Mphasis
Unleash the Next

Emerging Open and Standard Protocol Stack for IoT

By Aniruddha Chakrabarti
AVP Digital Practice

Internet of Things (IoT) has come a long way since British entrepreneur Kevin Ashton first coined the term in 1999. Though the advent of Industrial IoT goes a long way into the past (remember RFID?), it did not have the form and momentum currently present in the IoT space.

IoT is one of the key components of digital and digital transformation along with Social, Mobile, Analytics, and Cloud (SMAC). Some product vendors refer to IoT as Internet of Everything or Industrial IoT. Digital uses IoT and SMAC stack as building blocks to provide a superior customer experience for users. Today, IoT together with Big Data, Analytics and Cloud can help enable numerous possibilities that were unheard of earlier and technologically not possible otherwise. IoT takes the absolute center stage in the roadmap of product vendors, software companies, system integrators, and IT service companies.

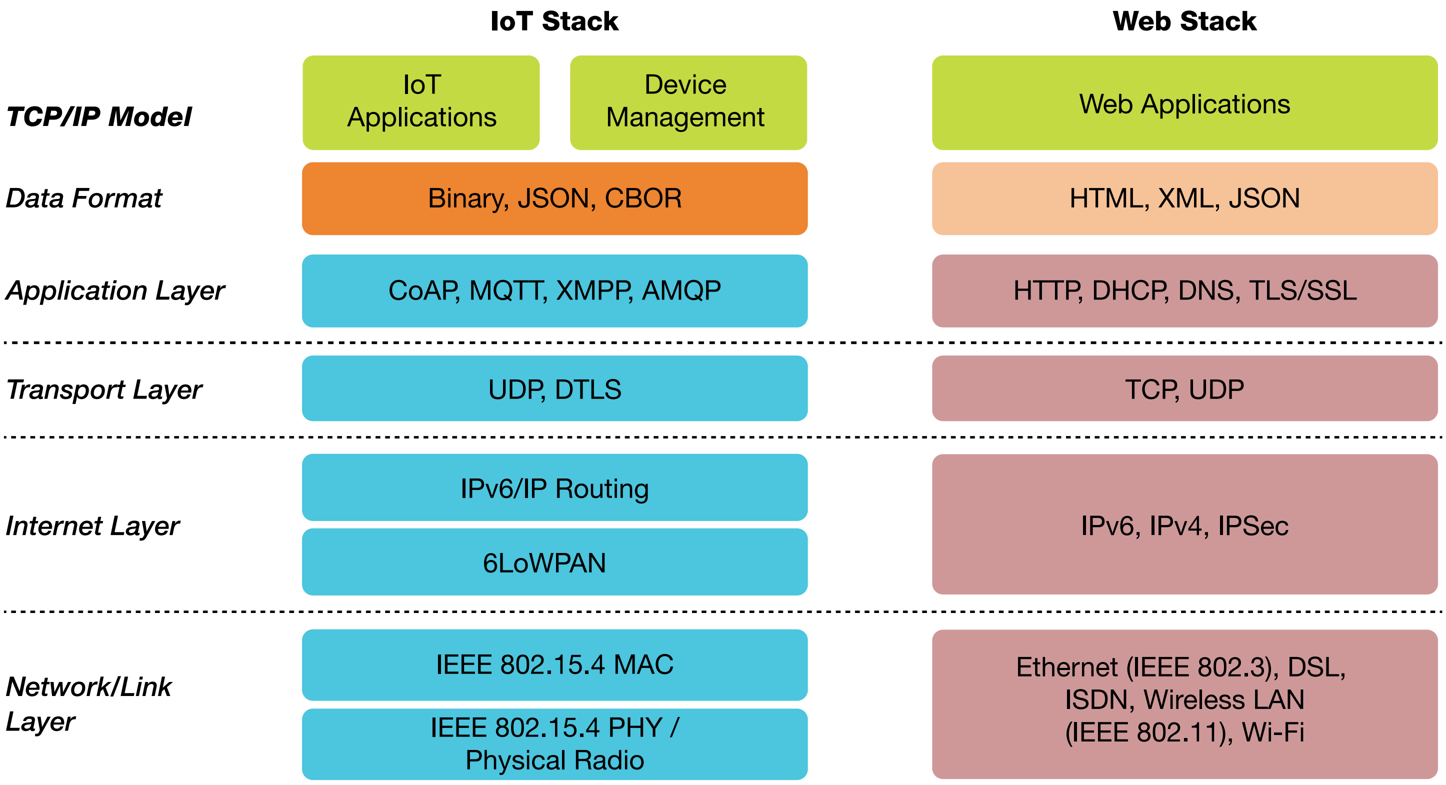
According to Industry Analysts there will be nearly 26 billion devices on the IoT by 2020. Cisco estimates that 50 billion devices and objects will be connected to the internet by 2020. Other vendors have slightly different predictions. Irrespective of these differences, it is very clear that the number of devices/things connected would be in the range of billions, not millions.

