# Chapter 6: Implementing WICED to use the AWS IOT Cloud

## Objective

At this end of Chapter 6 you will understand:

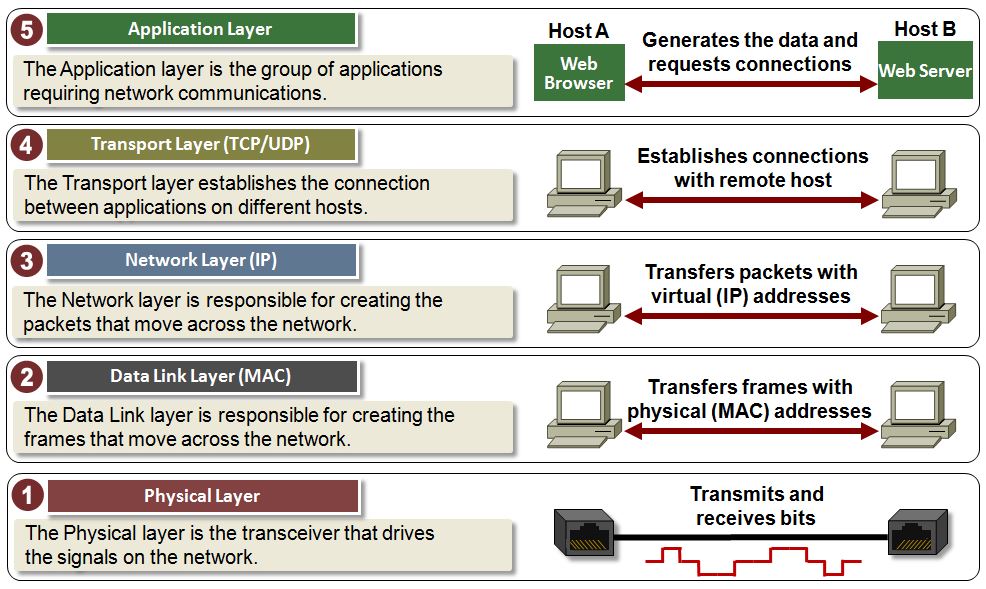
* How MQTT works:
  + How MQTT fits into the TCP/IP Networking Stack
  + What a Topic is
  + What a message broker is
  + What a Publisher is
  + What a Subscriber is
  + Understand QOS
  + MQTT security
  + What are the ports used by MQTT
  + What is keep alive
* How to use the JavaScript Object Notation (JSON) language.
* How the Amazon AWS MQTT Cloud works
  + Know how to provision “things” in the Amazon Cloud by creating things, policies and certificates
  + Understand what a “shadow device” is
  + Know how to use AWS MQTT Client to subscribe and publish to topics
  + Understand the scope of systems that can be implemented in the cloud (SNS, Database etc)
  + Understand the role of public and private key cryptography in the AWS cloud
* Understand in **DETAIL** how to write WICED firmware to interact with the AWS IOT Cloud

