MQTT协议详解

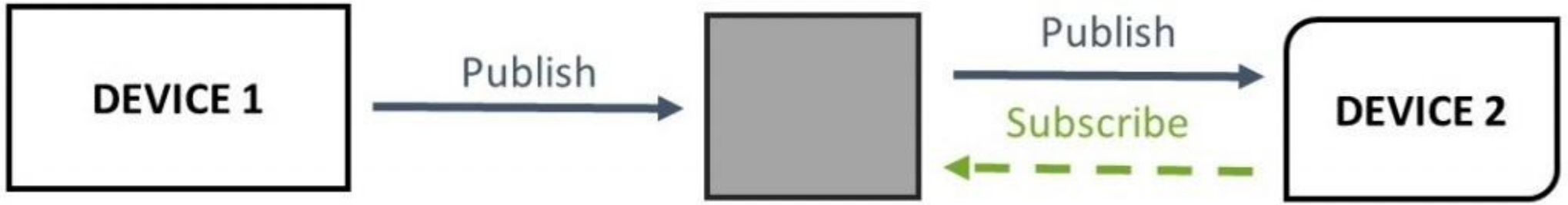
中科院云计算中心大数据研究院

@rh01

Dec 8th, 2018

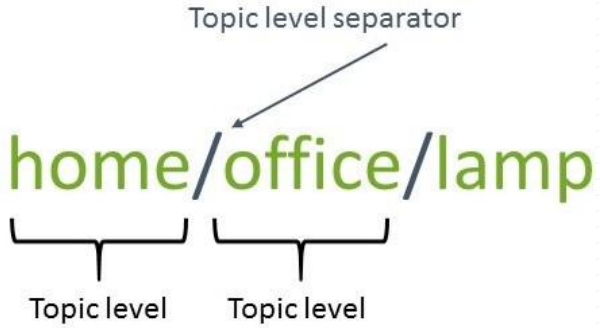
MQTT 的一些概念

- 发布/订阅模型

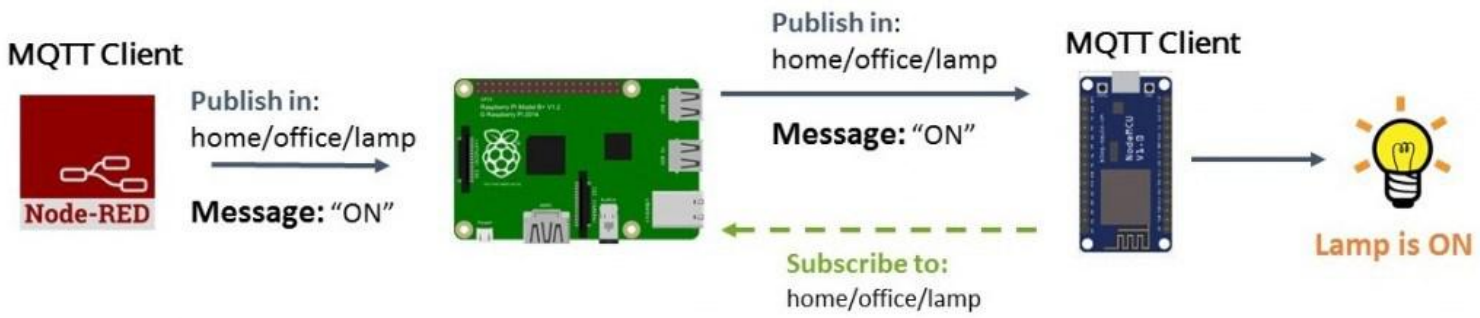


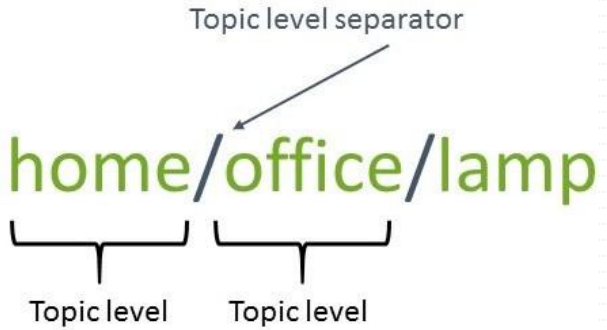
比如，设备1 在主题（topic）上发布（publish）了一个消息，设备2订阅（subscribe）了该主题，那么设备2将会收到设备1在主题上发布的消息

- 消息
  - 消息可以是命令或者数据
- 主题（Topic）
  - 主题是你将消息发布的地方
  - 主题使用“/”来分隔，“/”表示主题的级别
  - 一般情况下是目的性命名
    - 比如：Home，Company,...