One of the biggest challenges faced by businesses, architects, and developers while dealing with IoT projects is selecting the technology stack and tools – this stems from the fact that standardization in the IoT protocols is virtually non-existent. The root of the problem is the constrained environment of IoT characterized with low memory availability, low power, low bandwidth requirement, and high packet loss – combined, these do not allow TCP/IP stack and web technologies to be used easily for IoT.

However, to solve this challenge, there are hundreds of proprietary protocols in IoT, M2M (Machine to Machine) and Home Automation space such as ZigBee and Z-Wave. Though these protocols are supported by an alliance of product vendors, they are not standardized like TCP, IP, HTTP or SMTP.

Although the scenario is still a bit cloudy, a set of open, standardized set of protocols have started to emerge. Most bodies such as IEEE, IETF or W3C have standardized protocols such as 6LoWPAN or CoAP. In the long run, these protocols would emerge successful like the open standardized web standards used by the web today.

The figure below shows the IoT protocols that have been standardized for each layer of TCP/IP model including Network, Internet, Transport and Application Layers.



	IoT Stack		Web Stack
TCP/IP Model	IoT Applications	Device Management	Web Applications
Data Format	Binary, JSON, CBOR		HTML, XML, JSON
Application Layer	CoAP, MQTT, XMPP, AMQP		HTTP, DHCP, DNS, TLS/SSL
Transport Layer	UDP, DTLS		TCP, UDP
Internet Layer	IPv6/IP Routing 6LoWPAN		IPv6, IPv4, IPSec
Network/Link Layer	IEEE 802.15.4 MAC IEEE 802.15.4 PHY / Physical Radio		Ethernet (IEEE 802.3), DSL, ISDN, Wireless LAN (IEEE 802.11), Wi-Fi

Figure 1 – Standardized IoT Protocols

Let's take a look at the protocols bottom up the stack.

Network / Link Layer

IEEE 802.15.4

IEEE 802.15.4 is a standard for wireless communication that defines the Physical layer (PHY) and Media Access Control (MAC) layers. It is standardized by the IEEE (Institute for Electrical and Electronics Engineers) similar to IEEE 802.3 for Ethernet, IEEE 802.11 is for wireless LANs (WLANs) or Wi-Fi.

802.15 group of standards specifies a variety of wireless personal area networks (WPANs) for different applications (For instance, 802.15.1 is Bluetooth). IEEE 802.15.4 focuses on communication between devices in constrained environment with low resources (memory, power and bandwidth).

Internet Layer

6LoWPAN

6LoWPAN is the secret sauce that allows larger IPv6 packets to flow over 802.15.4 links that support much smaller packet sizes. 6LoWPAN is the acronym of IPv6 over Low Power Wireless Personal Area Networks. So 6LoWPAN as the name implies is an adaptation layer that allows transport of IPv6 packets over 802.15.4 links. Without 6LoWPAN IPv6, internet protocols would not work in these Low Power Wireless Personal Area Networks that uses IEEE 802.15.4.

