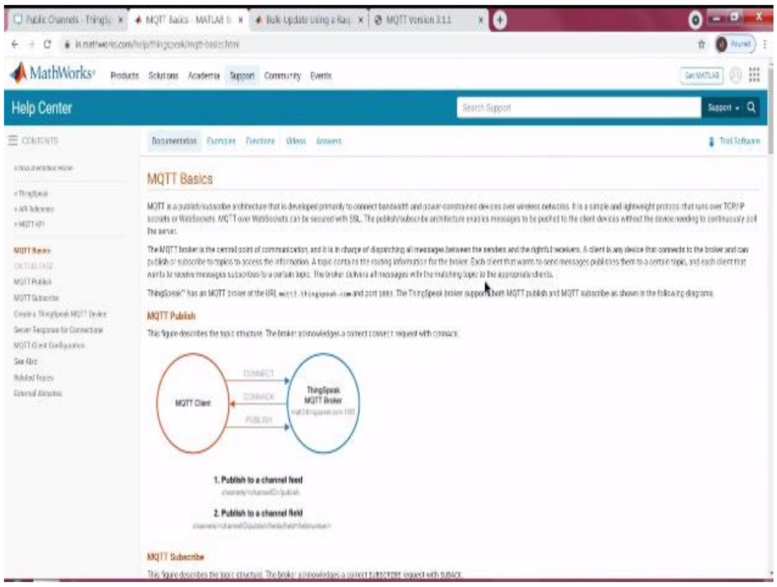
Design for Internet of Things

Prof. T V Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science-Bengaluru

Lecture - 35
MQTT – 02

This area of protocols can be very dry if you do not try by yourself. And it is very exciting if you try things by yourself, okay. So let me take you to an exciting area, an exciting website, which you can try. And that brings me to this website, okay.

(Refer Slide Time: 00:49)



MQTT Basics

MQTT is a publish/subscribe architecture that is developed primarily to connect bandwidth and power-constrained devices over wireless networks. It is a simple and lightweight protocol that runs over TCP/IP sockets or WebSockets. MQTT over WebSockets can be secured with SSL. The publish/subscribe architecture enables messages to be pushed to the client devices without the device needing to continuously poll the server.

The MQTT broker is the central point of communication, and it is in charge of dispatching all messages between the senders and the rightful receivers. A client is any device that connects to the broker and can publish or subscribe to topics to access the information. A topic contains the routing information for the broker. Each client that wants to send messages publishes them to a certain topic, and each client that wants to receive messages subscribes to a certain topic. The broker delivers all messages with the matching topic to the appropriate clients.

ThingSpeak™ has an MQTT broker at the URL mqtt.thingspeak.com and port 1883. The ThingSpeak broker supports both MQTT publish and MQTT subscribe as shown in the following diagrams.

MQTT Publish

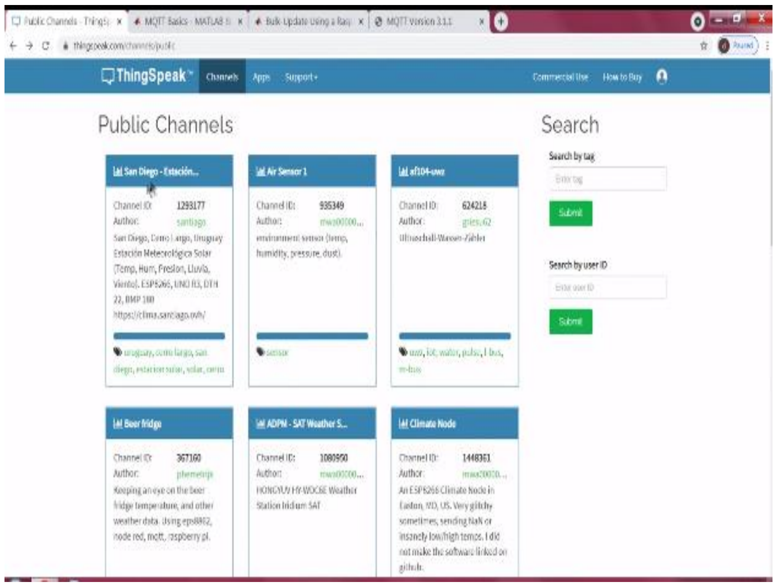
This figure describes the topic structure. The broker acknowledges a correct CONNECT request with CONNACK.

MQTT Subscribe

This figure describes the topic structure. The broker acknowledges a correct SUBSCRIBE request with SUBACK.

This is what is known as the website related to ThingSpeak, okay.

(Refer Slide Time: 00:54)



Public Channels

San Diego - Estación...	Air Sensor 1	af104-uwz
Channel ID: 1293177	Channel ID: 935349	Channel ID: 624218
Author: santiago	Author: mwa00000...	Author: gries.62
San Diego, Como Largo, Uruguay Estación Meteorológica Solar (Temp, Hum, Presion, Lluvia, Viento). ESP8266, UNO R3, DTH 22, BMP 180 https://clima.santiago.ovh/	environment sensor (temp, humidity, pressure, dust).	Ultraschall-Wasser-Zähler
uruguay, como largo, san diego, estacion solar, solar, como	sensor	uwz, iot, water, pulse, l bus, m-bus
Beer fridge	ADPN - SAT Weather S...	Climate Node
Channel ID: 367160	Channel ID: 1080950	Channel ID: 1448361
Author: phemetrip	Author: mwa00000...	Author: mwa00000...
Keeping an eye on the beer fridge temperature, and other weather data. Using esp8862, node red, mqtt, raspberry pi.	HONGYUV HY-WDC6E Weather Station Iridium SAT	An ESP8266 Climate Node in Easton, MD, US. Very glitchy sometimes, sending NaN or insanely low/high temps. I did not make the software linked on github.

Search

Search by tag: Enter tag [Submit]

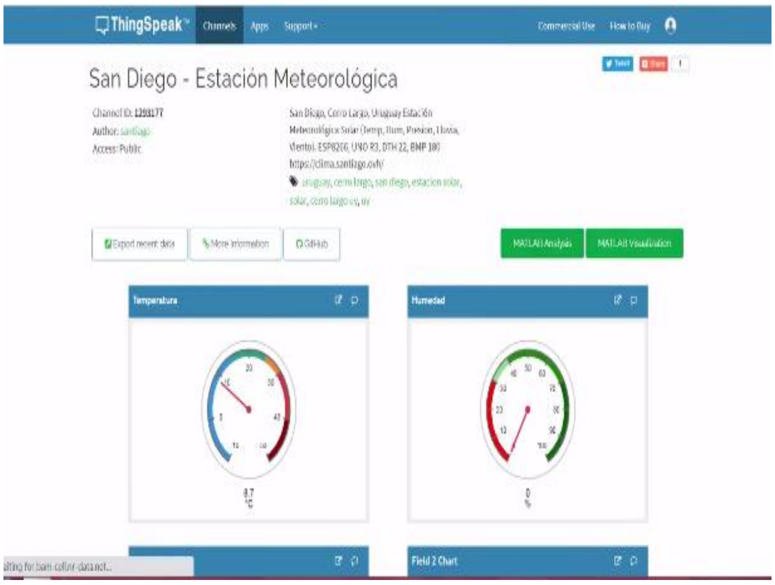
Search by user ID: Enter user ID [Submit]

This thing speak has what are known as public channels from where you can download data, which was uploaded by different MQTT clients. Several of them gathered the environment data and put it up on a free cloud based broker, okay. And now the data is available on the broker, you can connect to that broker and download the data and build your own application.

Remember what I mentioned about the blinds, window and blinds application, how new applications can be built. Quite like that, you can build your own applications. What are those free channels, public channels? You have San Diego some information, okay. Then you have air sensor data. You have some other, some channels. Then, you can look up. There is a weather climate node.

Then there is a weather station data and so on and so forth. Air quality data, carbon dioxide data okay, and so on. So you can download this data and build your own applications. Live, you can connect to the internet to ThingSpeak.com channels public, and you can actually download. So you click on this, let us say this website.

(Refer Slide Time: 02:19)



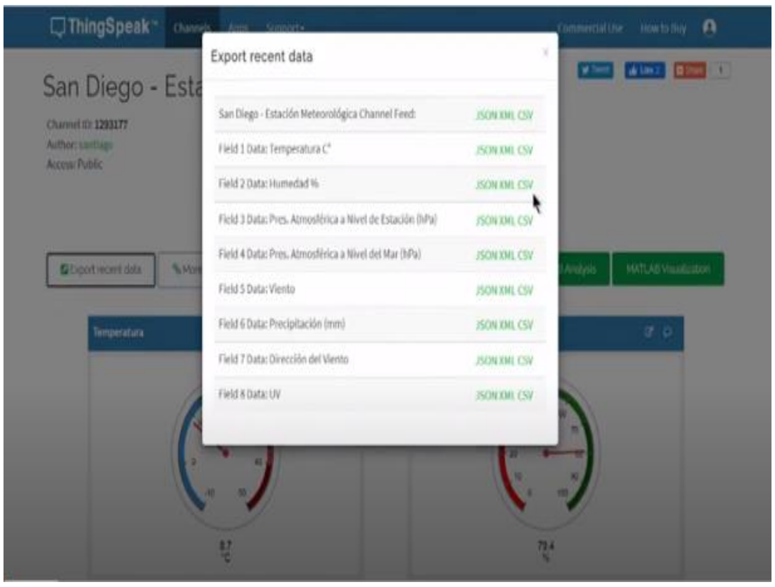
San Diego - Estación Meteorológica

Channel ID: 1293177
Author: santiago
Access: Public

San Diego, Cerro Largo, Uruguay Estación Meteorológica Solar (Temp, Hum, Presion, Lluvia, Viento). ESP8266, UNO R3, DTH 22, BMP 180
https://clima.santiago.ovh/
uruguay, cerro largo, san diego, estacion solar, solar, cerro largo uy, uy

You can see that live data is actually available to you. Okay, this is in some language, which perhaps difficult for you to interpret. It must be Spanish. So anyway, so this information is available, okay. Supposing you want to download this data, you click on this button.

(Refer Slide Time: 02:39)


Export recent data

San Diego - Estación Meteorológica Channel Feed:	JSON XML CSV
Field 1 Data: Temperatura C°	JSON XML CSV
Field 2 Data: Humedad %	JSON XML CSV
Field 3 Data: Pres. Atmosférica a Nivel de Estación (hPa)	JSON XML CSV
Field 4 Data: Pres. Atmosférica a Nivel del Mar (hPa)	JSON XML CSV
Field 5 Data: Viento	JSON XML CSV
Field 6 Data: Precipitación (mm)	JSON XML CSV
Field 7 Data: Dirección del Viento	JSON XML CSV
Field 8 Data: UV	JSON XML CSV

You can get it in CSV, XML, JSON, right? CSV is comma delimited file. XML and JSON are machine readable formats, which you can directly build. You can take it into another machine. You can write a program for one machine to suck this data and consume this data without humans. If you do CSV, you are stuck. You have to download, you are to open Excel or any one of these spreadsheet applications and you can plot it.

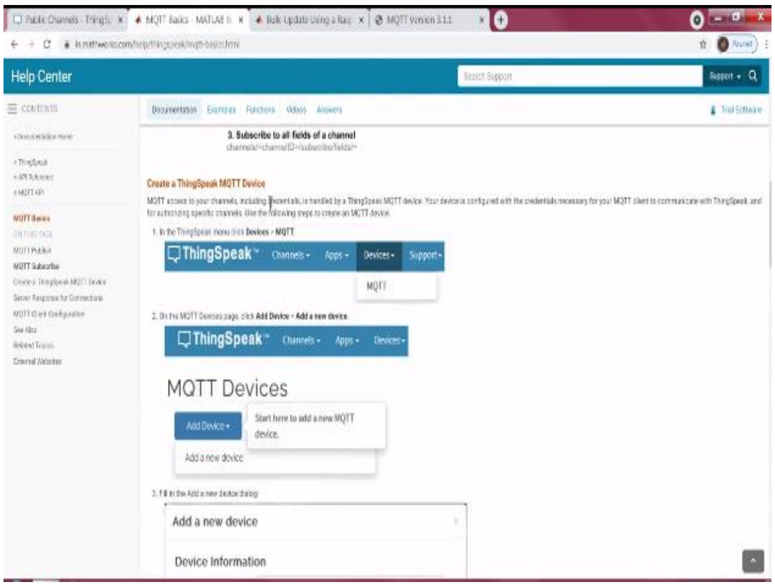
That is a human consumption. But supposing you want sensors to consume this data, because you said IoT paradigm is sensor talking to another sensor. Data traffic is generated not by humans, but from machines for machines, from machines. From machines to for machines. So how will you do that? You put it either in XML or JSON. So machine readable formats are available.

So machines will consume this data and whatever application you can build, you can build it that way. So folks, what is the lesson that you learn? The lesson you learn here is that if you want to make machines consume data, you must have a particular format in which you can download and get machines to download data automatically, and get the data to be consumed automatically. Amazing.

So you can try this on your own. Now what we can do is, see this is something you can build free. You do not need any account, okay. For example, you want to build, you can do this. You can also do a few things for free. So you should try that, okay. For example, you can create some free channels. And you can use those free channels.