- 比如你想在你家里面的办公室（home/office）中打开一盏灯（lamp），那么你可以使用手机或者其他客户端，向 /home/office/lampf 发布一条消息 **“ON”**





- 注意：主题是大小写敏感的

- MQTT – Broker
  - 代理
  - 主要负责接受消息、过滤消息、路由消息等
  - 接下来，我们将在树莓派上使用Mosquitto broker.

大小写敏感

深入 MQTT Topics

- Topic的组织方式/命名方式类似于文件系统，使用“/”来分隔
- Topic的特点：
  - 大小写敏感
  - 使用UTF-8字符串
  - 至少包含一个字符
- $SYS topic
  - broker创建的特殊主题
  - 对于客户端来说，只读
  - 有特殊的用处
  - **详见：https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics**
- 订阅主题
  - 客户端**可以订阅一个或者多个主题**，订阅多个主题时需要用到**通配符**
  - 通配符
    - # 匹配多级主题
    - + 匹配单个级
  - 注意：通配符若要使用，必须在通配符前加上“/”
    - /house/# ✓ hou# ×

- Topic naming Examples:
  - 可行的主题命名方式
    - /
    - /house
    - house/room/main-light
    - house/room/side-light
    - house/room1/main-light
    - house/room1/alarm
    - house/garage/main-light
    - house/main-door

    house/#
    - house/room1/main-light
    - house/room1/alarm
    - house/garage/main-light

    house/+/main-light
- 在主题上发布消息
  - 客户端**只能在单独的一个主题上**发布消息，因此**不能使用通配符**；
- 主题创建的时间
  - 客户端订阅；
  - 客户端在一个主题上发布一个消息，且保留消息的选项设置为True；
- 主题移除的时间
  - 没有一个客户端订阅broker时，且清除会话选项设置为True；
  - 当客户端连接时，清除会话的选项设置为True时；

Q & A

Q- How do I subscribe to all topics?

A-Subscribe to #

Q- How Do I subscribe to all **$SYS** topics?

A-Subscribe to **$SYS/#**

Q- Should I start my Topic hierarchy with a /.

A- It is not necessary and just adds another level to the structure.

Q- Can I get list of all topics on a broker?

A- Not unless you subscribe to all topics and scan them.

Q- Can I tell who is subscribed to a topic?

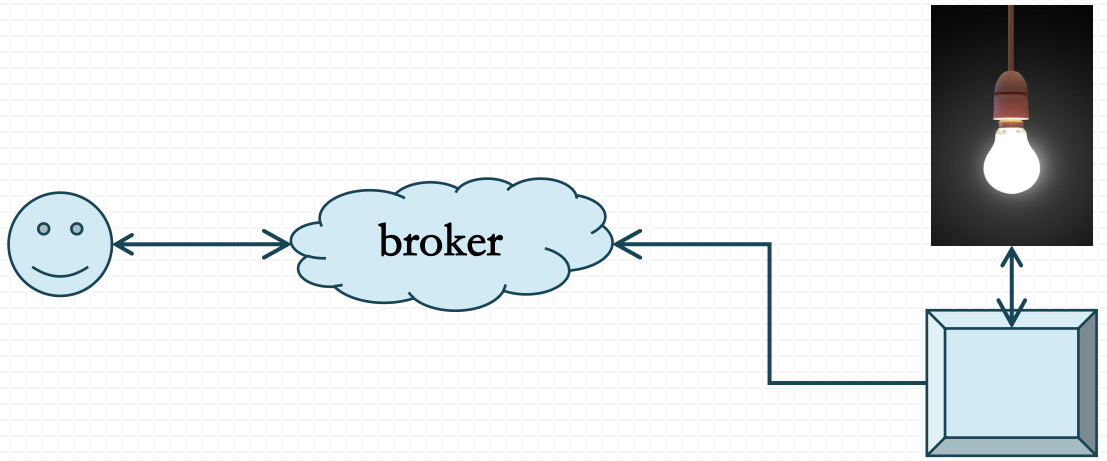
A – No

Q- How do I discover topics?

A- There is currently no mechanism for that except as described in list all topics.

MQTT Topic and Payload Design Notes

- 设计Topic的命名方式和Payload格式是非常重要的
- MQTT Network Devices
  - 我们在现实生活中可能有以下设备：
    - 发送传感器感知到的数据
    - 发送请求数据
    - 接受命令或者控制数据
- 设计
  - 如图，设计远程开关灯的应用
  - Topic Name：
    - light-switch ✓
    - light-switch/status ✓ 用来发送状态数据
    - light-switch/set ✓ 用来发送指令（ON/OFF）



Payload
- { “status” :” ON” } or { “status” :” OFF” } – Data
- { “set” :” ON” } or { “set” :” OFF” } – – Commands
- light-switch