6LoWPAN is an open standard defined under RFC 6282 by the Internet Engineering Task Force (IETF), the body that defines many of the open standards used on the internet such as UDP, TCP and HTTP to name a few.

As mentioned previously, an IPv6 packet is too large to fit into a single 802.15.4 frame. What 6LoWPAN does to fit an IPv6 packet in 802.15.4 frame is -

- Fragmentation and Reassembly - It fragments the IPv6 packet and sends it through multiple smaller size packets that can fit in an 802.15.4 frame. On the other end, it reassembles the fragmented packets to re-create the IPv6 packet
- Header Compression – Additionally it also compresses the IPv6 packet header to reduce the packet size

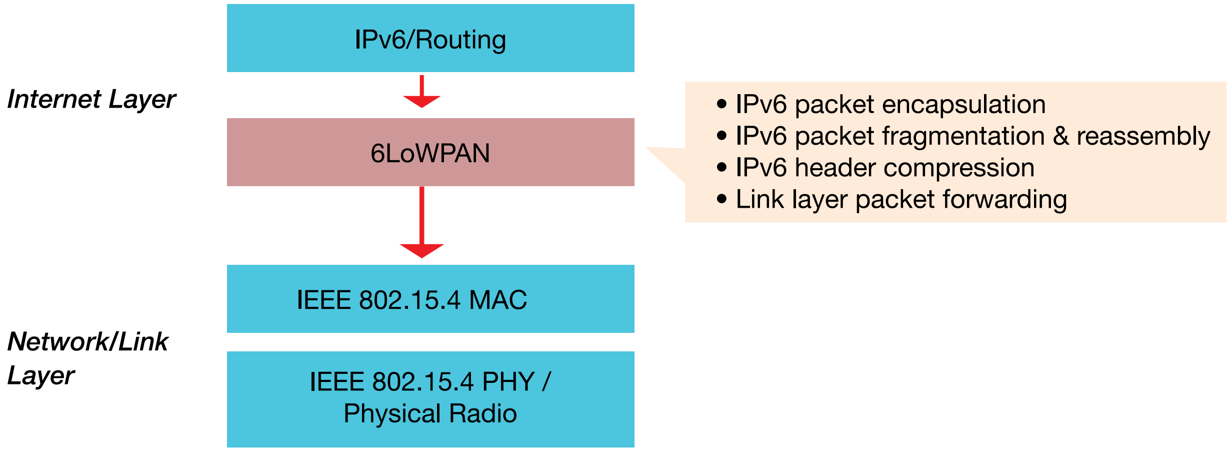

Internet Layer
IPv6/Routing
6LoWPAN
• IPv6 packet encapsulation
• IPv6 packet fragmentation & reassembly
• IPv6 header compression
• Link layer packet forwarding
Network/Link Layer
IEEE 802.15.4 MAC
IEEE 802.15.4 PHY / Physical Radio

Figure 2 - IPv6 sent into 802.15.4 MAC by 6LoWPAN

Transport Layer

UDP

While TCP is used predominantly in Internet as Transport Layer Protocol (except gaming or video streaming where User Datagram Protocol or UDP is used), most IoT scenarios are well suited for UDP. UDP is a much lighter protocol compared to TCP. UDP is a connection protocol and does not come with resiliency features of TCP, such as guaranteed packet delivery. On the other hand, UDP is much faster than TCP, the header size is much smaller than TCP – making it suitable for the constrained environment of devices and sensors.

Higher level Application Layer IoT protocols like CoAP use UDP rather than TCP.

DTLS

DTLS or Datagram Transport Layer Security is a TLS/SSL counterpart that runs on UDP. The way TLS/SSL takes care of security for TCP communication, DTLS provides the same security features on UDP or Datagrams.

Application Layer

CoAP

CoAP or Constrained Application Protocol is a specialized Web Transfer Protocol for constrained nodes and constrained networks on the IoT. CoAP is an Application Layer protocol in the TCP/IP model (Web uses HTTP as an Application Layer protocol).