So this website actually is supported by MathWorks, which is known for the MATLAB software. And this whole thing is actually set up by them for academic purposes, for education purposes. Students, if they have a MathWorks account, can freely use this account. And know start learning a little bit about IoT and how MATLAB simulations can be uploaded to cloud and how as you can build major applications and so on. Anyway, so it is from MathWorks. I will come to this basics.

(Refer Slide Time: 05:00)



Help Center

3. Subscribe to all fields of a channel

channels/<channelID>/subscribe/fields/+

Create a ThingSpeak MQTT Device

MQTT access to your channels, including credentials, is handled by a ThingSpeak MQTT device. Your device is configured with the credentials necessary for your MQTT client to communicate with ThingSpeak, and for authorizing specific channels. Use the following steps to create an MQTT device.

1. In the ThingSpeak menu click **Devices > MQTT**.

2. On the MQTT Devices page, click **Add Device > Add a new device**.

MQTT Devices

Start here to add a new MQTT device.

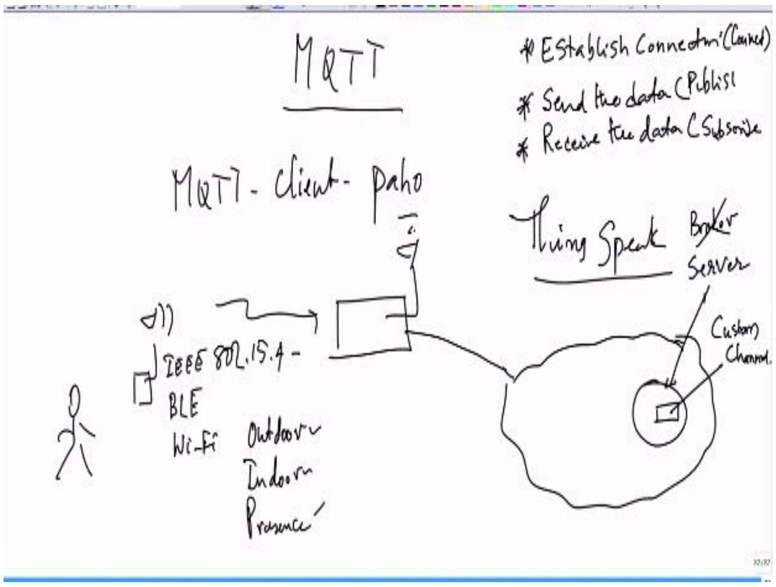
3. Fill in the Add a new device dialog

Add a new device

Device Information

But you can see that you can create ThingSpeak MQTT device.

(Refer Slide Time: 05:08)



In other words, suppose you have a sensor, IoT node sensor okay, which runs MQTT client and I gave you the name, Paho. You can run on a small embedded node. You can run MQTT client, Paho and even including nodes like, you know if you are talking of slightly high computing, you can talk about Raspberry Pi.

Otherwise, you can also talk about small embedded controller boats, which have the capability to talk to the internet directly, or pass the data to a Wi-Fi Bluetooth router and that router in turn getting it data to the MQTT broker. So please note ThingSpeak is a cloud service. And what does it support on the cloud? So I will put it like this somewhere on the internet.

And here inside this internet somewhere here is this ThingSpeak broker. But we will start using the word server. So let us call it MQTT server okay, server. So let me put it properly. And you are sitting here somewhere in your home, office, lab wherever. And you have your sensor node, which supports IEEE802.15.4 or it can support, this is one type of wireless protocol.

Another one is Bluetooth Low Energy. The third one is Wi-Fi. Any one of them it supports. Your lab has a internet router which is connected to the internet, okay. Now your node is gathering some data, let us say movement data or it is collecting outdoor. Go back to the same example. Let us say it is collecting the outdoor light data, not insulation, I would say light, outdoor light.

So we said we will have outdoor light, outdoor, indoor and presence, human presence okay, these three. So you can keep uploading the outdoor light information to your wireless router, which in turn, puts it up on the internet and gets uploaded on one channel which you have created. So I will say custom channel, custom channel. And on this custom channel you are uploading outdoor, indoor and presence data, okay.

And now this data once it is sitting there your lab, home blind has to consume this data because it has to open the blind automatically. Remember, we said two possibilities exist in that application. One is the blind system has a motor actuator, which has a mechanism to move the blind up and down. It has a sensor node and the algorithm is running right on the sensor node.

This is one way. That is what we said in the last class. But why should it be that way? Someone asked in your set of questions you know also that why cannot I do all the compute on the cloud and keep my sensor systems simple, sense and send, sense and send? Why not do all the compute on the cloud? Sure. That is another possibility, okay?

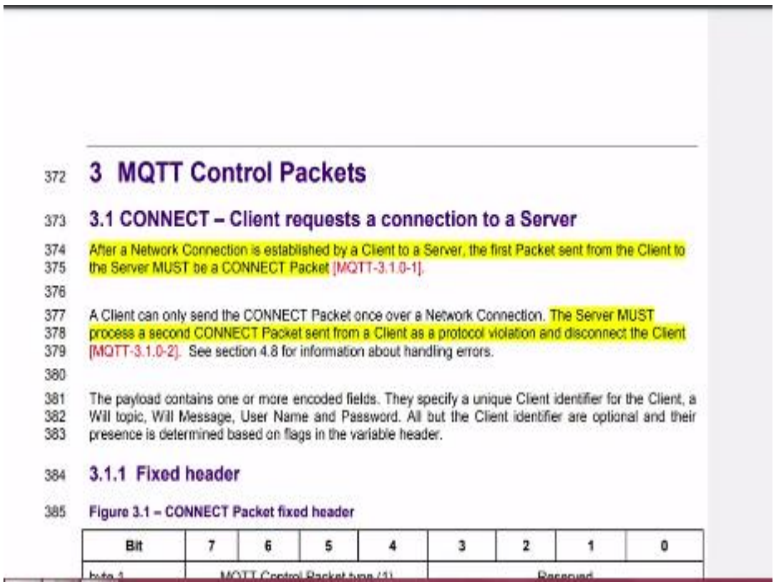
You can resend everything to the cloud and let the cloud compute the amount of opening of the blind and make it available to the actuator system. That is another possibility. But there was also another question by one of you, which said, why do I need internet to enable IoT from my home lab? Someone said that also. Someone, one of you said, why do I need internet to enable these applications.

That is another possibility. You do not need internet. Get your sensor node to transmit the data. Let your actuator node do all the computation, both are possible. Anyway, we are just discussing protocol. So I thought it is important for you to connect everything that we have done in the previous class. So you can do this. Put it onto your custom channel and get the other sensor, blind sensor to download this data.

Now you have to follow MQTT, which means there is a certain protocol. There is a certain way to upload the data. There is a certain way to establish a connection. So one is establish connection. And there is a certain way to send the data. And there is a certain way to receive the data. The way to receive the data is you do a subscribe. The way to send the data is publish.

The way to establish a connection is CONNECT. These are all part of the protocol. Do I need to know a lot about the protocol? Yes, you must know the protocol in detail. Where do I find this information? Well, folks, there is information out there, which is quite easy for you. I will point you to that. Here you are.

(Refer Slide Time: 11:00)



3 MQTT Control Packets

3.1 CONNECT – Client requests a connection to a Server

After a Network Connection is established by a Client to a Server, the first Packet sent from the Client to the Server MUST be a CONNECT Packet [MQTT-3.1.0-1].

A Client can only send the CONNECT Packet once over a Network Connection. The Server MUST process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client [MQTT-3.1.0-2]. See section 4.8 for information about handling errors.

The payload contains one or more encoded fields. They specify a unique Client identifier for the Client, a Will topic, Will Message, User Name and Password. All but the Client identifier are optional and their presence is determined based on flags in the variable header.

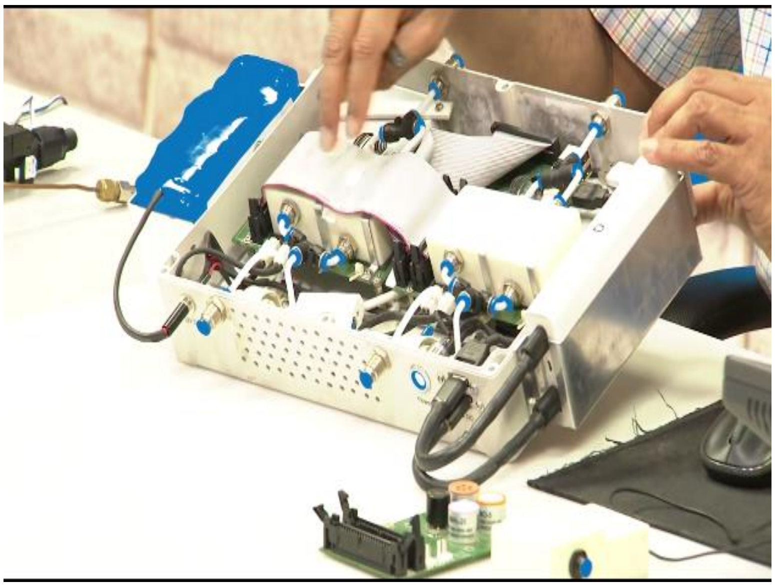
3.1.1 Fixed header

Figure 3.1 – CONNECT Packet fixed header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (1)				Reserved			

CONNECT. Client requires a connection to a server. It is right there. Read this in detail, you will understand how to connect. Now where is the demo? We are still talking about so much of the protocol, where is the demo? Yes, demo will come provided you understand the protocol well. So let me shift now back to the demo.

(Refer Slide Time: 11:21)

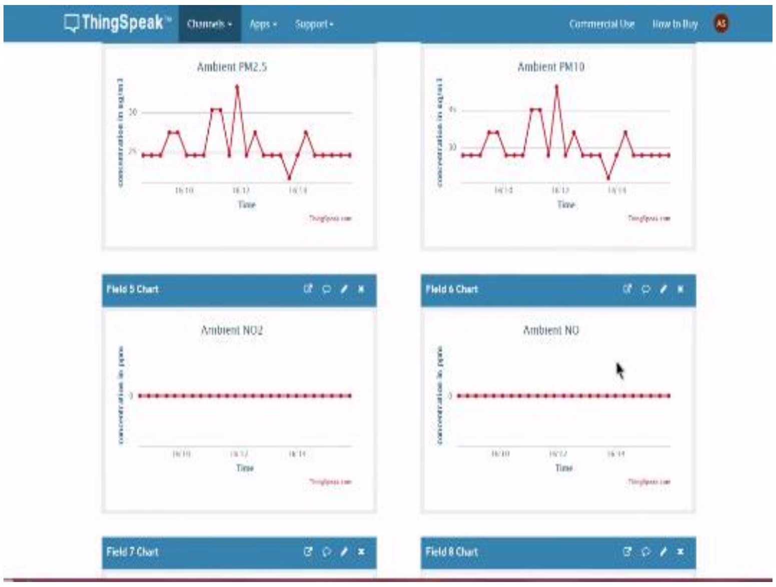


So let me draw your attention to this little box here, which has several components. Maybe I should tilt a little bit in this direction so that it will help you to look up the different components. What this has are two identical units on either side and it is doing air quality check from automobiles. One of them is taking from automobiles and the other is taking from ambient, right?

This is from the automobile part. I think this is from the ambient and this is from the automobile part. And this will take it from the exhaust directly. And this will take from the ambient systems. These sensors here acquire the data by this computer here. This is a Raspberry Pi, this Raspberry Pi computer here. And this in turn, collects the data and uploads using a 4G modem, 3G or a 4G modem.

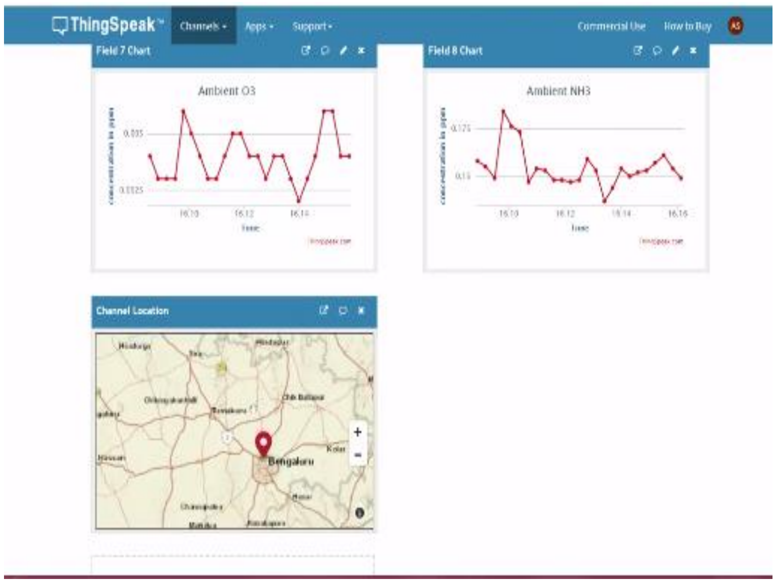
And in order to know where because it is a mobile unit, since you want to know where this unit and where that measurement was, you need a GPS system. So the GPS is here. So you can see this is the GPS, this is the 3G/4G modem and these are some massive batteries for powering this system.

(Refer Slide Time: 12:48)



The hardware that I showed you essentially has monitoring all these parameters. You can see that we are monitoring ambient temperature, we are monitoring ambient humidity, we are monitoring ambient particulate matter 2.5, PM10. Ambient nitrogen dioxide is also measured here. Ambient nitric oxide to the right. What you see here is nitric oxide. What you see here is ambient nitrogen dioxide and let us see what else do you measure.