Big Example--Sensors in a Building

- house/sensor1, house/sensor2 …… house/sensor10

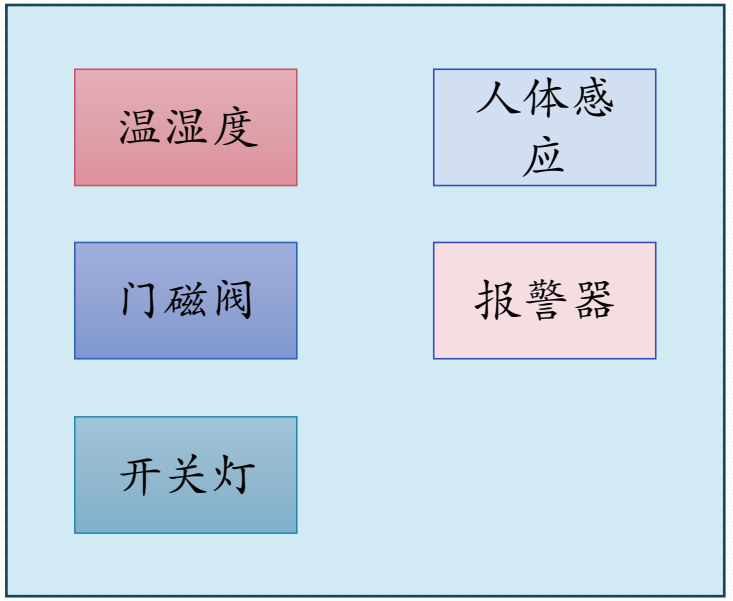
on topic – house/living-room-light
With message payload of – ON or OFF

状态

{ “living-room-light” :” ON” }
{ “living-room-li ght” :” OFF” }

命令

{ “SET” :” ON” }
{ “SET” :” OFF” }



- 为什么使用JSON格式？？
  - 以后为我们搭建IOT平台定义统一的数据格式

Approaches Creating a Topic Naming Scheme

- 两种方式:
  - 尽可能把更多的信息放在主题上
  - 尽可能把更多的信息放在message payload
- 主题的层级参考:
  - High level topic grouping devices.- optional
  - Assigned Sensor name – optional
  - function e.g. status,set,get,cmd –optional
- Message Payload:
  - Payload data
  - Sensor ID- optional
  - Function – optional
- 更标准的命名方式参考:
  - https://github.com/mqtt-smarthome/mqtt-smarthome/blob/master/Architecture.md

Packaging Topic data

- 易读性好

Test/AppSpin/read false
Test/Position3/read false
Test/TankEmpty/read true
Test/AppOpen/read false
Test/Position7/read false

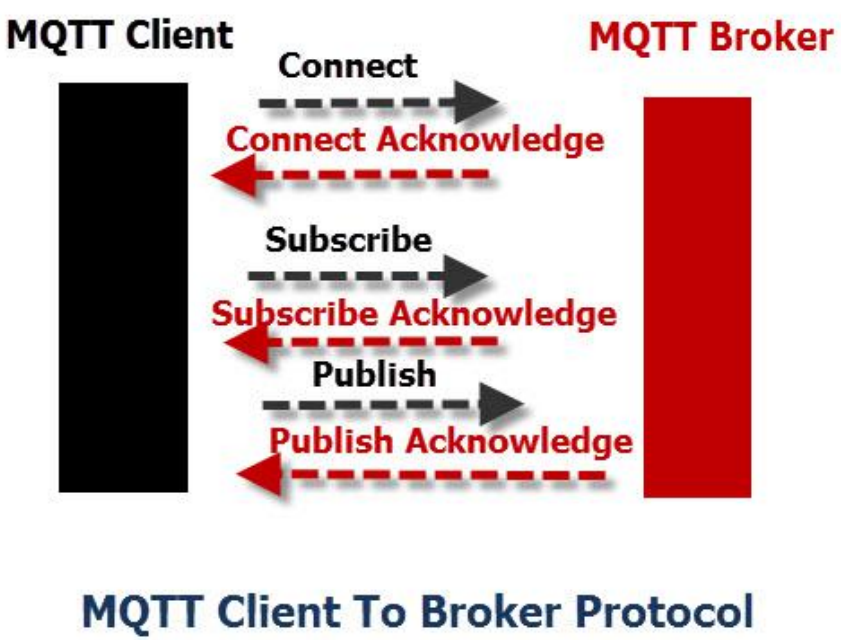
{AppSpin_read:False,Position3_read:False,TankEmpty_read:True,AppOpen_read:False, Position7_read:False}

Video: Publishing and Receiving JSON data with Python.