The term “Constrained” is used because it is designed specifically from the ground up to work well in constrained environments. The devices, sensors and actuators in IoT operate in a constrained environment with low memory, low power, low bandwidth, and high rate of packet failure. HTTP was not designed to work in this sort of environment, so HTTP, which is relatively heavyweight with large header size and text encoding struggles to work in IoT constrained environment.

This is where CoAP comes to play – CoAP has been standardized by IEFT (The Internet Engineering Task Force) Constrained RESTful Environments (CoRE) Working Group. Think of CoAP as web protocols for devices.

CoAP can be transparently mapped to HTTP. Following are the similarities between CoAP and HTTP –

- CoAP follows the same request-response pattern used by HTTP that we all are very familiar with. The CoAP client (a smartphone, for example) sends a request to the CoAP server (device/things) and the server then sends a response back
- Like the web, devices are addressed using IP address and port number. Access to services exposed by the device is via RESTful URLs
- CoAP uses familiar HTTP features like Methods (Get, Post, Put, and Delete), Status Codes, URLs and content-type
- CoAP supports discovery so that IoT devices/things could be discovered
- CoAP provides simple proxy and caching capabilities

CoAP has a few differences as well -

- CoAP runs on UDP as compares to HTTP, which typically uses TCP. UDP is lighter than TCP and has less overhead
- CoAP supports only Get, Put, Post and Delete methods. CoAP uses small and reduced set of headers (header size is limited to 4 bytes), and HTTP status codes to be lightweight
- CoAP supports confirmable and non-confirmable message types

In the example given below, to get the temperature from the thermostat (which acts like a server), the client, which is a smartphone sends a GET request. The URL uses RESTful architecture - clearly indicating the device name, sensor data it is looking for, etc. The thermostat or the server responds back with the current temperature.

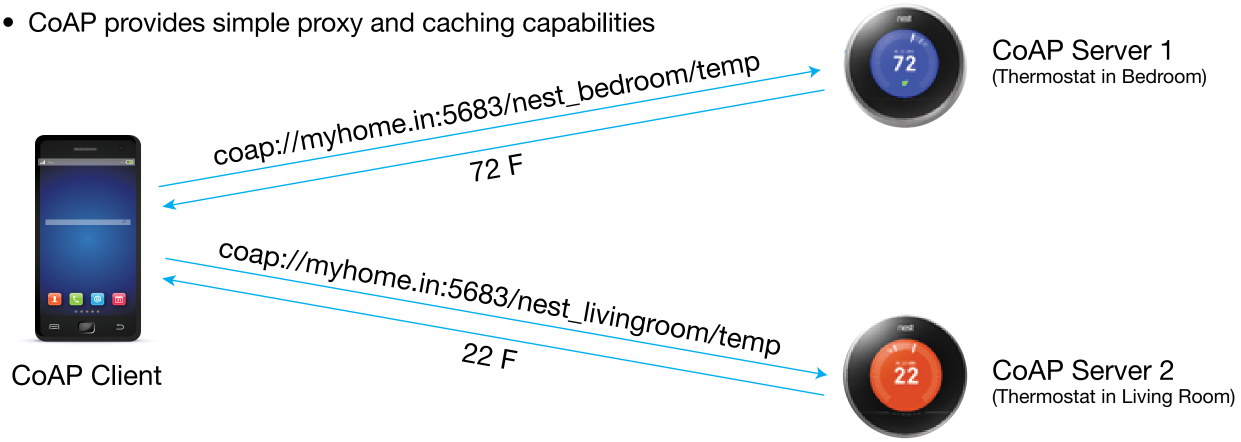

coap://myhome.in:5683/nest_bedroom/temp
72 F
72
CoAP Server 1
(Thermostat in Bedroom)
coap://myhome.in:5683/nest_livingroom/temp
22 F
22
CoAP Server 2
(Thermostat in Living Room)
CoAP Client