(Refer Slide Time: 13:22)

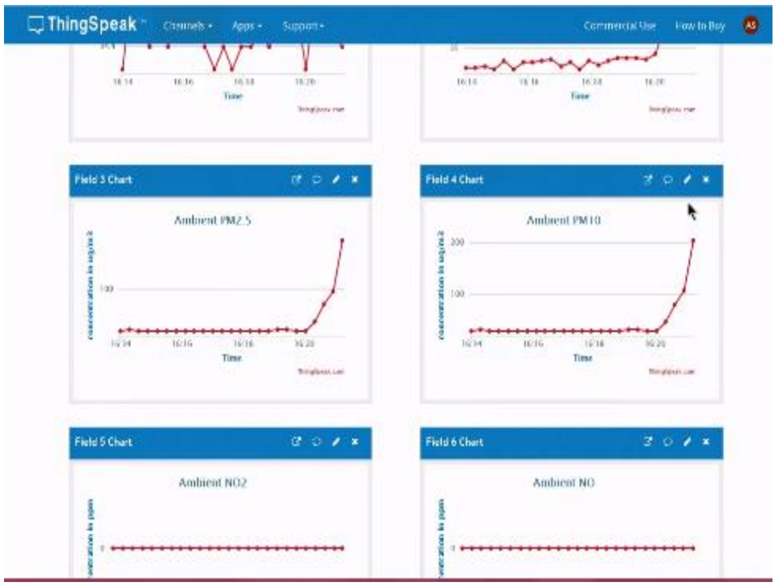


You measure ambient ozone, you measure ambient ammonia, and the location is indicated here as you can see, nicely it is indicating because we have a GPS module inside this system. You see that it is now actually being monitored in real time. If I put my mouse on this, it is actually measuring the value and if I put it on this it is actually putting the value.

Now that we are showing you a live demonstration, live of this data that is coming in, can we invoke some way by which we increase the particulate matter and see whether that corresponding, you do something at the inlet and see if you can increase the particulate matter whether our sensor is sensitive enough to show a slight rise.

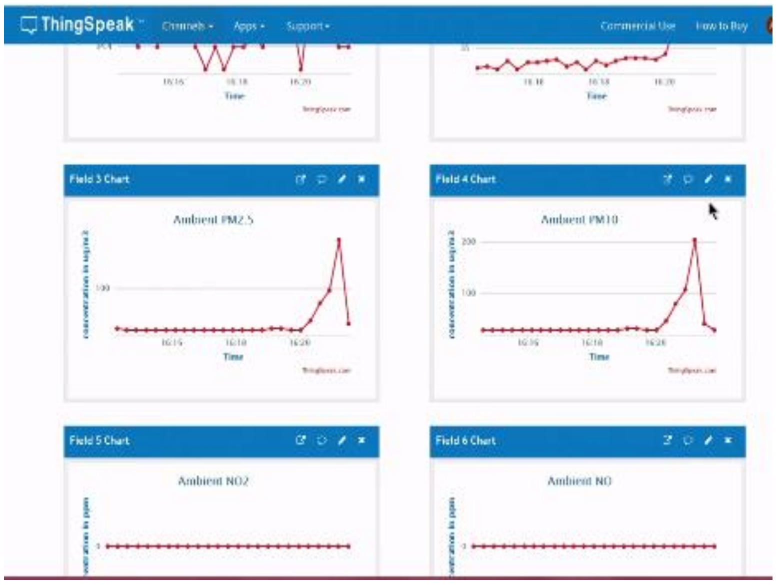
So you can see he has taken some waste, automobile waste, which perhaps has all the ingredients of increasing the particulate matter and then he just gives it as an input inside the system, okay. And the air is absorbed because there is a pump here which will absorb the air. And now let us see what happens. Let us turn our attention to the screen.

(Refer Slide Time: 14:42)



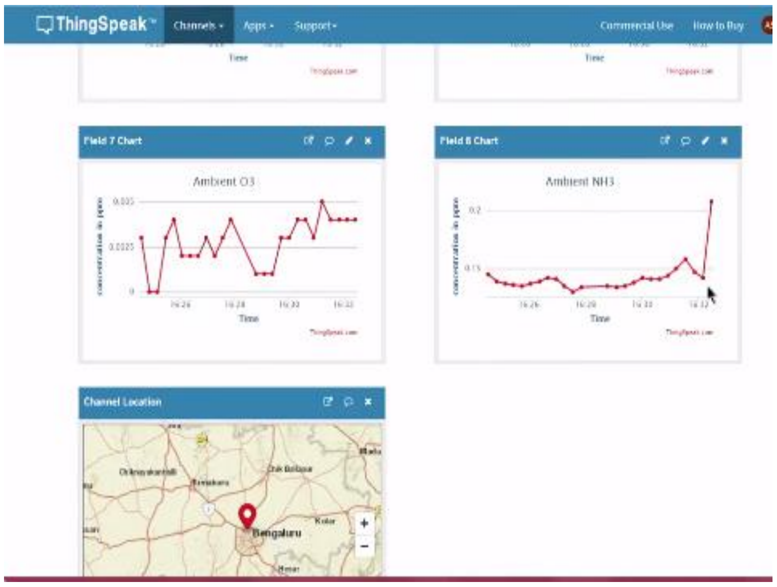
You see quickly that there is a rise in the particulate matter. There is a rise in the particulate matter. So you can see that it has risen quite significantly actually. It has the particulate matter 2.5 has gone quite high as high even close to 200 micrograms per meter cube, and the same with PM10, which has gone up to 200 micrograms per meter cube.

(Refer Slide Time: 15:06)



And why has it come down, because he has stopped. We have stopped feeding polluted air by adding any pollutants to the air that is being sucked by the system here. We can do another demonstration for ammonia as well. So let us now try an ammonia experiment. Look at this live data coming about ammonia.

(Refer Slide Time: 15:34)

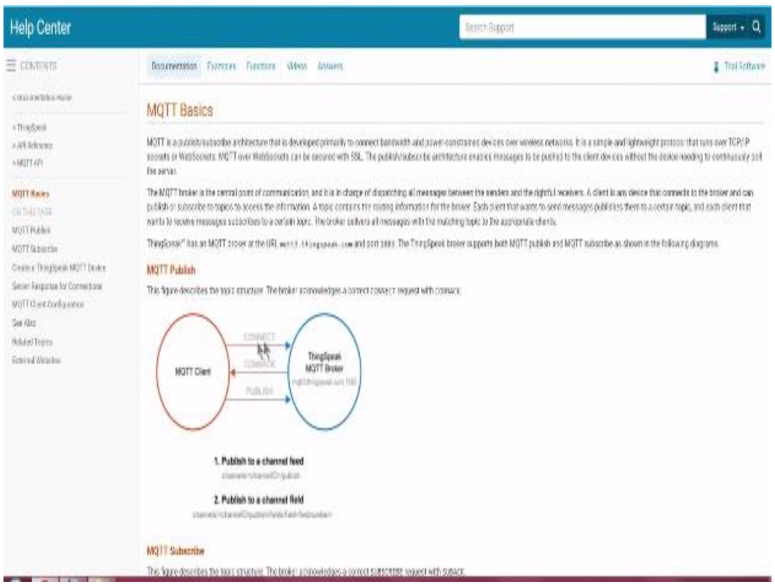


You can see that I brought the bottle close to the device and then it had a shoot. Now our project staff Abishek has moved away and then the value has come down. I will request him to come closer, you will see a spike again. So let us keep watching this. So let us see. Let us put it closer, now it is closer. It should start sensing the ammonia. Now data you can see that he has brought that bottle.

So there is air mixed with ammonia. You can see that there is a rise, right? The air sample is taken, measured by the ammonia sensor and then there is a shoot, okay. Now he has gone away, we have gone away from the system. So it should now drop back into some value. Again here we are measuring in terms of concentration in parts per million ppm. This is one way of measuring.

You can see it has come down. And now the sensor says okay the air measuring is now replaced with fresh air and that fresh air does not have any more ammonia. Okay?

(Refer Slide Time: 16:48)



MQTT Basics

MQTT is a publish/subscribe architecture that is developed primarily to connect bandwidth and power-constrained devices over wireless networks. It is a simple and lightweight protocol that runs over TCP/IP sockets or WebSockets. MQTT over WebSockets can be secured with SSL. The publish/subscribe architecture enables messages to be pushed to the client devices without the device needing to continuously poll the server.

The MQTT broker is the central point of communication, and it is in charge of dispatching all messages between the senders and the rightful receivers. A client is any device that connects to the broker and can publish or subscribe to topics to access the information. A topic contains the routing information for the broker. Each client that wants to send messages publishes them to a certain topic, and each client that wants to receive messages subscribes to a certain topic. The broker delivers all messages with the matching topic to the appropriate clients.

ThingSpeak™ has an MQTT broker at the URL mqtt3.thingspeak.com and port 1883. The ThingSpeak broker supports both MQTT publish and MQTT subscribe as shown in the following diagrams.

MQTT Publish

This figure describes the topic structure. The broker acknowledges a correct CONNECT request with CONNACK.

1. Publish to a channel feed
channels/<channelID>/publish

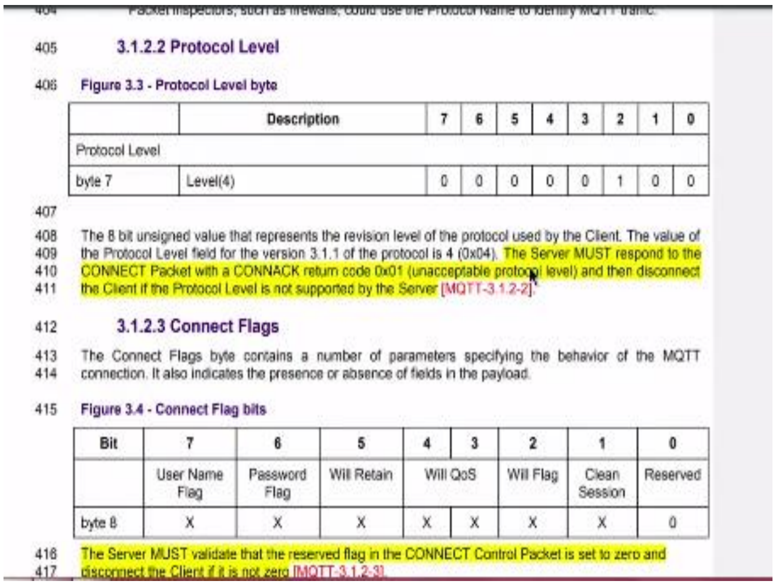
2. Publish to a channel field
channels/<channelID>/publish/fields/field<fieldnumber>

MQTT Subscribe

This figure describes the topic structure. The broker acknowledges a correct SUBSCRIBE request with SUBACK.

Now see what is written here. Very simple to read actually. Way to start uploading data to the cloud is first is to try from an MQTT client like Paho to establish a CONNECT. I want to know more about CONNECT. You know what to do. Open this here, you come to CONNECT, right?

(Refer Slide Time: 17:11)



Packet inspectors, such as firewalls, could use the Protocol Name to identify MQTT traffic.

3.1.2.2 Protocol Level

Figure 3.3 - Protocol Level byte

	Description	7	6	5	4	3	2	1	0
Protocol Level									
byte 7	Level(4)	0	0	0	0	0	1	0	0

The 8 bit unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Level field for the version 3.1.1 of the protocol is 4 (0x04). The Server MUST respond to the CONNECT Packet with a CONNACK return code 0x01 (unacceptable protocol level) and then disconnect the Client if the Protocol Level is not supported by the Server [MQTT-3.1.2-2].

3.1.2.3 Connect Flags

The Connect Flags byte contains a number of parameters specifying the behavior of the MQTT connection. It also indicates the presence or absence of fields in the payload.

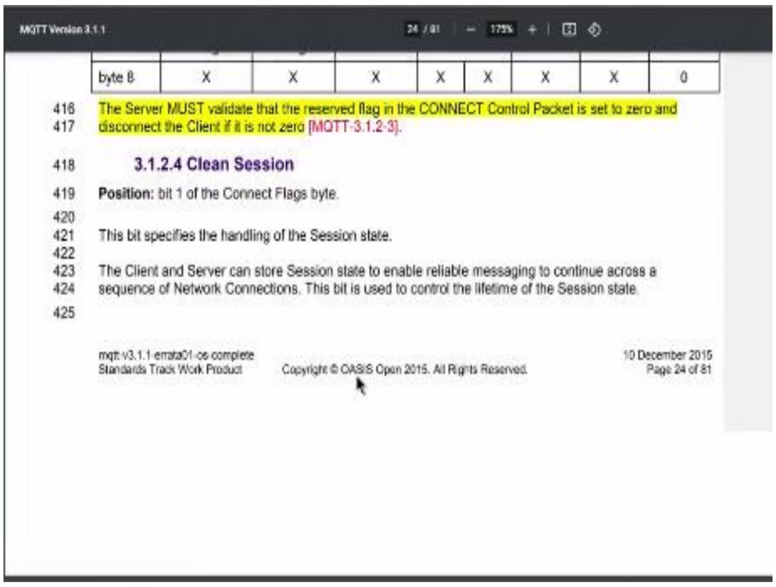
Figure 3.4 - Connect Flag bits

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero [MQTT-3.1.2-3].