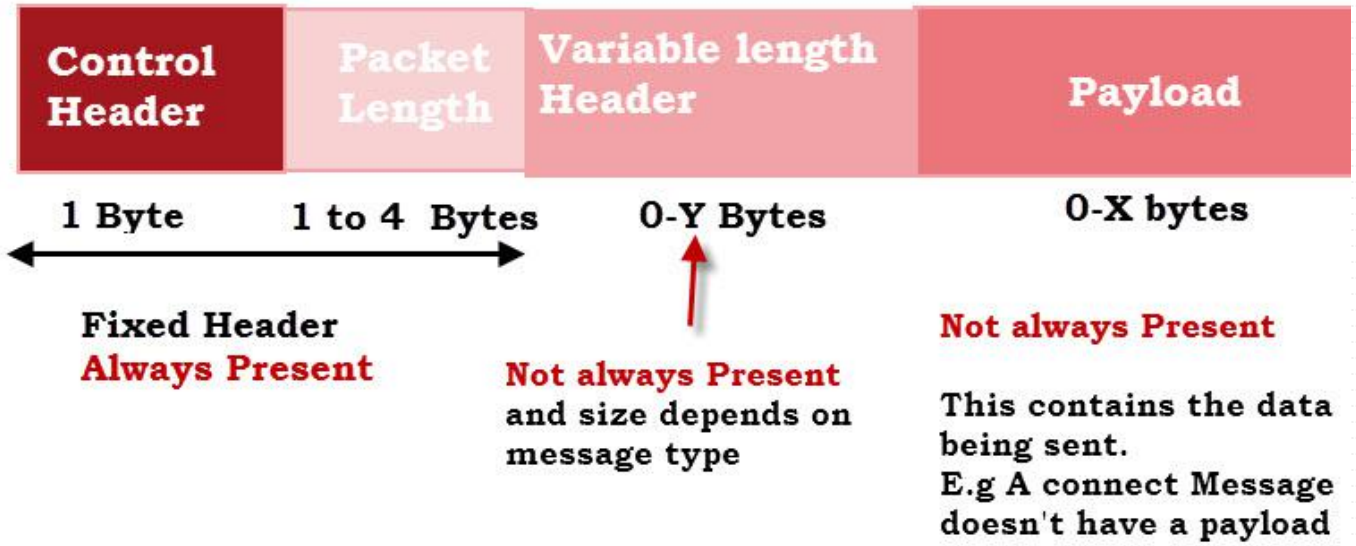
https://www.youtube.com/watch?v=92lxSd7ejeY

Understanding the MQTT Protocol Packet Structure

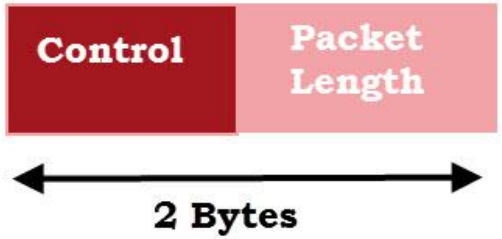
- The MQTT message format.
- The MQTT message header
- Message fields and coding
- Control Message coding example



MQTT Client To Broker Protocol

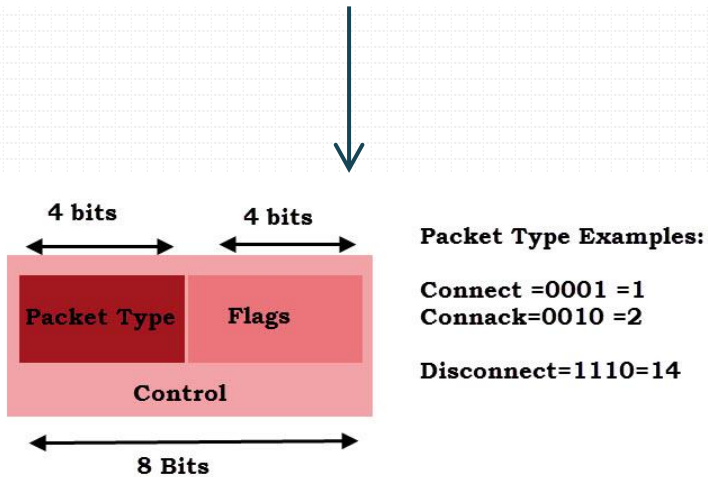


MQTT Standard Packet Structure



Example a disconnect message
Code sent = b'\xe0\x00'

MQTT Minimum Packet



Packet Type Examples:

Connect =0001 =1
Connack=0010 =2

Disconnect=1110=14

MQTT Control Field Structure

Sample MQTT Control Message

Name	Value	Direction of flow	Description	Decimal
Reserved	0	Forbidden	Reserved	
CONNECT	1	Client to Server	Client request to connect to Server	16
CONNACK	2	Server to Client	Connect acknowledgment	32
PUBLISH	3	Client to Server or	Publish message	48

=00010000 =16 Dec

=00100000 =32 Dec

Note: taken from the OASIS MQTT 3.1.1 specification document

MQTT Control Flags (Partial)