Figure 3 – COAP using GET and RESTful URLs

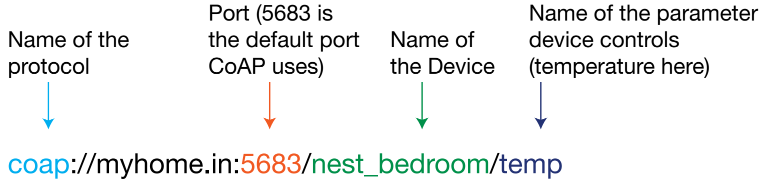

Name of the protocol
Port (5683 is the default port CoAP uses)
Name of the Device
Name of the parameter device controls (temperature here)
coap://myhome.in:5683/nest_bedroom/temp

Figure 3.2 – CoAP using URLs following RESTful services

CoAP also adds the feature of "observation", which HTTP lacks. This is very useful in IoT scenarios. In the above example where the CoAP client, which is a smartphone app, is requesting temperature from the thermostats, it can also specify that it should notify when the temperature changes – this functionality is called "Observe:" When the temperature changes, the CoAP server (in this case the thermostat), notifies the client. This was not possible before the recent server-sent event and WebSocket specification in HTTP.

CoAP libraries exists practically in all programming languages like C, Java, C#, Python, JavaScript, Go, Erlang, Ruby, etc. There are libraries for iOS and Android as well. Coap.technology website lists many of them.

MQTT

MQTT stands for Message Queue Telemetry Transport. MQTT is a publish-subscribe based "light weight" messaging protocol for IoT and M2M (Machine-to-Machine communication). To put it simply, MQTT is the AMQP or JMS for the constrained environment of IoT. Andy Stanford-Clark and Arlen Nipper invented MQTT back in 1999, when their use case was to create a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over a satellite connection.

MQTT uses a broker-based pub-sub architecture in the constrained IoT environment similar to other messaging products that exist in the Web and Client Server world.

So there is:

- **MQTT Publisher -** a sensor or device in IoT world that publishes a piece of information
- **MQTT Subscriber –** anyone who is interested to subscribe and receive a piece of information they're interested in. It could be a smartphone, a wearable device or even other devices that are interested to know about other devices around and change themselves based on self-learning
- **MQTT Broker –** An intermediary that receives information from publisher and forwards them to the subscribers