You want to know more about Connect flags, it is right here, okay?

(Refer Slide Time: 17:15)



MQTT Version 3.1.1   24 / 81   175%

byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero [MQTT-3.1.2-3].

3.1.2.4 Clean Session

Position: bit 1 of the Connect Flags byte.

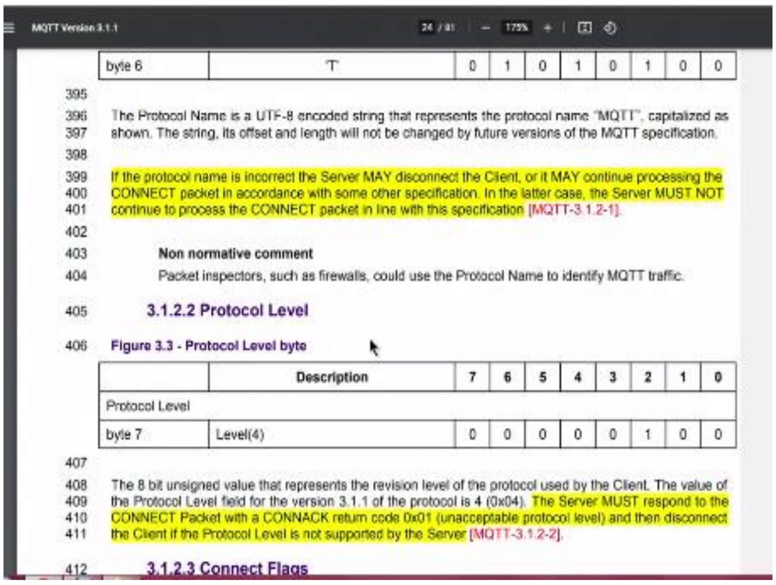
This bit specifies the handling of the Session state.

The Client and Server can store Session state to enable reliable messaging to continue across a sequence of Network Connections. This bit is used to control the lifetime of the Session state.

mqtt-v3.1.1-errata01-os-complete
Standards Track Work Product   Copyright © OASIS Open 2015. All Rights Reserved.   10 December 2015
Page 24 of 81

How to clean session, how to start with a clean session? How to start resuming from an existing session?

(Refer Slide Time: 17:23)



MQTT Version 3.1.1   24 / 81   175%

byte 6	'T'	0	1	0	1	0	1	0	0

The Protocol Name is a UTF-8 encoded string that represents the protocol name "MQTT", capitalized as shown. The string, its offset and length will not be changed by future versions of the MQTT specification.

If the protocol name is incorrect the Server MAY disconnect the Client, or it MAY continue processing the CONNECT packet in accordance with some other specification. In the latter case, the Server MUST NOT continue to process the CONNECT packet in line with this specification [MQTT-3.1.2-1].

Non normative comment
Packet inspectors, such as firewalls, could use the Protocol Name to identify MQTT traffic.

3.1.2.2 Protocol Level

Figure 3.3 - Protocol Level byte

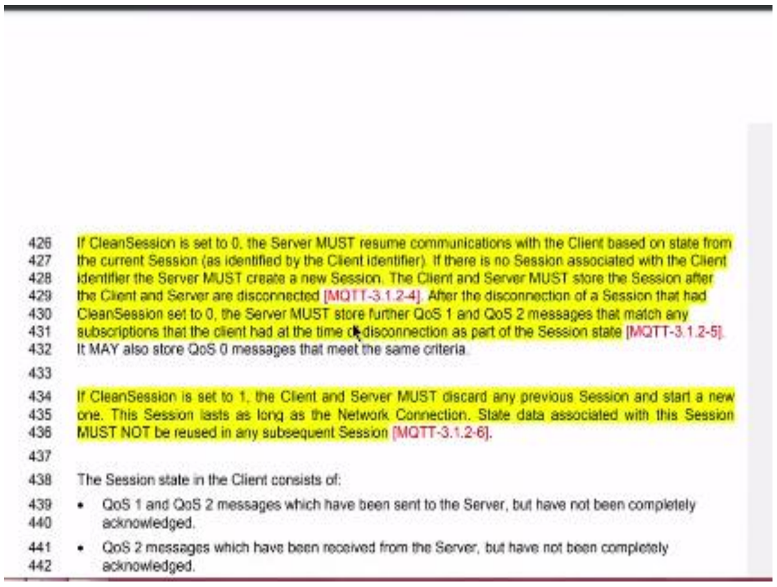
	Description	7	6	5	4	3	2	1	0
Protocol Level									
byte 7	Level(4)	0	0	0	0	0	1	0	0

The 8 bit unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Level field for the version 3.1.1 of the protocol is 4 (0x04). The Server MUST respond to the CONNECT Packet with a CONNACK return code 0x01 (unacceptable protocol level) and then disconnect the Client if the Protocol Level is not supported by the Server [MQTT-3.1.2-2].

3.1.2.3 Connect Flags

You make it 0, during Connect. If you make a flag as 0, then it will resume. If you make the flag 1, it will start a new session. That is written here very clearly. Where is it? Right here, you can see that. There is a nice definition of these two here.

(Refer Slide Time: 17:39)



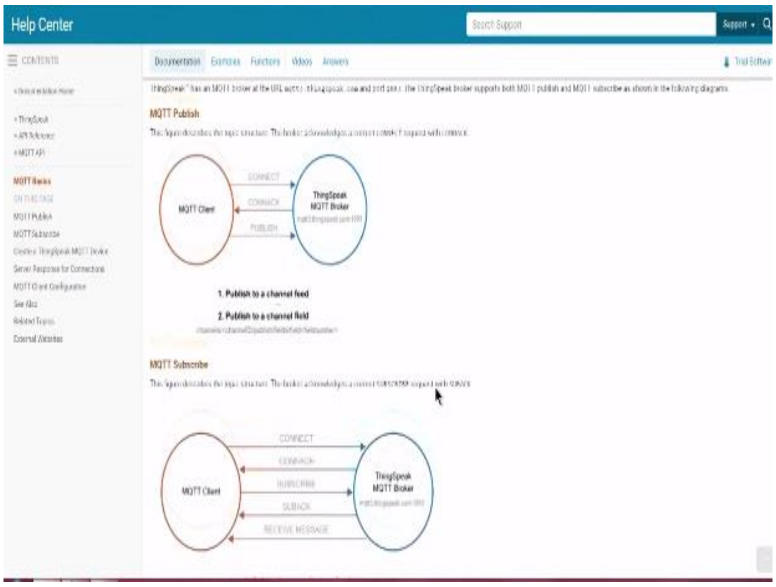
426 If CleanSession is set to 0, the Server MUST resume communications with the Client based on state from
427 the current Session (as identified by the Client identifier). If there is no Session associated with the Client
428 identifier the Server MUST create a new Session. The Client and Server MUST store the Session after
429 the Client and Server are disconnected [MQTT-3.1.2-4]. After the disconnection of a Session that had
430 CleanSession set to 0, the Server MUST store further QoS 1 and QoS 2 messages that match any
431 subscriptions that the client had at the time of disconnection as part of the Session state [MQTT-3.1.2-5].
432 It MAY also store QoS 0 messages that meet the same criteria.
433
434 If CleanSession is set to 1, the Client and Server MUST discard any previous Session and start a new
435 one. This Session lasts as long as the Network Connection. State data associated with this Session
436 MUST NOT be reused in any subsequent Session [MQTT-3.1.2-6].
437
438 The Session state in the Client consists of:
439 • QoS 1 and QoS 2 messages which have been sent to the Server, but have not been completely
440 acknowledged.
441 • QoS 2 messages which have been received from the Server, but have not been completely
442 acknowledged.

Line number 426 talks about 0. Line number 434 talks about 1. Very simple, folks. QoS 1, 2 we have already discussed, it is out there. Okay, we will come to the remaining flags as we go along. But what is important is this CONNECT and not to lose track of what this guy is actually telling us. So you know all about CONNECT. If you send a CONNECT, you will get a CONNACK.

You will get a connection establishment from the broker. Remember, this broker is sitting on the ThingSpeak Cloud. Remember this is your windows blind client, okay. You are putting everything onto the cloud here. There is a certain API support from the ThingSpeak broker which you should know very well. Only based on that you can actually write data to the cloud system on the channel that you have created, okay.

Then after you do that, then you start publishing your outdoor light data, indoor light data so on and so forth, you can start pushing it here.

(Refer Slide Time: 18:47)


Help Center

Documentation Examples Functions Videos Answers

ThingSpeak™ has an MQTT broker at the URL mqtt.thingspeak.com and port 1883. The ThingSpeak broker supports both MQTT publish and MQTT subscribe as shown in the following diagrams.

MQTT Publish

This figure describes the topic structure. The broker acknowledges a correct CONNECT request with CONNACK.

1. Publish to a channel feed

2. Publish to a channel field

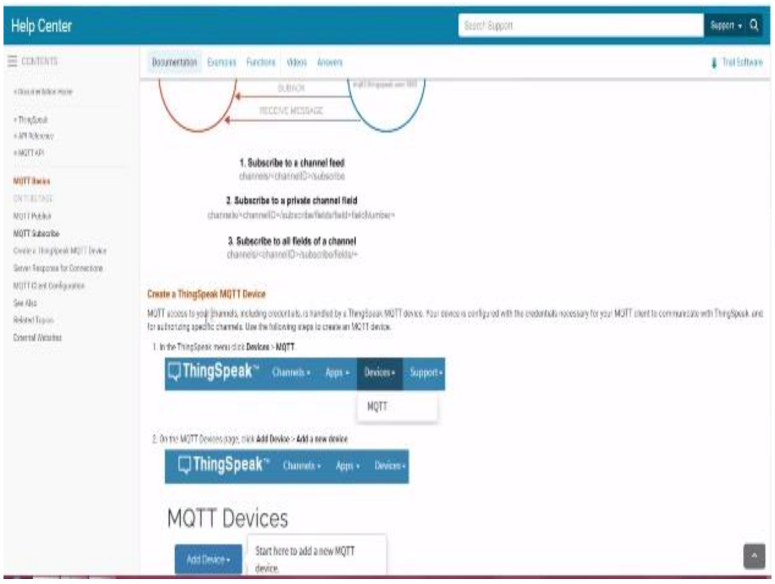
MQTT Subscribe

This figure describes the topic structure. The broker acknowledges a correct SUBSCRIBE request with SUBACK.

And that is exactly what he has spoken about in MQTT publish. Now how does your window blind motor controller get the data is the next question. How it does is very simple. It also connects to the Cloud, CONNECT, CONNACK, then it says I am subscribing to three pieces of information. One piece is related to outdoor light, other piece of information is indoor light.

The other piece of information is presence or absence of humans. So it will say I am going to subscribe like that. Please give me an acknowledgement whether you have received my subscription, which is SUBACK. And then once it receives a SUBACK, it starts getting the information as uploaded by the sensors, which was done in this step. It is as simple as that. And that is described clearly here.

(Refer Slide Time: 19:40)


Help Center

Documentation Examples Functions Videos Answers

1. Subscribe to a channel feed
channels/<channelID>/subscribe

2. Subscribe to a private channel field
channels/<channelID>/subscribe/fields/field<fieldNumber>

3. Subscribe to all fields of a channel
channels/<channelID>/subscribe/fields/+

Create a ThingSpeak MQTT Device

MQTT access to your channels, including credentials, is handled by a ThingSpeak MQTT device. Your device is configured with the credentials necessary for your MQTT client to communicate with ThingSpeak, and for authorizing specific channels. Use the following steps to create an MQTT device.

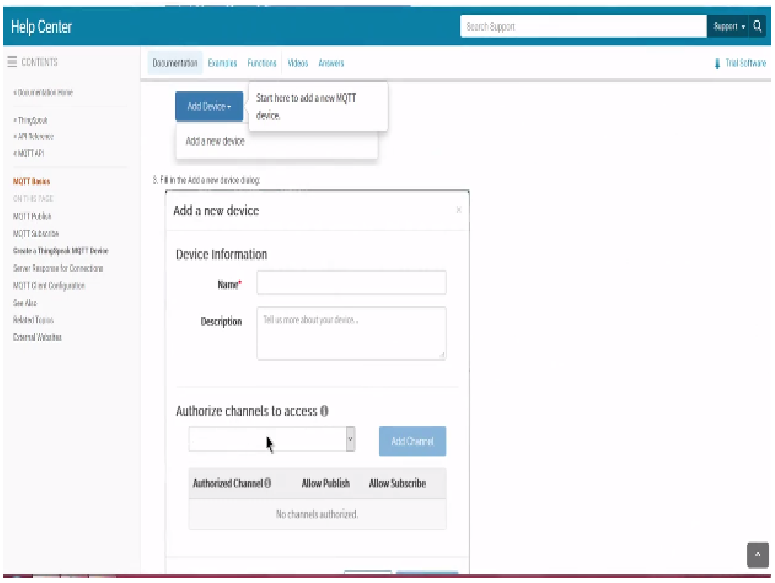
1. In the ThingSpeak menu click Devices > MQTT.

2. On the MQTT Devices page, click Add Device > Add a new device

ThingSpeak™ Channels Apps Devices
MQTT Devices
Add Device
Start here to add a new MQTT device.