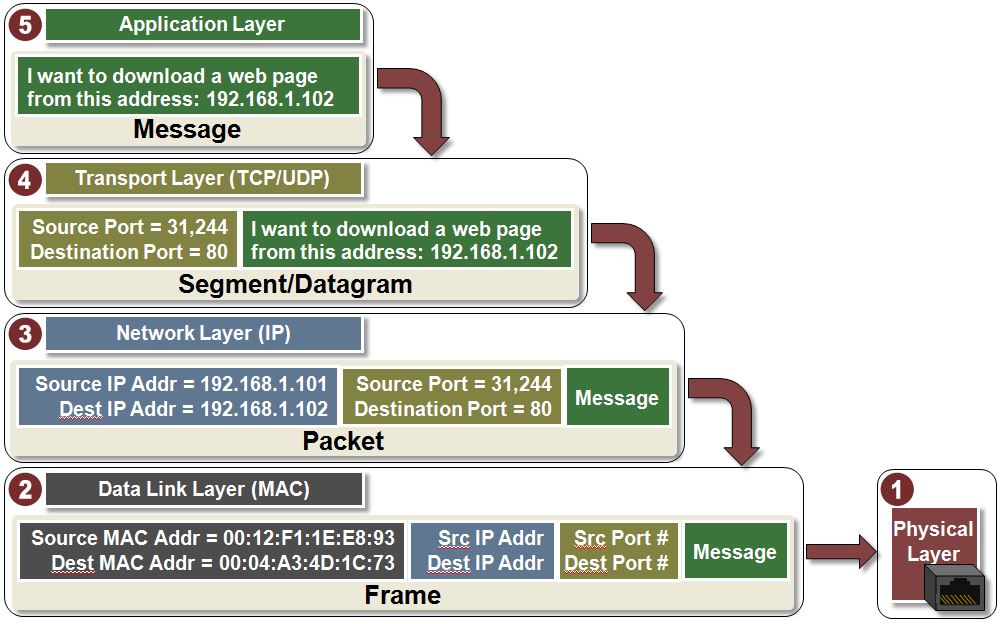
## Time: 4 Hours

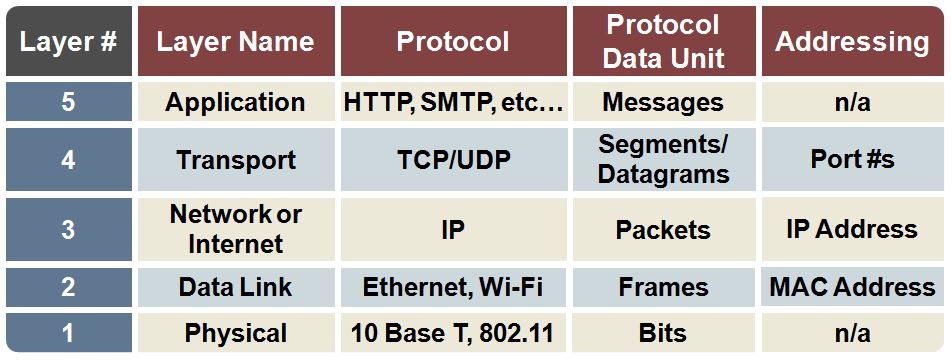
## Fundamentals

### Networking Stack the Application Layer

http://microchip.wikidot.com/tcpip:tcp-ip-five-layer-model







### (Application Protocol) MQQT

#### Topic

#### Publisher

#### Subscriber

### (Data Format) JSON

### (Cloud) AWS (MQTT)

#### <http://www.slideshare.net/PeterREgli/mq-telemetry-transport>

#### Introduction + Setup

#### Shadow

Legal shadow topic format of topic

<http://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-mqtt.html>

(format of json that you need to “publish” <http://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-document-syntax.html>

## Exercise(s)

* Provision a new thing in the AWS IOT Cloud, establish its policy and credentials
* Build the publisher app (modify dct and MQTT functions)
* Subscribe to the topic using the AWS MQTT client
* Explain in detail the firmware flow for publisher app
  + How do the MQTT library functions (e.g. wiced\_mqtt\_publish ) get into your project?
  + Why did the firmware author create a function called “wait\_for\_result”?
  + What mechanism does the “wait\_for\_result” function use to cause the “wait”?
  + What are the 7 WICED MQTT events? What file are they defined in?
  + Do you have to name the client certificate client.cer? How would you change the name?
  + What is the naming convention used to differentiate WICED MQTT library functions versus wrappers around those function in the publisher app?
  + What is the sequence of function calls to open a connection?
  + What prevents a hung connection from deadlocking the publisher app?
  + What function is called when the button is pressed?
  + How does the button callback unlock the main thread?
  + What is the name of the flag that prevents the firmware from sending multiple button presses before the publish is finished?
  + Are all message sent to the AWS IOT MQTT Message broker required to be in JSON format?
  + Add debugging prints to the function mqtt\_connetion\_event\_cb
  + To open a connection, the flow looks like this:
    - mqtt\_conn\_open calls wiced\_mqtt\_connect with a call back of mqtt\_connection\_event\_cb
    - mqtt\_conn\_open calls wait\_for\_result which gets the msg\_semaphore (which halts everything)
    - When Amazon responds, the mqtt\_connecton\_event\_cb occurs and wakes up the thread
    - The mqtt\_connection\_event\_cb updates the “expected\_event” global variable and set the msg\_semaphore then returns
    - Back in wait\_for\_event If the message WICED\_MQTT\_EVENT\_TYPE\_CONNECT\_REQ\_STATUS is received then it then it returns WICED\_SUCCESS else it returns WICED\_FAIL
    - What is the equivalent flow to publish?
* Implement the subscriber on another kit (don’t forget to update the DCT parameters and the MQTT parameters)
* Change the subscriber APP to print on the screen the messages that are received
* Modify the publisher app to publish updates to the shadow
* Modify the subscriber app to get updates from the AWS Message Broker.
* Implement the shadow application
* “Bake-In” the AWS security credential into the shadow app.
* “Bake-In” the WiFi credentials into the shadow app