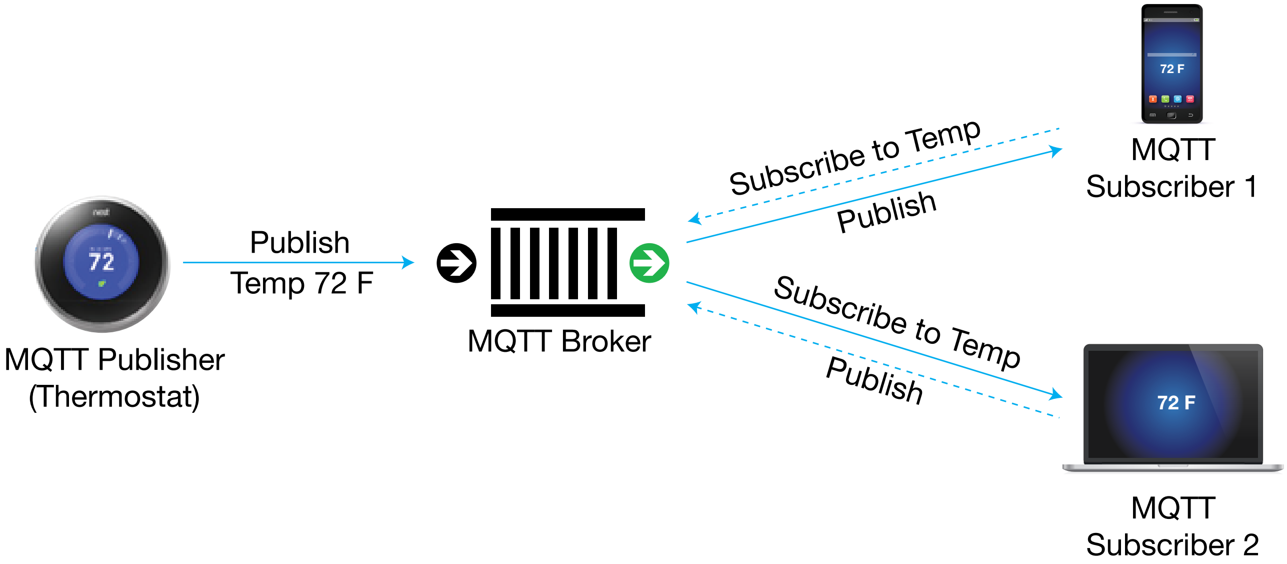


Figure 4 – MQTT in the world of IoT

MQTT provides subject/topic-based, content-based, and type-based filtering. The above example uses topic-based filtering where both the subscribers have subscribed for the topic "temp" to the broker. When any device or publisher publishes a message with topic "temp", the broker sends the message to all subscribers who have subscribed for temp – in this case both Subscriber 1 and Subscriber 2.

MQTT is actively promoted and supported by industry giants and it is part of many popular messaging suite of products. There are also MQTT specific brokers like Mosquitto and HiveMQ. MQTT libraries are available in many programming languages – one of the very prominent MQTT Library is Eclipse Paho. MQTT.org website lists many of them.

Data Format

CBOR

CBOR (Concise Binary Object Representation) is a data format designed with the goal of providing IoT with very small message sizes. CBOR is based on the wildly successful JSON data model. While JSON uses text encoding, CBOR uses binary encoding that results in compact message size.

Note: This article does not discuss IoT protocols for device management. There are separate protocols like OMA Lightweight M2M, which is an industry standard for device management of IoT and M2M devices.

Conclusion

The future of IoT lies in standardizing the protocols used across network stack. These set of open and standard protocols like CoAP, MQTT and 6LoWPAN would eventually become as successful as the TCP/IP stack used across web and Internet.