Now I mentioned to you that you can create MQTT devices.

(Refer Slide Time: 19:44)



Add Device ▾

Start here to add a new MQTT device.

Add a new device

3. Fill in the Add a new device dialog:

Add a new device

Device Information

Name*

Description: Tell us more about your device...

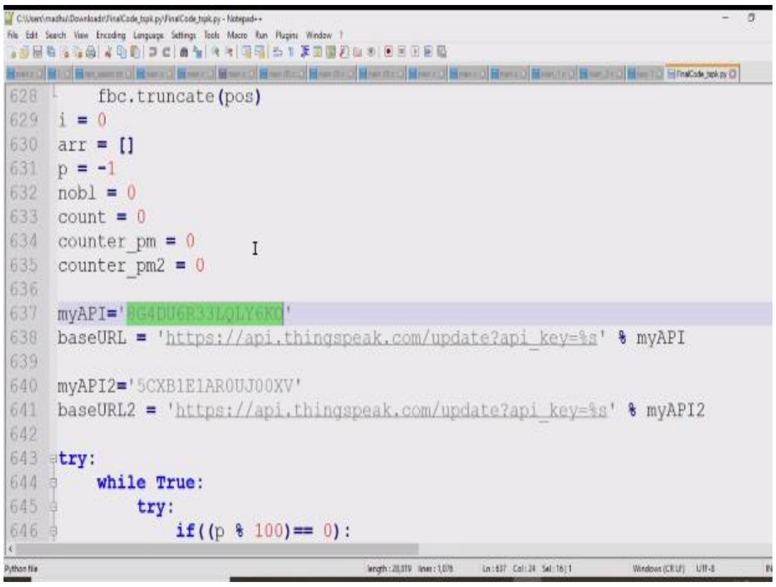
Authorize channels to access

Add Channel

Authorized Channel	Allow Publish	Allow Subscribe
No channels authorized.		

You can see that this is the way by which it is you can create the device and it will this website will generate a code which you can copy and paste it into your Paho client and then start uploading the data. Let us shift to that and then also show you that specific part of where exactly that this code that is generated actually appears in your Paho client. So let us see.

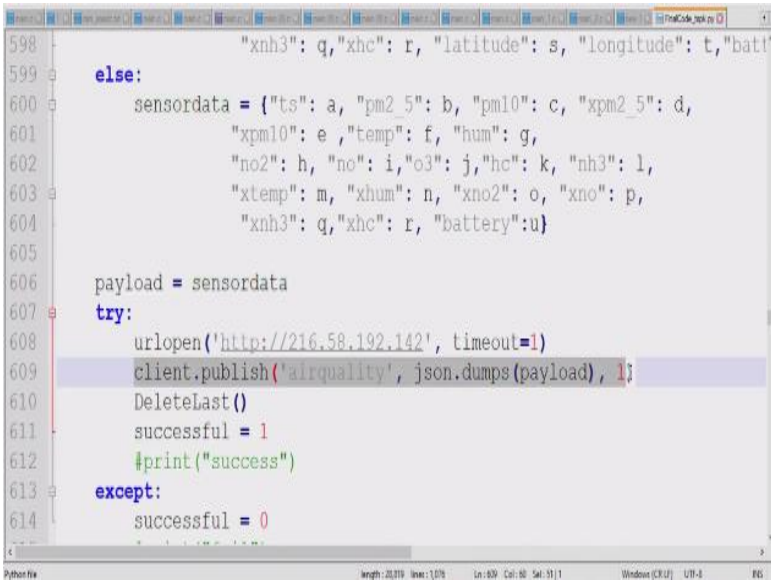
(Refer Slide Time: 20:16)



```
    fbc.truncate(pos)
i = 0
arr = []
p = -1
nobl = 0
count = 0
counter_pm = 0
counter_pm2 = 0

myAPI='8G4DU6R33LQLY6KQ'
baseURL = 'https://api.thingspeak.com/update?api_key=%s' % myAPI

myAPI2='5CXB1E1AR0UJ00XV'
baseURL2 = 'https://api.thingspeak.com/update?api_key=%s' % myAPI2

try:
    while True:
        try:
            if((p % 100)== 0):
```

There you are. This is the code. Let us first see, the code that is generated. You see, the one that is highlighted by Vasanth clearly indicates that this is the code that was generated by during the time when you added the device, okay. Then there is the standard API provided by ThingSpeak. One such example can also be shown, which is related to this particular thing. For example, in what format you should upload is actually mentioned here.

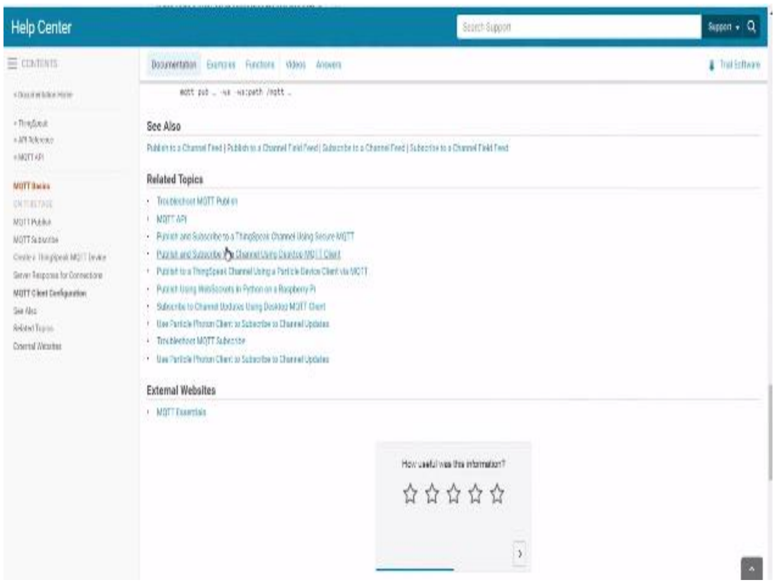
(Refer Slide Time: 20:45)



```
                    "xnh3": q,"xhc": r, "latitude": s, "longitude": t,"batt
    else:
        sensordata = {"ts": a, "pm2_5": b, "pm10": c, "xpm2_5": d,
                   "xpm10": e ,"temp": f, "hum": g,
                   "no2": h, "no": i,"o3": j,"hc": k, "nh3": l,
                   "xtemp": m, "xhum": n, "xno2": o, "xno": p,
                    "xnh3": q,"xhc": r, "battery":u}

    payload = sensordata
    try:
        urlopen('http://216.58.192.142', timeout=1)
        client.publish('airquality', json.dumps(payload), 1)
        DeleteLast()
        successful = 1
        #print("success")
    except:
        successful = 0
```

Client.publish air quality Json.dumps payload. You write like this, you put it in this format, URL open, you open the URL, you do a client publish, that means you are able to upload the data on to the cloud, on the ThingSpeak cloud. So it is as simple as that. You need to understand the API well and then start using the API. In order to create your own quick application, please I encourage you to start using this ThingSpeak as a first step to in order to start using and learning how to use this basic system.

(Refer Slide Time: 21:24)



Help Center

```
mqtt pub … -ws -ws:path /mqtt …
```

See Also

Publish to a Channel Feed | Publish to a Channel Field Feed | Subscribe to a Channel Feed | Subscribe to a Channel Field Feed

Related Topics

- Troubleshoot MQTT Publish
- MQTT API
- Publish and Subscribe to a ThingSpeak Channel Using Secure MQTT
- Publish and Subscribe to a Channel Using Desktop MQTT Client
- Publish to a ThingSpeak Channel Using a Particle Device Client via MQTT
- Publish Using WebSockets in Python on a Raspberry Pi
- Subscribe to Channel Updates Using Desktop MQTT Client
- Use Particle Photon Client to Subscribe to Channel Updates
- Troubleshoot MQTT Subscribe
- Use Particle Photon Client to Subscribe to Channel Updates

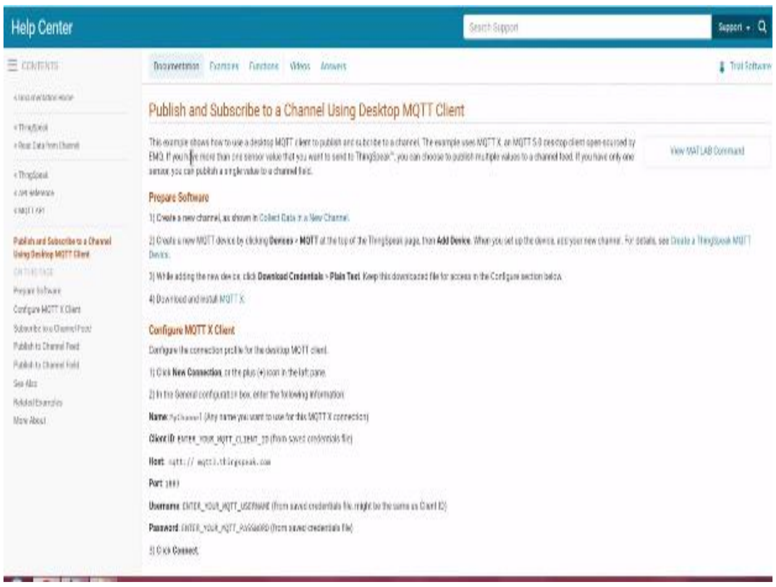
External Websites

- MQTT Essentials

How useful was this information?

Okay, then there are lots of stuff related to publish and subscribe to a ThingSpeak channel using secure MQTT. Publish and channel to a channel using desktop MQTT client. Just click on this, you will get a topic.

(Refer Slide Time: 21:39)



Publish and Subscribe to a Channel Using Desktop MQTT Client

This example shows how to use a desktop MQTT client to publish and subscribe to a channel. The example uses MQTT X, an MQTT 5.0 desktop client open-sourced by EMQ. If you have more than one sensor value that you want to send to ThingSpeak™, you can choose to publish multiple values to a channel feed. If you have only one sensor, you can publish a single value to a channel field.

Prepare Software

1) Create a new channel, as shown in Collect Data in a New Channel.

2) Create a new MQTT device by clicking **Devices** > **MQTT** at the top of the ThingSpeak page, then **Add Device**. When you set up the device, add your new channel. For details, see Create a ThingSpeak MQTT Device.

3) While adding the new device, click **Download Credentials** > **Plain Text**. Keep this downloaded file for access in the Configure section below.

4) Download and install MQTT X.

Configure MQTT X Client

Configure the connection profile for the desktop MQTT client.

1) Click **New Connection**, or the plus (+) icon in the left pane.

2) In the General configuration box, enter the following information:

Name: `MyChannel1` (Any name you want to use for this MQTT X connection)

Client ID: `ENTER_YOUR_MQTT_CLIENT_ID` (from saved credentials file)

Host: `mqtt:// mqtt3.thingspeak.com`

Port: `1883`

Username: `ENTER_YOUR_MQTT_USERNAME` (from saved credentials file, might be the same as Client ID)

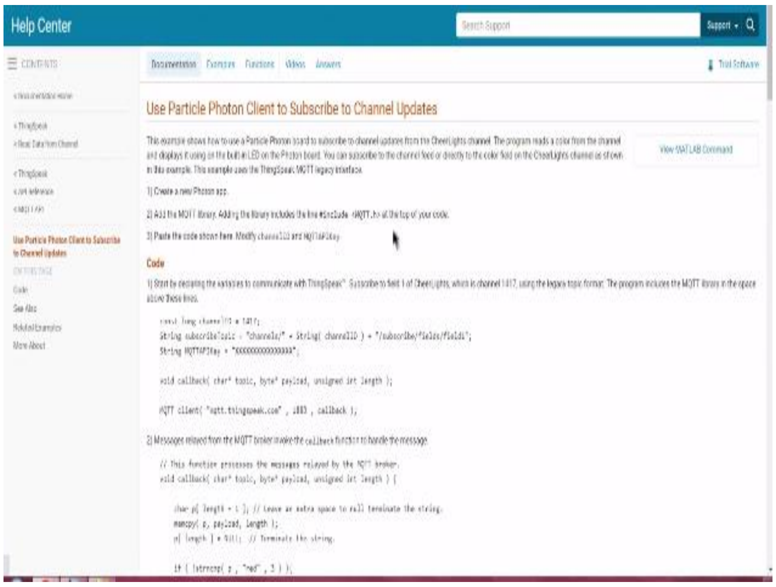
Password: `ENTER_YOUR_MQTT_PASSWORD` (from saved credentials file)

3) Click **Connect**.

It will tell you how to do it, okay. This example shows you a desktop and all that. Try simple things first folks, before you complicate anything else, right? And read this particular aspect of ThingSpeak well so that you will be able to practice MQTT protocol extremely well, okay. This is very important. Now the ThingSpeak also encourages you to look at some standard boards.

And one such standard board appears to be the particle photon board. So you can use this particle photon board.

(Refer Slide Time: 22:16)


Help Center

Documentation Examples Functions Videos Answers

Use Particle Photon Client to Subscribe to Channel Updates

This example shows how to use a Particle Photon board to subscribe to channel updates from the CheerLights channel. The program reads a color from the channel and displays it using on the built in LED on the Photon board. You can subscribe to the channel feed or directly to the color field on the CheerLights channel as shown in this example. This example uses the ThingSpeak MQTT legacy interface.

