Chapter 7a: Application Layer Protocols for Cloud Connectivity

Objective

At the end of Chapter 7 you will understand how to build a complete WICED IoT App using one of the cloud application protocols (MQTT, COAP, AMQP, HTTP or Sockets). In addition, you will have a big picture understanding of each of those protocols. You will also be introduced to one of the Cloud vendors such as Amazon AWS, IBM Bluemix, Ali Cloud, or Microsoft Azure.

This section contains the foundation information required to understand the chapter(s) that pertain to specific cloud protocols. At the end of this section you should understand the basics of the Application Protocols HTTP, MQTT, AMQP, and COAP.

Time: 3 ¾ Hours

Fundamentals

The "Cloud"

What is the Cloud? The Cloud is a simple name for a giant amalgamation of all the stuff that you need in order to provide web sites and other network based services (e.g. iTunes). Why do you need the Cloud? When you try to service large numbers of people and devices you have a very difficult and expensive problem. In order to have a fast and always available system you need to have enough networks, disk drives, computers and people (to run it all). The solution to this problem is a standardized, shared, scalable system: The Cloud.

The term "The Cloud" generally includes:

1. Networks (high bandwidth, worldwide, distributed)
2. Storage (disks, databases)
3. Servers (running Windows and Unix)
4. Security
5. Scalability
6. Load Balancing
7. Fault tolerance (redundancy)
8. Management tools (reports, user identity management, etc.)
9. Software (Web servers, APIs, Languages, Development tools etc.)

In order to make something interesting you need to be able to hook up your IoT device(s) (the "T" in IoT) to the Cloud (the "I" in IoT) which is the goal of this chapter.

Application Layer Protocols

How do you get data to and from the Cloud? Simple, there are a number of standardized application layer protocols to do that task.

Hyper Text Transfer Protocol (HTTP)

HTTP is a text-based Application Layer Protocol that operates over TCP Sockets. It can perform the following functions:

- GET (retrieve data) from a specific place
- POST (put data) to a specific place
- HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATH (less commonly used)

To initiate these commands, you open a socket typically to TCP port 80, send the text based command (CRLF terminated) and read the replies. This request/reply protocol is used for every command. Replies are sent with a resulting Content-Type string which indicates the type of data encoding for the response. The content-type string uses a Multipurpose Internet Mail Extension (MIME) type to indicate the type of data being received (e.g. text/html or image/jpeg).

For instance, you can send an HTTP GET request to open "/index.html" on www.example.com:

- GET /index.html HTTP/1.1

Example.com will respond with:

- HTTP/1.1 200 OK
- Date: Mon, 23 May 2005 22:38:34 GMT
- Content-Type: text/html; charset=UTF-8
- Content-Encoding: UTF-8
- Content-Length: 138
- Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
- Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
- ETag: "3f80f-1b6-3e1cb03b"
- Accept-Ranges: bytes
- Connection: close
- <html>
- <head>
- <title>An Example Page</title>
- </head>
- <body>
- Hello World, this is a very simple HTML document.
- </body>
- </html>

It is possible (and semi-common) to build IoT devices that use HTTP to "PUT" their data to webservers in the cloud and "GET" their instructions/data from webservers. However, HTTP is somewhat heavy (that is, bandwidth intensive) and is generally being displaced by other protocols that are more suited to IoT.

Message Queueing Telemetry Transport (MQTT)

MQTT is a lightweight messaging protocol that allows a device to **Publish Messages** to a specific **Topic** on a **Message Broker**. The Message Broker will then relay the message to all devices that are **Subscribed** to that **Topic**.

The format of the messages being sent in MQTT is unspecified. The message broker does not know (or care) anything about the format of the data and it is up to the system designer to specify an overall format of the data. All that being said, JavaScript Object Notation (JSON) has become the lingua franca of IoT.

A Topic is simply the name of a message queue e.g. "mydevice/status" or "mydevice/pressure". The name of a topic can be almost anything you want but by convention is hierarchical and separated with slashes "/".

Publishing is the process by which a client sends a message as a blob of data to a specific topic on the message broker.

A Subscription is the request by a device to have all messages published to a specific topic sent to the client.