Appendix 1: CoAP Details

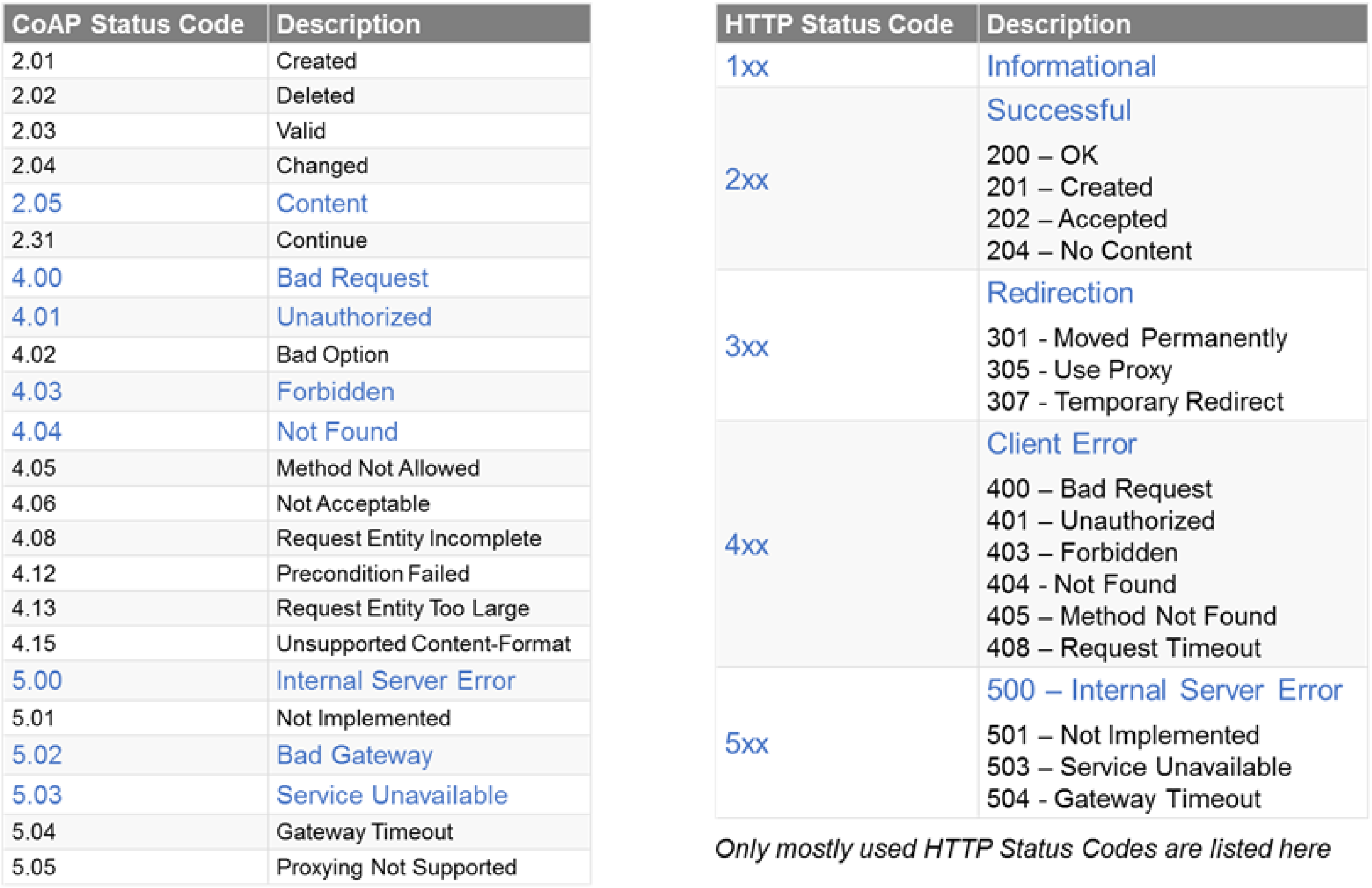
CoAP Message Types

CoAP defines four different types of messages.

- **CON or Confirmable Message**
  A confirmable message requires a response, either a positive acknowledgement or a negative acknowledgement. In case acknowledgement is not received, retransmissions are made until all attempts are exhausted.
- **NON or Non-Confirmable Message**
  A non-confirmable request is used for unreliable transmission (like a request for a sensor measurement made periodically. Even if one value is missed, the impact is not significant). Such a message is not generally acknowledged.
- **ACK or Acknowledgement**
  It is sent to acknowledge a confirmable (CON) message.
- **RST or Reset**
  This represents a negative acknowledgement and literally means resetting. It generally indicates some kind of failure (example - unable to parse received data).

CoAP Status Codes

As mentioned earlier CoAP status codes are directly influenced by HTTP status codes. CoAP uses X.YY format for status codes. So HTTP's "404 - Resource Not Found" status code is "4.04 Not Found" in CoAP. The table below shows the similarity between CoAP and HTTP status codes.



CoAP Status Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
2.31	Continue
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method Not Allowed
4.06	Not Acceptable
4.08	Request Entity Incomplete
4.12	Precondition Failed
4.13	Request Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported

HTTP Status Code	Description
1xx	Informational
2xx	Successful 200 – OK 201 – Created 202 – Accepted 204 – No Content
3xx	Redirection 301 - Moved Permanently 305 - Use Proxy 307 - Temporary Redirect
4xx	Client Error 400 – Bad Request 401 – Unauthorized 403 – Forbidden 404 - Not Found 405 – Method Not Found 408 – Request Timeout
5xx	500 – Internal Server Error 501 – Not Implemented 503 – Service Unavailable 504 - Gateway Timeout

Only mostly used HTTP Status Codes are listed here

Figure 5 – CoAP and HTTP Status Codes

Appendix 2: Using CoAP Libraries

CoAP libraries exist in almost all programming languages like C, Java, C#, Python, JavaScript, Go, Erlang, and Ruby. There are libraries for iOS and Android also.