1) Create a new Photon app.

2) Add the MQTT library. Adding the library includes the line `#include <MQTT.h>` at the top of your code.

3) Paste the code shown here. Modify `channelID` and `MQTTAPIKey`.

Code

1) Start by declaring the variables to communicate with ThingSpeak™. Subscribe to field 1 of CheerLights, which is channel 1417, using the legacy topic format. The program includes the MQTT library in the space above these lines.

```
const long channelID = 1417;
String subscribeTopic = "channels/" + String( channelID ) + "/subscribe/fields/field1";
String MQTTAPIKey = "XXXXXXXXXXXXXXXX";

void callback( char* topic, byte* payload, unsigned int length );

MQTT client( "mqtt.thingspeak.com" , 1883 , callback );
```

2) Messages relayed from the MQTT broker invoke the `callback` function to handle the message.

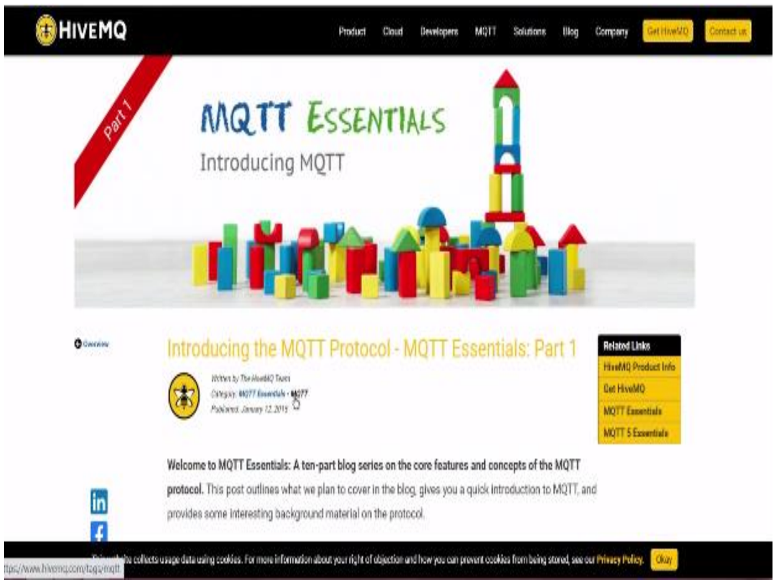
```
// This function processes the messages relayed by the MQTT broker.
void callback( char* topic, byte* payload, unsigned int length ) {

    char p[ length + 1 ]; // Leave an extra space to null terminate the string.
    memcpy( p, payload, length );
    p[ length ] = NULL;  // Terminate the string.

    if ( !strncmp( p , "red" , 3 ) ){
```

He gives you some standard code, which you can take and integrate into your system. So let us read this. This example shows how a particle photon board, how to use a particle photon board to subscribe to a channel updates from some topic called cheer lights, okay. I will simply use the word replace channel with the word topic.

The program reads a color from the channel, from the topic and displays it using built in LED on the photon board. You can subscribe to the channel feed or directly to the color field on the cheer lights channel as shown in the example. So you download this code run it, your photon board has LEDs and you can play with the LEDs on the photon board. Essentially it is, this is the code.

You can put up on the MQTT broker site and you will see your LEDs dancing as and when the whole system comes up. So I would strongly encourage you to look up this website and start using them and understand MQTT as much as required and also its nuances okay, very important.

(Refer Slide Time: 23:30)



MQTT ESSENTIALS

Introducing MQTT

Introducing the MQTT Protocol - MQTT Essentials: Part 1

Written by The HiveMQ Team

Category: MQTT Essentials · MQTT

Published: January 12, 2015

Welcome to MQTT Essentials: A ten-part blog series on the core features and concepts of the MQTT protocol. This post outlines what we plan to cover in the blog, gives you a quick introduction to MQTT, and provides some interesting background material on the protocol.

This website collects usage data using cookies. For more information about your right of objection and how you can prevent cookies from being stored, see our Privacy Policy.

So if you go to MQTT essentials, this website, it will tell you a lot about introducing the MQTT protocol. Okay. What is it you need to know about MQTT, this is written very well. And there is so much to learn from this whole system. And of course, HIVEMQ has published many articles. And they actually tell you why you should shift to from 3.1.1 to MQTT version 5, okay.

And they will tell you why that is an important thing. A lot of videos are out there. Please do read up those videos, you will understand the protocol extremely well. I have to teach you the protocol in a little more detail. This is as far as building applications are concerned, quick building of the applications. Some little bit you should know about the theory.

You should know about the protocol, its limitations, what are the things it does from low power, correct? You can have low, you need low power, is it not? Because you need 10 year lifetime? How can you use this protocol for 10 year lifetime? What happens if the battery is draining fast? How can nodes come to know about it? Those issues will come up as we start putting IoT nodes into real practice. Fine?

Protocol support is there. But how do you exploit that support if you do not use the hardware properly. That is also an important thing and we will get into that detail right away now.

(Refer Slide Time: 25:08)

OASIS

MQTT Version 3.1.1 Plus Errata 01

OASIS Standard Incorporating Approved Errata 01

10 December 2015

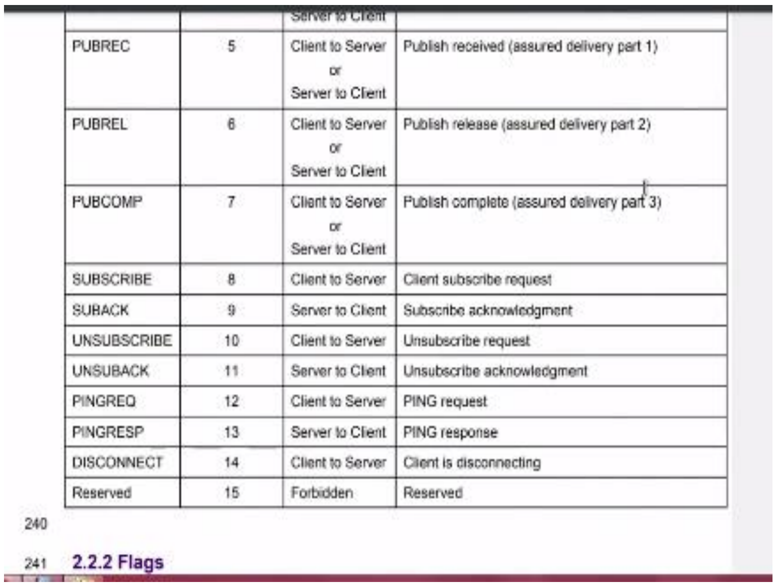
Specification URIs

This version:

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.doc (Authoritative)

So let me open the best possible resource for MQTT I mentioned to you is the standard. These are all the possible MQTT control packet types, CONNECT, CONNACK, PUBLISH, we already know.

(Refer Slide Time: 25:19)



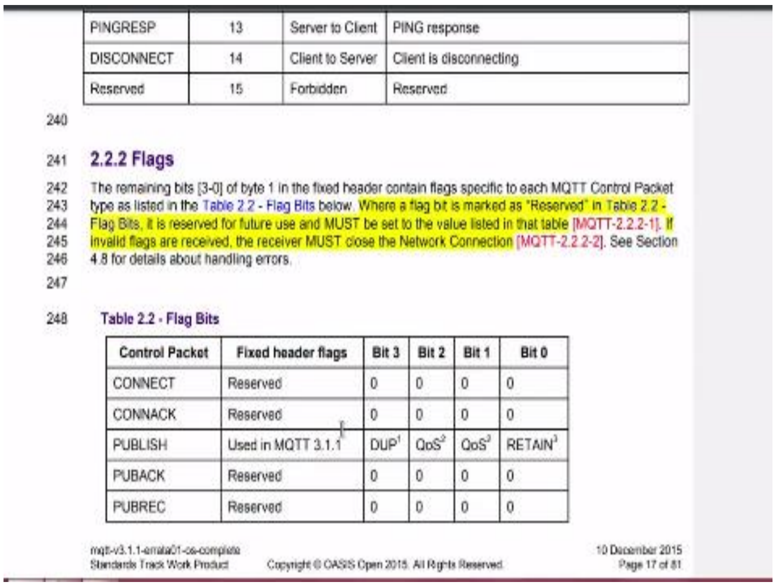
		Server to Client	
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

2.2.2 Flags

PUBACK is there. PUBREC is there. PUBREL is there. PUBCOMP is there and so on. There are small differences between each one of these and their requirements. Based on the QoS level, you would be using all these additional control packets, okay. So if you use the highest level, which is QoS 0, 1, 2 right, the third is the highest level QoS0, QoS1 and QoS2, which is exactly once, exactly once.

Then PUBCOMP also will come into picture okay, because this is all about assured delivery as you can see here. Then there is SUBSCRIBE, there is a SUBACK, UNSUBSCRIBE. I do not want this topic anymore. So UNSUBSCRIBE. You want to check if your client is alive or not use PING. If you want to disconnect the client from the server use DISCONNECT and Reserved. These are the main set of commands.

(Refer Slide Time: 26:26)



PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

2.2.2 Flags

The remaining bits [3-0] of byte 1 in the fixed header contain flags specific to each MQTT Control Packet type as listed in the Table 2.2 - Flag Bits below. Where a flag bit is marked as "Reserved" in Table 2.2 - Flag Bits, it is reserved for future use and MUST be set to the value listed in that table [MQTT-2.2.2-1]. If invalid flags are received, the receiver MUST close the Network Connection [MQTT-2.2.2-2]. See Section 4.8 for details about handling errors.

Table 2.2 - Flag Bits

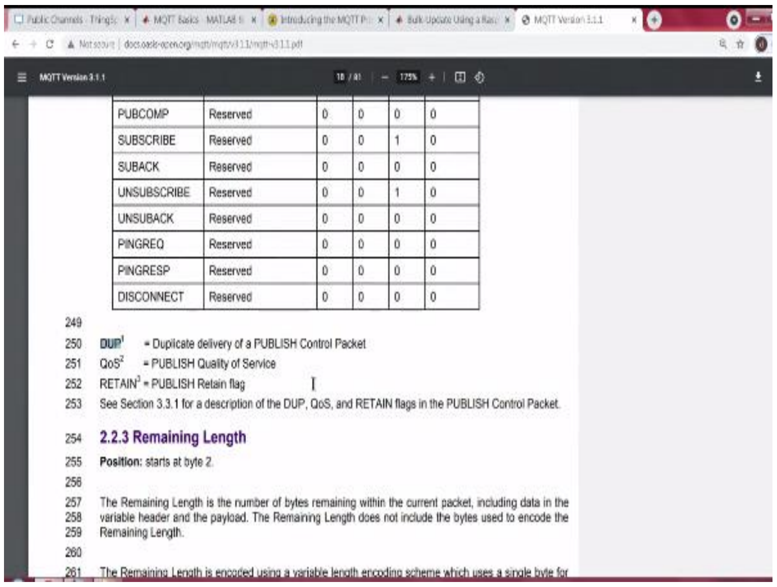
Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP[1]	QoS[2]	QoS[2]	RETAIN[3]
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0

mqtt-v3.1.1-errata01-os-complete
Standards Track Work Product   Copyright © OASIS Open 2015. All Rights Reserved.   10 December 2015
Page 17 of 81

Then there is I mentioned to you about flags, know them in great detail. They are all part of the header, which is an important thing, it is part of the fixed header. And understand how these flags are used. Now let us get into little more detail. It is a little dry, but you should know, okay? Look at this four bits 3, 2, 1, 0 okay? Here, if you take PUBLISH message, how you should use these flags is mentioned.

In CONNECT, you do not need this bit, they are all set to 0, but only when you are publishing. Question is why? Is it not? You are telling the broker to do something, to accept this message in a particular way. That is why you need it in the PUBLISH. So what is DUP with a superscript of 1? You can look up that in a little more detail to understand it is written here.

(Refer Slide Time: 27:24)



PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

DUP[1] = Duplicate delivery of a PUBLISH Control Packet
QoS[2] = PUBLISH Quality of Service
RETAIN[3] = PUBLISH Retain flag
See Section 3.3.1 for a description of the DUP, QoS, and RETAIN flags in the PUBLISH Control Packet.

2.2.3 Remaining Length

Position: starts at byte 2.

The Remaining Length is the number of bytes remaining within the current packet, including data in the variable header and the payload. The Remaining Length does not include the bytes used to encode the Remaining Length.

The Remaining Length is encoded using a variable length encoding scheme which uses a single byte for

It says duplicate delivery of PUBLISH control packet. QoS to PUBLISH quality of service. Retain 3, PUBLISH retain flag, okay. You are setting these three flags, these are one bit flags, you are setting them. See essentially you are saying that I will support or not support data packet duplicates, okay. You are setting a QoS field in this systems.