A Message Broker is just a server that handles the tasks:

- Establishing connections (MQTT Connect)
- Tearing down connections (MQTT Disconnect)
- Accepting subscriptions to a Topic from clients (MQTT Subscribe)
- Turning off subscriptions (MQTT Unsubscribe)
- Accepting messages from clients and pushing them to the subscribers (MQTT Publish)

MQTT provides three levels of Quality of Service (QOS):

- Level 0: At most once (each message is delivered once or never)
- Level 1: At least once (each message is certain to be delivered, possibly multiple times)
- Level 2: Exactly once (each message is certain arrive and do so only once)

MQTT operates on TCP Ports 1883 for insecure and 8883 for secure (TLS).

Cloud providers that support MQTT include Amazon AWS and IBM Bluemix.

Constrained Application Protocol (CoAP)

CoAP makes use of two message types, requests and responses, using a simple, binary, base header format. The base header may be followed by options in an optimized Type-Length-Value format. CoAP is by default bound to UDP and optionally to DTLS, providing a high level of communications security.

Any bytes after the headers in the packet are considered the message body, if any. The length of the message body is implied by the datagram length. When bound to UDP the entire message MUST fit within a single datagram. When used with 6LoWPAN as defined in RFC 4944, messages SHOULD fit into a single IEEE 802.15.4 frame to minimize fragmentation.

The mapping of CoAP with HTTP is also defined, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way.

Cloud providers that use CoAP include Samsung ARTIK.

Advanced Message Queuing Protocol (AMQP)

AMQP is a binary application layer protocol designed to efficiently support a wide variety of messaging applications and communication patterns. It provides flow controlled, message-oriented communication with message-delivery guarantees such as *at-most-once*, *at-least-once* and *exactly-once*, and authentication and/or encryption based on SASL and/or TLS. It assumes an underlying reliable transport layer protocol such as Transmission Control Protocol (TCP).

The AMQP specification is defined in several layers: (i) a type system, (ii) a symmetric, asynchronous protocol for the transfer of messages from one process to another, (iii) a standard, extensible message format and (iv) a set of standardized but extensible 'messaging capabilities.'

Cloud providers that use AMQP include Microsoft (e.g. Windows Azure), VMWare, and Redhat.

JavaScript Object Notation (JSON)

JSON is an open-standard format that uses human-readable text to transmit data consisting of attribute–value pairs. JSON supports the following data types:

- Double precision floating point
- Strings
- Boolean (true or false)
- Arrays (use "[]" to specify the array with values separated by ",")
- Key/Value (keymap) pairs as "key":value (use "{}" to specify the keymap) with "," separating the pairs

Key/Value values can be arrays as well as key/value maps

Arrays can hold Key/Value Maps

For example, a legal JSON file looks like this:

```
{
        "name" : "alan",
        "age" : 48,
        "badass" : true,
        "children":  ["Anna","Nicholas"],
        "address" : {
                "number":201,
```

```
            "street": "East Main Street",
            "city": "Lexington",
            "state":"Kentucky",
            "zipcode":40507
        }
}
```

Note that carriage returns and spaces (except within the strings themselves) don't matter. For example, the above JSON code could be written as:

{"name":"alan","age":48,"badass":true,"children":["Anna","Nicholas"],"address":{"number":201,"street":"East Main Street","city":"Lexington","state":"Kentucky","zipcode":40507}}

While this is more difficult for a person to read, it is easier to create such a string in the firmware when you need to send JSON documents.

For receiving JSON documents, the WICED SDK has a JSON parser built in. You can find these functions in the directory "Utilities→JSON_parser"

Further Reading

[2] RFC2045 – “Multipurpose Internet Mail Extensions”; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc2045

[4] RFC2616 – “Hypertext Transfer Protocol (HTTP) “ ; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc2616

[5] RFC7159 – “The Javascript Object Notation (JSON) Data Interchange Format”; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc7159

[6] MQTT - http://mqtt.org/

[7] RFC7959 – “The Constrained Application Protocol (CoAP)” ; Internet Engineering Task Force (IETF) - https://tools.ietf.org/html/rfc7252

[8] AMQP - http://www.amqp.org/