For example, we would use the Node.js JavaScript CoAP library (both for the client and server). This library would seem very familiar if you have used Node. More details about the library could be found at https://github.com/mcollina/node-coap

Installing the Node CoAP Library

You need to have Node and npm installed on your machine as prerequisites. You can download and install them from nodejs.org. To install Node CoAP library use “npm install coap” in the terminal/command window. Once Node CoAP library is installed you can use it in your node code.

Connecting to a remote CoAP server

```
// Load coap module - similar to node http module
var coap = require(‘coap’);


// coap.me hosts CoAP servers. The default CoAP port
is 5683
var remoteServerUrl = ‘coap://coap.me:5683/large’;

var request = coap.request(remoteServerUrl);

var request = coap.request(localServerUrl);

request.on(‘response’, function(response) {

    response.pipe(process.stdout);

    response.on(‘end’, function() {

        process.exit(0);

    });

});

request.end();
```

Local CoAP server and client

Server:

```
// Load coap module - similar to node http module

var coap = require(‘coap’);

// Create a CoAP Server object - similar to creating
a Http Server object in node

// var http = require(‘http’);

// var httpServer = http.
createServer(function(req,res){ });

// The default CoAP port is 5683

var coapServer = coap.createServer(function(req,
res){

    res.write(‘Hello’ + request.url.split(‘/’)[1] +
‘\n’)

    res.end(‘Hello from CoAP server’);

});

// Keeps the CoAP server running

coapServer.listen(function(){

    console.log(‘CoAP server started!’);

});
```

Client:

```
var coap = require(‘coap’);

var localServerUrl = ‘coap://localhost:5683/world’;

var request = coap.request(localServerUrl);

request.on(‘response’, function(response) {

    response.pipe(process.stdout);

    response.on(‘end’, function() {

        process.exit(0);

    });

});

request.end();
```

Further Reading

https://en.wikipedia.org/wiki/Internet_of_Things

https://en.wikipedia.org/wiki/IEEE_802.15.4

https://en.wikipedia.org/wiki/6LoWPAN

http://postscapes.com/internet-of-things-protocols

http://coap.technology

http://mqtt.org

https://www.utwente.nl/ewi/dacs/colloquium/archive/2010/slides/2010-utwente-6lowpan-rpl-coap.pdf

Aniruddha Chakrabarti

AVP Digital Practice

Aniruddha has 16 years of IT experience spread across systems integration, technology consulting, IT outsourcing, and product development. He has extensive experience of delivery leadership, solution architecture, presales, technology architecture, and program management of large scale distributed systems.

As AVP, Digital Practice in Mphasis Aniruddha is responsible for Presales, Solution, RFP/RFI and Capability Development of Digital Practice. He has been the Lead Architect for many large scale distributed systems. His interests include digital, cloud, mobility, IoT, distributed systems, web, open source, .NET, Java, programming languages and NoSQL.

About Mphasis

Mphasis is a global Technology Services and Solutions company specializing in the areas of Digital and Governance, Risk & Compliance. Our solution focus and superior human capital propels our partnership with large enterprise customers in their Digital Transformation journeys and with global financial institutions in the conception and execution of their Governance, Risk and Compliance Strategies. We focus on next generation technologies for differentiated solutions delivering optimized operations for clients.

For more information, contact: marketinginfo@mphasis.com

USA
460 Park Avenue South
Suite #1101
New York, NY 10016, USA
Tel.: +1 212 686 6655
Fax: +1 212 683 1690

UK
88 Wood Street
London EC2V 7RS, UK
Tel.: +44 20 8528 1000
Fax: +44 20 8528 1001

INDIA
Bagmane World Technology Center
Marathahalli Ring Road
DoddanakundhiVillage, Mahadevapura
Bangalore 560 048, India
Tel.: +91 80 3352 5000
Fax: +91 80 6695 9942

www.mphasis.com

VAL 10/12/15 US LETTER BASIL 3724