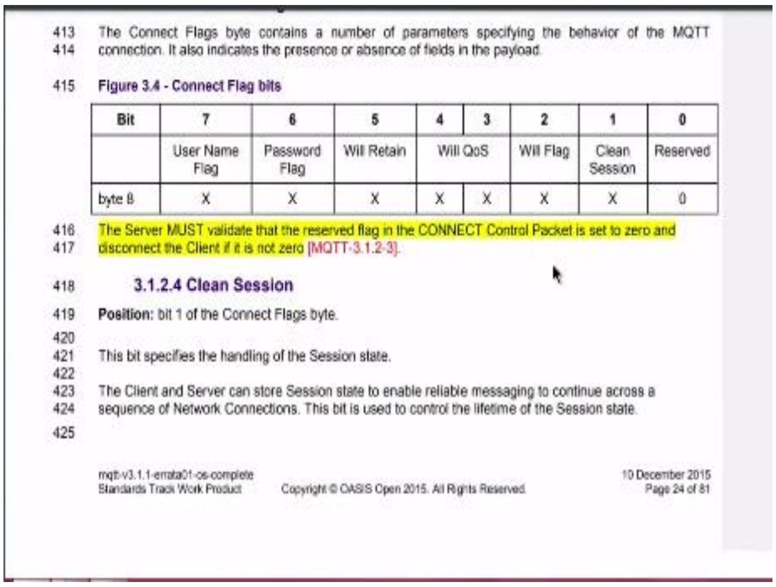
One bit is not sufficient for QoS because you have QoS 0, 1, 2 which means you need minimum two bits. And you are actually enabling DUP, means you are saying that I have already sent once I am sending again. That means I have, you put that flag 1, then the receiver will know oh I see, the packet was sent earlier, but it is being sent again. And that is why it comes in the PUBLISH message. RETAIN.

RETAIN is interesting. What RETAIN says is if I publish to my broker, either the ThingSpeak broker or my home lab broker, without using internet, please keep the message with you, RETAIN the message with you. Then, if you RETAIN it with you, clients may come with subscriptions later in that time. That time you can PUBLISH to them when they join. It is not necessary that subscribers and publishers should be on at the same time.

That is the beauty of MQTT. The sender and the receiver are disconnected, okay. I can PUBLISH today night and go to sleep and do what I like. You can wake up in the morning tomorrow, your node can wake up in the morning tomorrow and get the data provided it was retained on the broker. That is the key point. So such things have to be thought of. So it is a very important requirement for building MQTT apps.

Now to understand that you should know about it well. So we should spend at least a little more time on these important flags. So let me go to another important setting and that setting is related to Wll flag. Now where does the Wll flag come? Wll flag should also come in somewhere here.

(Refer Slide Time: 30:09)



The Connect Flags byte contains a number of parameters specifying the behavior of the MQTT connection. It also indicates the presence or absence of fields in the payload.

Figure 3.4 - Connect Flag bits

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero [MQTT-3.1.2-3].

3.1.2.4 Clean Session

Position: bit 1 of the Connect Flags byte.

This bit specifies the handling of the Session state.

The Client and Server can store Session state to enable reliable messaging to continue across a sequence of Network Connections. This bit is used to control the lifetime of the Session state.

mqtt-v3.1.1-errata01-os-complete
Standards Track Work Product Copyright © OASIS Open 2015. All Rights Reserved. 10 December 2015
Page 24 of 81

You see, it is here. It is in the CONNECT. When you are doing a connection, not in the Publish. Publish we already discussed. It is about the data. What are the flags in the Publish? We know already. But when you are connecting to the server, you are using the very first protocol, which is a command is called the CONNECT. There will is there. Now we will see what is this will? Will Retain is there.

Will Flag is there. And how do you do this Will Retain and what is will at all in the first place is the question, right? See folks, IoT nodes get the philosophy correct. In IoT, you are driving it with energy harvesting. You are driving the sensor node with battery driven systems.

You do not know when the battery goes down, you cannot be planning the protocol supports that part which says, even if you go down, can you not send that last message saying that I am going away, this is my last sensor data. And request the broker to say, I am giving you this last message. Serve it to whoever asks, comes for this topic, the topic which I published.

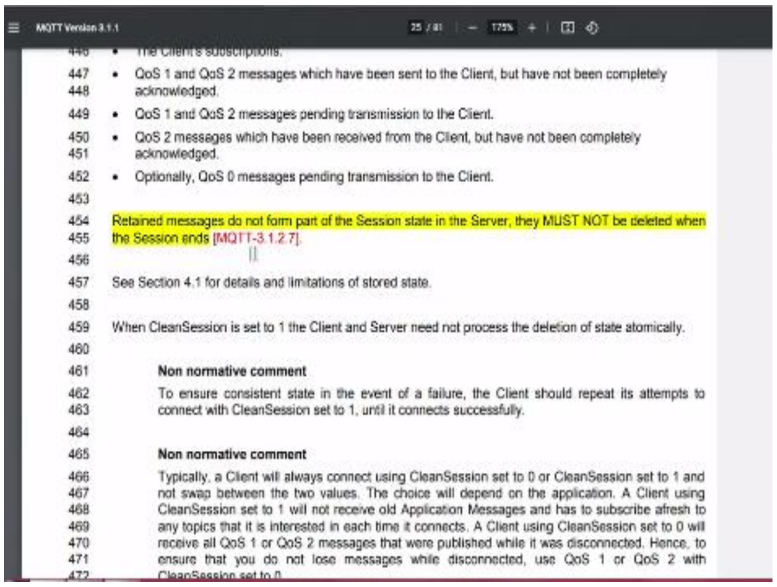
There must be subscribers, give them that message. Beauty, right? Think about the blind example. The outdoor sensor ran out of battery, ran completely out of battery. It has got the outdoor data last sample it could get. What it did, it just said it did a CONNECT, ran this flag as 1 and said this is the Wll and please retain this. Now the broker knows, oh, this guy has put a Wll flag, which means this guy may not come back and give a periodic update.

He is not going to give a periodic update, it put in Wll already. Now the applications which are built around the outdoor light data are asking for the data to the broker. So give me that data, give me the data. Broker says, look I am giving up because my sensor node has given up. He has told me this is the last piece of data.

And this is the last piece he has given me. I am giving you this information, date and time and outdoors light data and pushed it to the application that has requested. So it does that. So applications which are build in says, Oh, I see. The last update was three hours back, or two hours back or whatever. And now I have to plan my strategy because I do not have further updates. I do not have further updates.

So you see now it is all integrated into the protocol. This is the beauty of the protocol. So you should use that protocol feature. And that feature comes from Wll. And that is what I am trying to show you here.

(Refer Slide Time: 33:31)


MQTT Version 3.1.1 25 / 81 175%

- The Client's subscriptions.
- QoS 1 and QoS 2 messages which have been sent to the Client, but have not been completely acknowledged.
- QoS 1 and QoS 2 messages pending transmission to the Client.
- QoS 2 messages which have been received from the Client, but have not been completely acknowledged.
- Optionally, QoS 0 messages pending transmission to the Client.

Retained messages do not form part of the Session state in the Server, they MUST NOT be deleted when the Session ends [MQTT-3.1.2.7].

See Section 4.1 for details and limitations of stored state.

When CleanSession is set to 1 the Client and Server need not process the deletion of state atomically.

Non normative comment

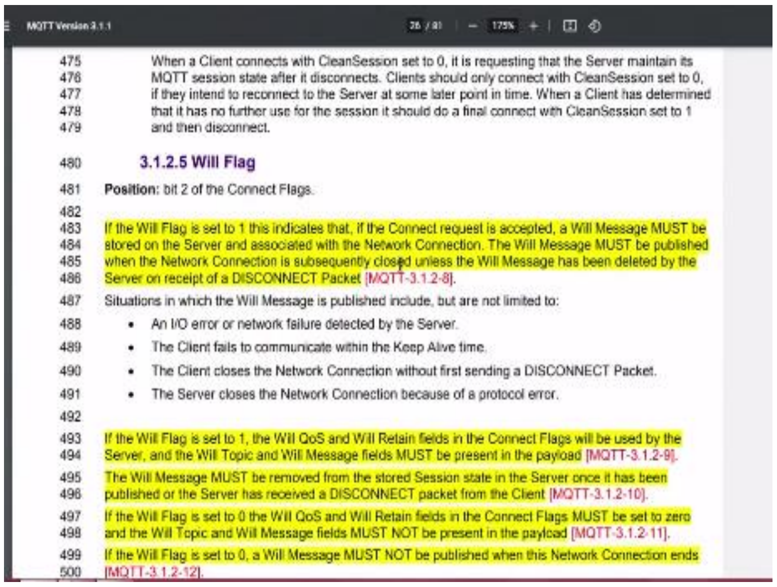
To ensure consistent state in the event of a failure, the Client should repeat its attempts to connect with CleanSession set to 1, until it connects successfully.

Non normative comment

Typically, a Client will always connect using CleanSession set to 0 or CleanSession set to 1 and not swap between the two values. The choice will depend on the application. A Client using CleanSession set to 1 will not receive old Application Messages and has to subscribe afresh to any topics that it is interested in each time it connects. A Client using CleanSession set to 0 will receive all QoS 1 or QoS 2 messages that were published while it was disconnected. Hence, to ensure that you do not lose messages while disconnected, use QoS 1 or QoS 2 with CleanSession set to 0.

Retained messages do not form part of the session state in the server and they must not be deleted when the session ends. See the beauty. Retained messages do not form part of the session state in the server. They must not be deleted when the session and do not keep that message, okay. So what is Wll?

(Refer Slide Time: 33:53)


MQTT Version 3.1.1 26 / 81 175%

When a Client connects with CleanSession set to 0, it is requesting that the Server maintain its MQTT session state after it disconnects. Clients should only connect with CleanSession set to 0, if they intend to reconnect to the Server at some later point in time. When a Client has determined that it has no further use for the session it should do a final connect with CleanSession set to 1 and then disconnect.

3.1.2.5 Will Flag

Position: bit 2 of the Connect Flags.

If the Will Flag is set to 1 this indicates that, if the Connect request is accepted, a Will Message MUST be stored on the Server and associated with the Network Connection. The Will Message MUST be published when the Network Connection is subsequently closed unless the Will Message has been deleted by the Server on receipt of a DISCONNECT Packet [MQTT-3.1.2-8].

Situations in which the Will Message is published include, but are not limited to:

- An I/O error or network failure detected by the Server.
- The Client fails to communicate within the Keep Alive time.
- The Client closes the Network Connection without first sending a DISCONNECT Packet.
- The Server closes the Network Connection because of a protocol error.

If the Will Flag is set to 1, the Will QoS and Will Retain fields in the Connect Flags will be used by the Server, and the Will Topic and Will Message fields MUST be present in the payload [MQTT-3.1.2-9].

The Will Message MUST be removed from the stored Session state in the Server once it has been published or the Server has received a DISCONNECT packet from the Client [MQTT-3.1.2-10].

If the Will Flag is set to 0 the Will QoS and Will Retain fields in the Connect Flags MUST be set to zero and the Will Topic and Will Message fields MUST NOT be present in the payload [MQTT-3.1.2-11].

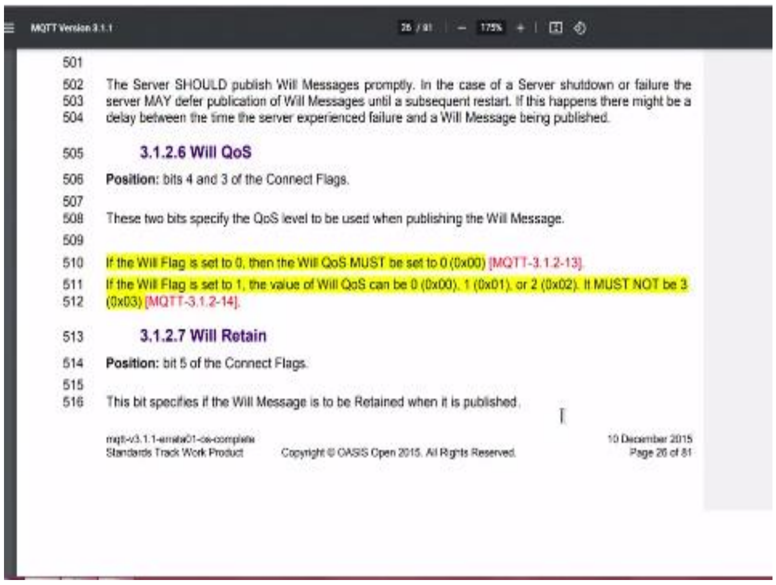
If the Will Flag is set to 0, a Will Message MUST NOT be published when this Network Connection ends [MQTT-3.1.2-12].

Wll says if the Will Flag is set to 1, this indicates that if the Connect request is accepted a will message must be stored on the server and associated with the network connection. The Will message must be published when the network connection is subsequently closed unless the Will message has been deleted by the server on the receipt of a disconnect packet.

So clearly he has given. Unless the guy who published made a disconnect you should not even delete it. You must keep serving. The server has to keep serving it to whoever connects to it and asks for the data. That is the point, okay. Condition under which if you have that flag 0, what should follow, what should not follow. If Will Flag is set to 1 what should follow, and what should not follow is mentioned clearly.

In other words Will Flag equal to 1 something should follow. If Will Flag set to 0, something should not follow. That is mentioned clearly. How will you learn all this? You learn all this through the protocol only. You learn all this through this superly good standard document. So please spend time reading this document.

(Refer Slide Time: 35:21)



The Server SHOULD publish Will Messages promptly. In the case of a Server shutdown or failure the server MAY defer publication of Will Messages until a subsequent restart. If this happens there might be a delay between the time the server experienced failure and a Will Message being published.

3.1.2.6 Will QoS

Position: bits 4 and 3 of the Connect Flags.

These two bits specify the QoS level to be used when publishing the Will Message.