Table 2.2 - Flag Bits

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP[1]	QoS[2]	QoS[2]	RETAIN[3]
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0

mqtt-v3.1.1-os

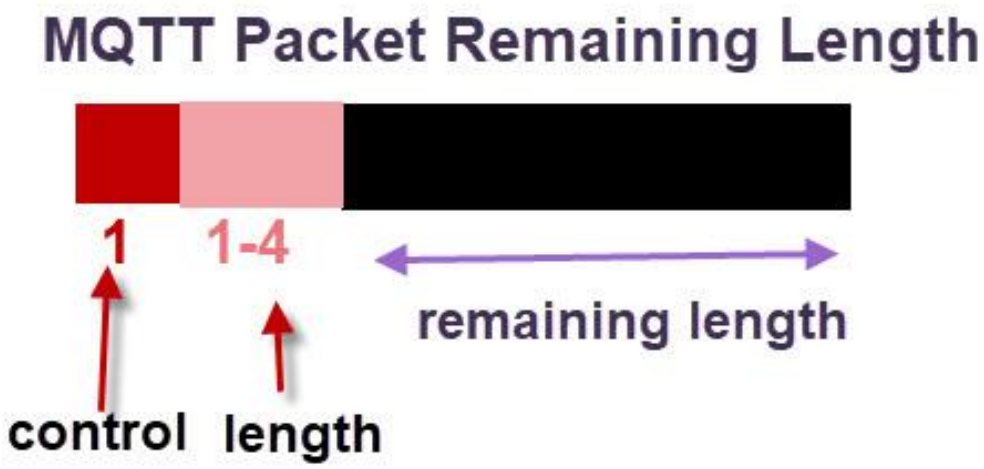
Duplicate message

Retain Message

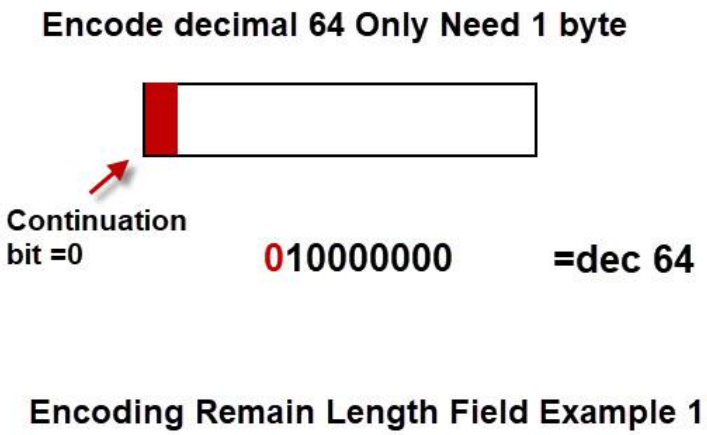
Quality of Service
00,01,10 =QOS 0,1,2

Note: taken from the OASIS MQTT 3.1.1 specification document

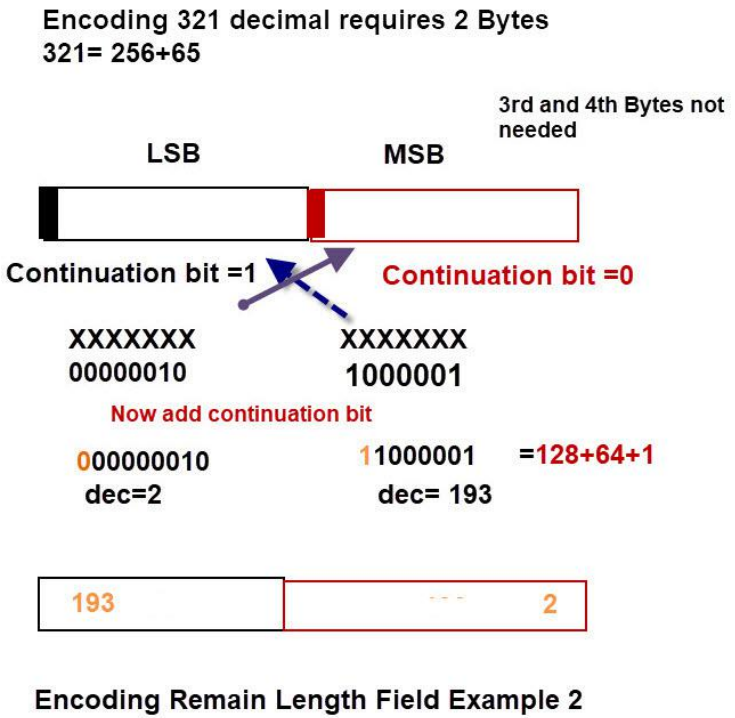
- 保留字段（Remaining Length）
  - This is of variable length between **1 and 4 bytes. Each byte uses 7 bits** for the length with the MSB used as a continuation flag.



