่อน
จากนั้นเปลี่ยนจาก String ให้เป็น Array

ฝั่งรับ (Subscriber)

เปิด website

<http://www.hivemq.com/demos/websocket-client/>

ทำการเชื่อมต่อ , ทำการ Subscribe และสังเกตค่าที่ได้



การทดลองที่ 4. Web : Publish -> Board : Subscribe



Websockets Client Showcase

Connection connected

Host: broker.mqttdashboard.com Port: 8000 ClientID: clientid-qfY8U4Rd Disconnect

Username: Password: Keep Alive: 60 Clean Session: ☒ Last-Will Topic: Last-Will QoS: 0 Last-Will Retain: ☐ Last-Will Message:

Publish

Topic: testtopic/1 QoS: 0 Retain: ☐ Publish

Message:

Subscriptions

Add New Topic Subscription

Publish

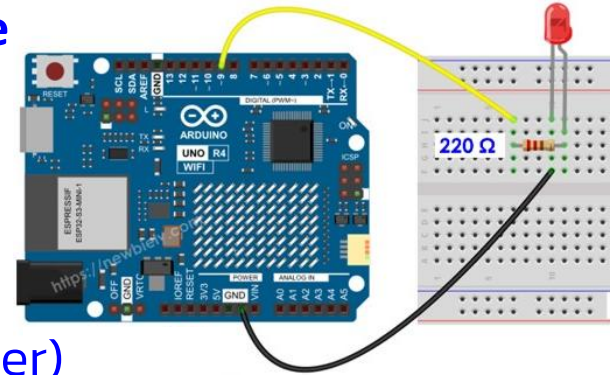
Public Broker



Subscribe

AnalogWrite

ต่อตัวต้านทาน + LED เข้าขา 9



ฝั่งส่ง (Publisher)

ให้ publish ค่า 0-255
แล้วไปสั่งให้ หลอด LED
หรือ สว่าง ตามค่าที่ส่งมา

0 คือ ดับ

255 คือ สว่างที่สุด

ฝั่งรับ (Subscriber)

ในฟังก์ชัน messageReceived

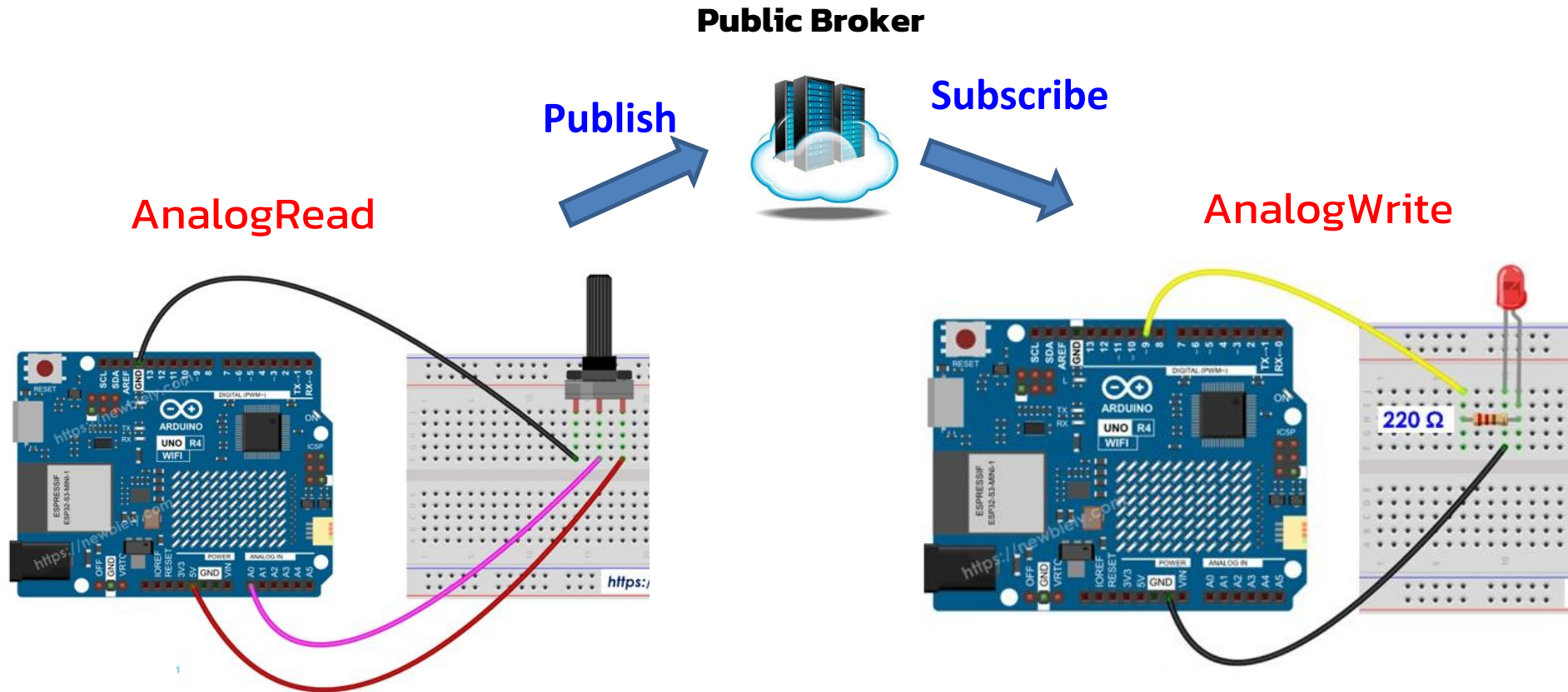
void messageReceived (String &topic,
String &payload)

เพิ่ม Code โดยจะต้องเปลี่ยน payload จาก String ให้เป็น
Integer ก่อน

```
val = payload.toInt();  
analogWrite(9, val);
```


การทดลองที่ 5. ส่งรีไฟผ่าน Internet

ให้ นศ จับคู่กัน



การทดลองอื่นๆ

- จับคู่กัน

- คนส่ง (Publisher) ทำการ publish ส่งค่าอุณหภูมิ
- คนรับ (Subscriber) นำค่า อุณหภูมิที่ได้รับไปแสดงออกจอ LCD หรือ ออก RGB Color LED ที่มีสีตามอุณหภูมิ ร้อน : แดง เย็น : เขียว

- จับคู่กัน

- คนส่ง (Publisher) ทำการ publish ค่าจาก Switch 3 ตัว
- คนรับ (Subscriber) นำค่าที่ได้จากไปสั่ง ปิด-เปิด RGB LED
 - เมื่อรับค่าจาก Switch 1 ให้แสดง RED
 - เมื่อรับค่าจาก Switch 2 ให้แสดง GREEN
 - เมื่อรับค่าจาก Switch 3 ให้แสดง BLUE

กรณีเกิด bug

- Code ตัวอย่างที่ให้ไป ถูกต้องแล้ว
- หลายคนทดลองแล้ว มันต่อ broker ไม่ติด มันขึ้น
- ให้แก้ Version ของ Board manager ให้เป็นตามรูป