If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00) [MQTT-3.1.2-13].
If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). It MUST NOT be 3 (0x03) [MQTT-3.1.2-14].

3.1.2.7 Will Retain

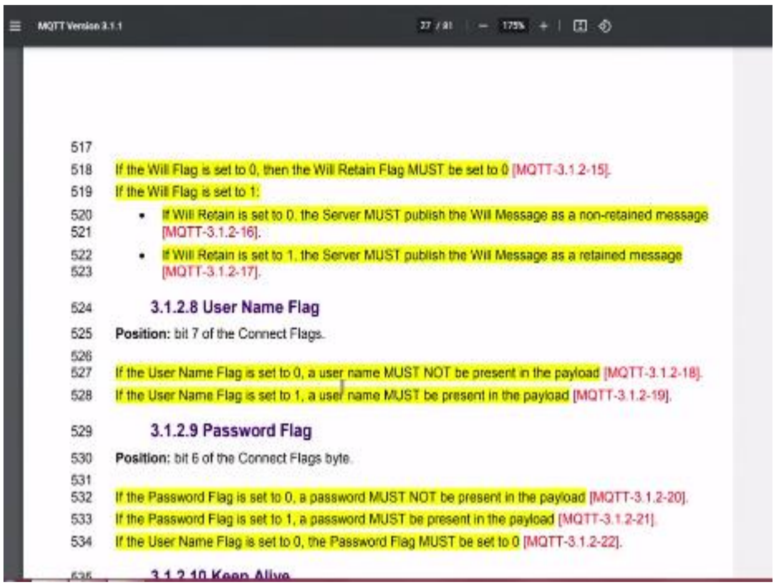
Position: bit 5 of the Connect Flags.

This bit specifies if the Will Message is to be Retained when it is published.

mqtt-v3.1.1-errata01-os-complete
Standards Track Work Product Copyright © OASIS Open 2015. All Rights Reserved.
10 December 2015
Page 26 of 81

What is the QoS level for the Will? That is also the same 3, 0, 1 and 2 right. So all this you can specify. Will Retain I mentioned to you already about the Will Retain.

(Refer Slide Time: 35:34)



MQTT Version 3.1.1 27 / 81 175%

If the Will Flag is set to 0, then the Will Retain Flag MUST be set to 0 [MQTT-3.1.2-15].
If the Will Flag is set to 1:

- If Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message [MQTT-3.1.2-16].
- If Will Retain is set to 1, the Server MUST publish the Will Message as a retained message [MQTT-3.1.2-17].

3.1.2.8 User Name Flag

Position: bit 7 of the Connect Flags.

If the User Name Flag is set to 0, a user name MUST NOT be present in the payload [MQTT-3.1.2-18].
If the User Name Flag is set to 1, a user name MUST be present in the payload [MQTT-3.1.2-19].

3.1.2.9 Password Flag

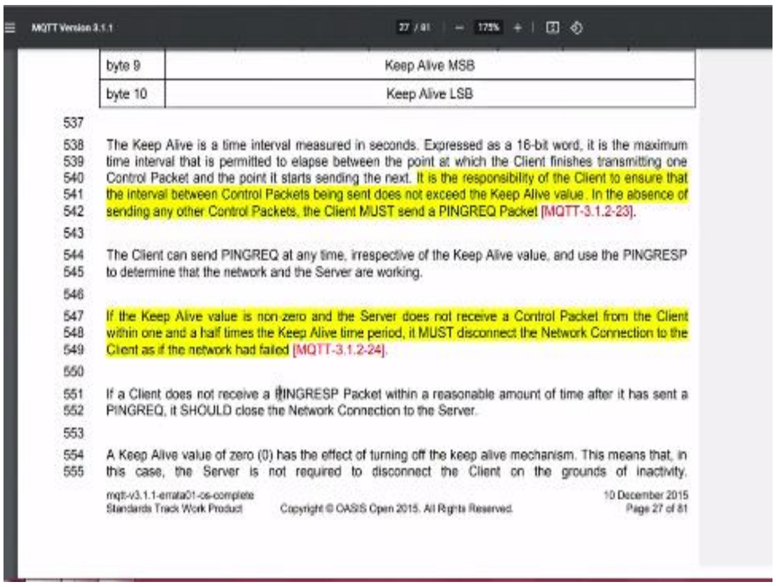
Position: bit 6 of the Connect Flags byte.

If the Password Flag is set to 0, a password MUST NOT be present in the payload [MQTT-3.1.2-20].
If the Password Flag is set to 1, a password MUST be present in the payload [MQTT-3.1.2-21].
If the User Name Flag is set to 0, the Password Flag MUST be set to 0 [MQTT-3.1.2-22].

3.1.2.10 Keep Alive

If the Will Flag is set to 0 Will Retain must be set to 0. You cannot have something following it and so on. So please read this in great detail, you will understand the whole thing.

(Refer Slide Time: 35:47)



MQTT Version 3.1.1 27 / 81 175%

byte 9	Keep Alive MSB
byte 10	Keep Alive LSB

The Keep Alive is a time interval measured in seconds. Expressed as a 16-bit word, it is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure that the interval between Control Packets being sent does not exceed the Keep Alive value. In the absence of sending any other Control Packets, the Client MUST send a PINGREQ Packet [MQTT-3.1.2-23].

The Client can send PINGREQ at any time, irrespective of the Keep Alive value, and use the PINGRESP to determine that the network and the Server are working.

If the Keep Alive value is non zero and the Server does not receive a Control Packet from the Client within one and a half times the Keep Alive time period, it MUST disconnect the Network Connection to the Client as if the network had failed [MQTT-3.1.2-24].

If a Client does not receive a PINGRESP Packet within a reasonable amount of time after it has sent a PINGREQ, it SHOULD close the Network Connection to the Server.

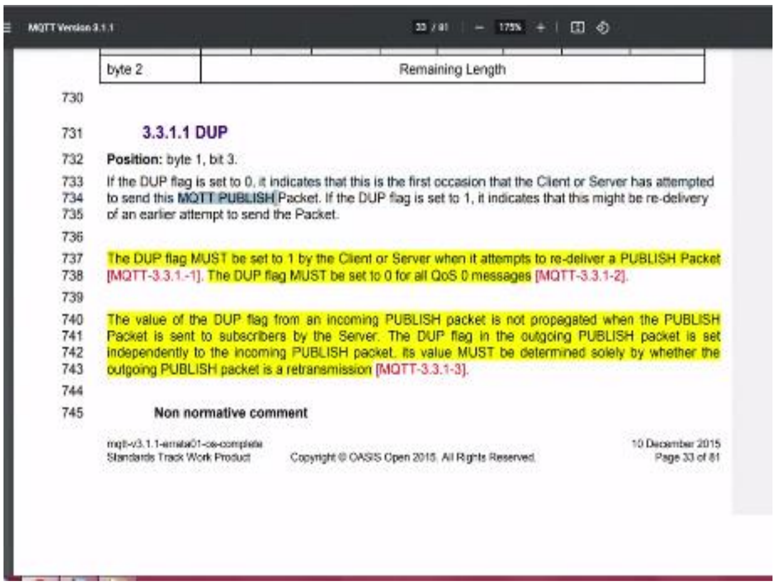
A Keep Alive value of zero (0) has the effect of turning off the keep alive mechanism. This means that, in this case, the Server is not required to disconnect the Client on the grounds of inactivity.

mqtt-v3.1.1-errata01-os-complete
Standards Track Work Product Copyright © OASIS Open 2015. All Rights Reserved. 10 December 2015
Page 27 of 81

Supposing broker wants to check if the CLIENT is free or not free, alive or not alive, it can use this PINGREQ and can also get a PINGRESP back, right? It can get this, so it is also there. So there you are folks, so many nice things about this protocol, which you can use, right? Please do spend the time reading this protocol in detail which is so beautifully written. Every command is mentioned so clearly.

What should CONNACK contain okay. And what should be maintained, what should be present in session, all that is mentioned here so beautifully. So you can see this is PUBLISH. I mentioned to you about PUBLISH having DUP, okay.

(Refer Slide Time: 36:33)



MQTT Version 3.1.1 33 / 81 175%

byte 2	Remaining Length

3.3.1.1 DUP

Position: byte 1, bit 3.

If the DUP flag is set to 0, it indicates that this is the first occasion that the Client or Server has attempted to send this MQTT PUBLISH Packet. If the DUP flag is set to 1, it indicates that this might be re-delivery of an earlier attempt to send the Packet.

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet [MQTT-3.3.1.-1]. The DUP flag MUST be set to 0 for all QoS 0 messages [MQTT-3.3.1-2].

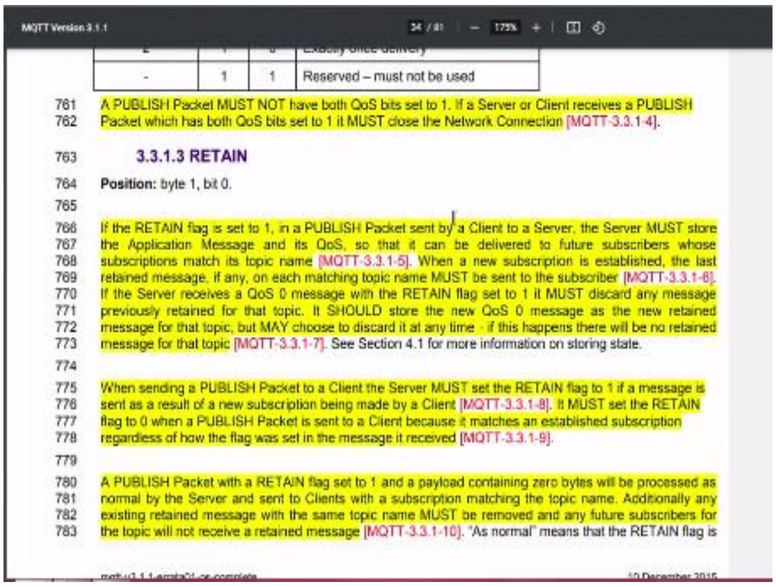
The value of the DUP flag from an incoming PUBLISH packet is not propagated when the PUBLISH Packet is sent to subscribers by the Server. The DUP flag in the outgoing PUBLISH packet is set independently to the incoming PUBLISH packet. Its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission [MQTT-3.3.1-3].

Non normative comment

mqtt-v3.1.1-errata01-os-complete 10 December 2015
Standards Track Work Product Copyright © OASIS Open 2015. All Rights Reserved. Page 33 of 81

DUP is set to 0, it indicates that this is the first occasion the client and the server has attempted to send the PUBLISH packet. If you have set DUP to 1, what does it mean? That means it was already sent once. So you have put the flag to 1. Simple as that, okay.

(Refer Slide Time: 36:52)



MQTT Version 3.1.1 34 / 81 175%

-	1	1	Reserved – must not be used

A PUBLISH Packet MUST NOT have both QoS bits set to 1. If a Server or Client receives a PUBLISH Packet which has both QoS bits set to 1 it MUST close the Network Connection [MQTT-3.3.1-4].

3.3.1.3 RETAIN

Position: byte 1, bit 0.

If the RETAIN flag is set to 1, in a PUBLISH Packet sent by a Client to a Server, the Server MUST store the Application Message and its QoS, so that it can be delivered to future subscribers whose subscriptions match its topic name [MQTT-3.3.1-5]. When a new subscription is established, the last retained message, if any, on each matching topic name MUST be sent to the subscriber [MQTT-3.3.1-6]. If the Server receives a QoS 0 message with the RETAIN flag set to 1 it MUST discard any message previously retained for that topic. It SHOULD store the new QoS 0 message as the new retained message for that topic, but MAY choose to discard it at any time - if this happens there will be no retained message for that topic [MQTT-3.3.1-7]. See Section 4.1 for more information on storing state.

When sending a PUBLISH Packet to a Client the Server MUST set the RETAIN flag to 1 if a message is sent as a result of a new subscription being made by a Client [MQTT-3.3.1-8]. It MUST set the RETAIN flag to 0 when a PUBLISH Packet is sent to a Client because it matches an established subscription regardless of how the flag was set in the message it received [MQTT-3.3.1-9].

A PUBLISH Packet with a RETAIN flag set to 1 and a payload containing zero bytes will be processed as normal by the Server and sent to Clients with a subscription matching the topic name. Additionally any existing retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message [MQTT-3.3.1-10]. "As normal" means that the RETAIN flag is

RETAIN. RETAIN is discussed in so great detail. So you can see here. The RETAIN flag set to 1 in a PUBLISH packet, that means this is about data, right? Remember the Will is in the CONNECT. Whereas the RETAIN is there in the PUBLISH. So when you say Will and that retain, there you are saying that Will message should be retained. That is what you are saying.

Because you have a Will sender and you have also a Will message, which will be served out. Whereas here this is about data. If the RETAIN flag is set to 1, the PUBLISH packet sent by the client to the server must store the application message and its quality of service and it can be delivered to future subscribers whose subscriptions match this topic.

When a new subscription is established, the last retained message if any on each matching topic must be sent to the subscriber and so on. So it is all mentioned quite clearly, right? So folks, that is all I wanted to tell you about the protocol. I am just flipping through this protocol in detail because I think you should spent time understanding, reading and interpreting this in great detail.

So please spend time and get to practice a little bit with the ThingSpeak as well. Thank you very much.