Remaining Packet length 64 bytes of requires only 1 byte



Packet length of 321 bytes requires a 2 byte remaining length field



Encoding Remain Length Field Example 2

Remaining Length Field Values

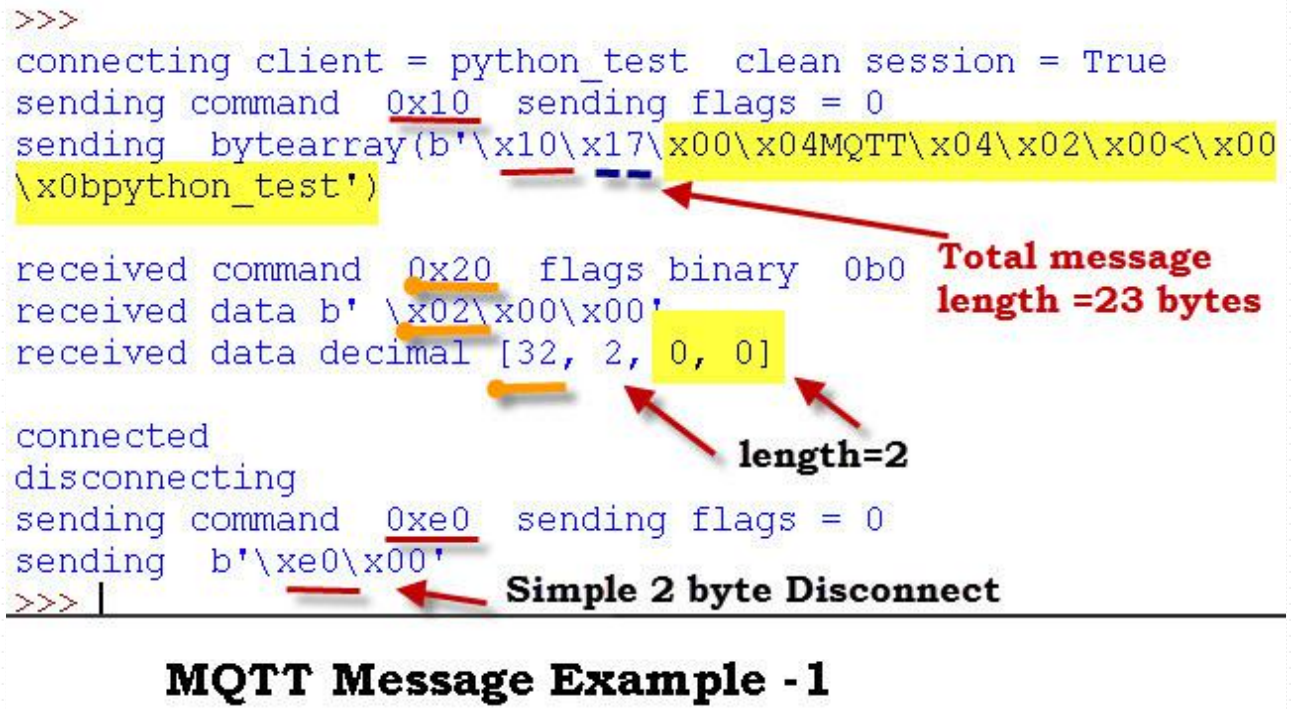
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

Table from Specification for mqtt-v-3.1.1

MQTT Connect and Disconnect Message Example

MQTT Message Types and Hex Codes

```
# Message types
CONNECT = 0x10          =16 decimal
CONNACK = 0x20
PUBLISH = 0x30
PUBACK = 0x40
PUBREC = 0x50
PUBREL = 0x60
PUBCOMP = 0x70
SUBSCRIBE = 0x80        =128 decimal
SUBACK = 0x90
UNSUBSCRIBE = 0xA0
UNSUBACK = 0xB0
PINGREQ = 0xC0
PINGRESP = 0xD0
DISCONNECT = 0xE0       =224 decimal
```

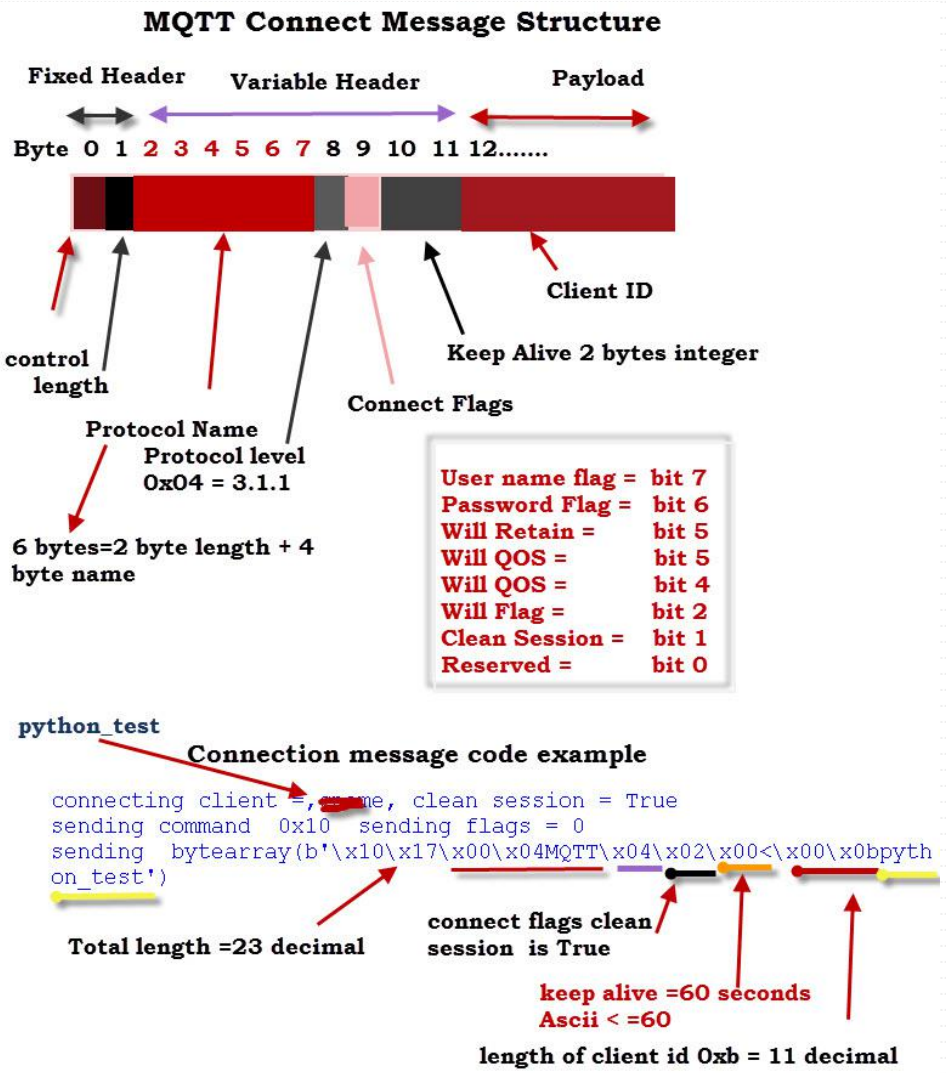


MQTT Message Example -1

- Notice the connection (0x10) and connection acknowledge (0x20) control codes.
- Notice the total length of hex17 or 23 bytes not including the control and length fields. The length field is only 1 byte.
- You should also be able the see the client ID (python_test) in the sent packet.

MQTT packet =control + length + protocol name + Protocol Level +Connect Flags + keep alive +**Payload**

MQTT packet



MQTT packet =control + length + protocol name + Protocol Level +Connect Flags + keep alive +Payload

Paho Python MQTT Client – Understanding Callbacks

- Event **Connection** acknowledged Triggers the **on_connect** callback
- Event **Disconnection** acknowledged Triggers the **on_disconnect** callback
- Event **Subscription** acknowledged Triggers the **on_subscribe** callback
- Event **Un-subscription** acknowledged Triggers the **on_unsubscribe** callback
- Event **Publish** acknowledged Triggers the **on_publish** callback
- Event **Message Received** Triggers the **on_message** callback
- Event **Log information** available Triggers the **on_log** callback

Callback Function Example- simply logs a message, and sets a flag.

```
def on_connect(client, userdata, flags, rc):
     logging.info("Connected flags"+str(flags)+"result code "\
         +str(rc)+"client1_id ")
     client.connected_flag=True

client.on_connect= on_connect  # associate my function with the On_connect callback
```

Callbacks and the Client Loop

回调取决于客户端循环，因为没有循环，不会触发回调。

```
import paho.mqtt.client as mqtt  #import the client
import time

def on_connect(client, userdata, flags, rc):
    global loop_flag
    print("  In on_connect callback ")
    loop_flag=0

broker_address="192.168.1.184"
#broker_address="iot.eclipse.org"
client = mqtt.Client()      #create new instance
client.on_connect=on_connect        #attach function to callback
client.connect(broker_address)      #connect to broker
client.loop_start()         #start loop to process callbacks

loop_flag=1
counter=0
while loop_flag==1:
    print("waiting for callback to occur ",counter)
    time.sleep(.01)  #pause 1/100 second
    counter+=1


client.disconnect()
client.loop_stop()
```

Break from Loop

```
>>>
waiting for callback to occur  0
waiting for callback to occur  1
waiting for callback to occur  2
waiting for callback to occur  3
waiting for callback to occur  4
waiting for callback to occur  5
  In on_connect callback
...
```

Ends here

当客户端连上broker时，会触发回调函数的执行，然后跳出客户端的循环.

如果注释掉这条语句会怎么样?

```
client.connect(broker_address)      #connect to broker
#client.loop_start()        #start loop to process callbacks

loop_flag=1
counter=0
```

Comment Out loop to see what happens

How Callbacks Work

Callbacks

on_connect()

```
on_connect(client, userdata, flags, rc)
```

Called when the broker responds to our connection request.

Asynchronous or Synchronous?

Notice: the `if on_connect` statement. This is what checks for the callback, and if it exists it calls the callback- `self.on_connect()`.

Python MQTT Client Callbacks Illustration

```
def _handle_connack(self):
    Note: I've removed the code that was here as
    not relevant to discussion
    self._easy_log(MQTT_LOG_DEBUG, "Received CONNACK ("+str
(flags)+", "+str(result)+")")
    self._callback_mutex.acquire()
    if self.on_connect:                          <-- Is callback assigned
        self._in_callback = True

        if sys.version_info[0] < 3:
            argcount = self.on_connect.func_code.co_argcount
        else:
            argcount = self.on_connect.__code__.co_argcount

        if argcount == 3:
            self.on_connect(self, self._userdata, result)
        else:
            flags_dict = dict()
            flags_dict['session present'] = flags & 0x01
            self.on_connect(self, self._userdata, flags_dict, result)
        self._in_callback = False
```

This is were the callback gets called and self.on_connect points to our function

Returning Values from Callbacks

- 由于它们的运行方式使用标准return命令无法从回调函数中返回值。因此，如果需要从回调函数中获取状态信息，则需要在回调中使用某种形式的全局变量。

```
import paho.mqtt.client as mqtt #import mqqt client
### extend main class to include flags etc
mqtt.Client.bad_connection_flag=False
mqtt.Client.connected_flag=False
mqtt.Client.disconnect_flag=False
mqtt.Client.disconnect_time=0.0
mqtt.Client.pub_msg_count=0

client=mqtt.Client("myclient") #create client object

client.client.connected_flag=True #set flag to true somewhere in script
```

DEMO

Client

```
# coding:utf-8
# MQTT Client demo
# Continuously monitor two different MQTT topics for data,
# check if the received data matches two predefined 'commands'

import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() - if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("CoreElectronics/test")
    client.subscribe("CoreElectronics/topic")
```

```
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

    if msg.payload == "Hello":
        print("Received message #1, do something")
        # Do something

    if msg.payload == "World!":
        print("Received message #2, do something else")
        # Do something else

# Create an MQTT client and attach our routines to it.
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("172.16.3.16", 1883, 60)

# Process network traffic and dispatch callbacks. This will also handle
# reconnecting. Check the documentation at
# https://github.com/eclipse/paho.mqtt.python
# for information on how to use other loop*() functions
client.loop_forever()
```

Server

```
import paho.mqtt.publish as publish

publish.single("CoreElectronics/test", "close", hostname="172.16.3.16")
publish.single("CoreElectronics/topic", "World!", hostname="172.16.3.16")
print("Done")
```

高级版本

```
# coding:utf-8
# MQTT Client demo
# Continuously monitor two different MQTT topics for data,
# check if the received data matches two predefined 'commands'

import paho.mqtt.client as mqtt
import time
import RPi.GPIO as GPIO
import os

led1 = 16

flag = False
# warning function, used to control led
```

```
# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))


    # Snitialize GPIO
    GPIO.setmode(GPIO.BCM) #使用BCM编码方式
    #设置引脚为输入和输出
    GPIO.setwarnings(False)
    #设置23针脚为输入，接到红外避障传感器模块的out引脚
    GPIO.setup(led1,GPIO.OUT)


    #subscribing in on_connect() - if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("CoreElectronics/test")
    client.subscribe("CoreElectronics/topic")

# The callback for when disconnection acknowledged Triggers from server
def on_disconnect(client, userdata, flags,rc ):
    GPIO.cleanup()
```

```
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

    if msg.payload == "Hello":
        print("Received message #1, do something")
        # flag boolean, open led or not.
         GPIO.output(led1,GPIO.HIGH)
        # Do something
    if msg.payload == "close":
        print("close the led")
        GPIO.output(led1,GPIO.LOW)
    if msg.payload == "World!":
        print("Received message #2, do something else")
        # Do something else
# Create an MQTT client and attach our routines to it.
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("172.16.3.16", 1883, 60)

# Process network traffic and dispatch callbacks. This will also handle
# reconnecting.
client.loop_forever()
```

Thank you!!!

@rh01

Dec 7th, 2018