# Data Transmission Protocols

The LogiSteps data, after being transmitted from the embedded device, will have to be sent from the Android application over the internet to the web application. There are many data protocols that can be used to send data over the internet. For LogiSteps, an IoT like system, we will look at protocols useful for IoT applications and data. Also, as we intend to use an Android app, the data protocols that have existing libraries for Android will be focused on. So, of the protocols found in broad research that fit the desire for IoT data, and android libraries, there are three that stand out and will be the options looked at.

## MQTT

First, there is MQTT which is Message Queueing Telemetry Transport. It is a lightweight (low bandwidth), publish/subscribe protocol that runs over TCP/IP. It is ideal for one-to-many information delivery. MQTT offers three QoS, or qualities of service, for its messages. At most once, the information might be lost. At least once, duplicate messages might occur. And Exactly once, no loss or duplicity. So, there is low bandwidth usage, medium, and high. Of note is its message header, which is a fixed 2 bytes in length. With these features MQTT was designed for constrained, low bandwidth devices, while assuring reliability in its messages. MQTT can also ensure the security of its data using SSL, or other data encryption methods.

Those are the advantages of MQTT for LogiSteps, low bandwidth, and reliability. We will want to ensure we receive every data point we send from the embedded system. What is likely the best advantage MQTT shoes over the other protocols is power consumption, as it is lighter weight than HTTP for example, it will consume less of the user’s phone’s battery.

A Stephen Nicholas performed a power profiling comparing HTTPS and MQTT. What he found was that for a connection receiving sporadically MQTT used about 70% as much power as HTTPS over 3G, and over WIFI only used about 10% as much power. Sending as fast as possible saw MQTT using 2% less battery/hour over 3G. For sending data he found that MQTT used 1% less battery/hour over 3G, and 2% less battery/hour over WIFI. So, for LogiSteps MQTT clear advantage is it will eat up less of the user’s phone’s battery life. The expected usage of LogiSteps and its app could lead to a considerable saving of battery life for the user, were MQTT to be the chosen data protocol.

For MQTT on android there are a few libraries out there that can be used. There is the Moquette library on GitHub which has almost one-thousand stars. Moquette also uses Netty for encoding/decoding. This library would be fairly easy to begin using. There is also an MQTT client by IBM using either IBM® MessageSight or IBM WebSphere® MQ, which has a lot of tutorial/development help.

The biggest problem with MQTT for LogiSteps is the publish/subscribe method is not particularly great for our system. As the LogiSteps system will use an Android app to send the data over the internet to our server, publish/subscribe is not ideal for the single connection between phone and server.

## CoAP

The next protocol of note is CoAP, or Constrained Application Protocol. Like MQTT CoAP is designed for IoT. CoAP is defined by the RFC7252, written by the Internet Engineering Task Force (IETF). CoAP’s big points are that it is built for nodes with small amounts of memory, and networks with high error rate, which can be translated to a phone app environment, and it is built for the IoT and many, many nodes. Its lightweight attributes include a small header of 4-bytes and requiring as little as 10KB of RAM and 100KB of code space. CoAP is also built on the REST model, and uses a request/response, so being very similar to HTTP can easily interface with HTTP services. And of course, CoAP data can be secured, with the default being 3072-bit RSA keys.

Like MQTT, CoAP has some open libraries that we could use for LogiSteps. The first is Californium. Californium is under the Eclipse Foundation and is well documented and supported. It implements CoAP (RFC 7252), the Observe draft (RFC 7641), the block wise transfers draft (RFC 7959), and DTLS1.2 for security. It provides CoAP-HTTP cross-proxy support. And it is designed as a scalable model for IoT.

Another implementation is SpitFireFox on GitHub, which is notable for, like Moquette of MQTT, encapsulating CoAP in Netty, the asynchronous, event-driven framework. This would be a fairly easy implementation to set up and begin using for LogiSteps.

The biggest problem for using CoAP with LogiSteps is that on the Android app, the constraint CoAP provides is unnecessary for LogiSteps, but it is still a possible option for data protocol use.

## HTTP

Of course, there is HTTP, HTTP/2, and HTTPS. They are the standards of internet communication. HTTP, Hyper Text Transfer Protocol, used by the entire Internet. HTTP is a stateless protocol, meaning there is no knowledge of what previously occurred on the network, that uses request-response protocol to communicate between client and server, and it operates on TCP/IP. HTTP is not designed with constraints in mind and has very large and complex headers. Another fault is HTTP is not a secure method of data transfer and communication.

However, there are the HTTP upgrades HTTPS and HTTP/2. Both of them are improvements and widely uses. HTTPS secures HTTP by using either Transport Lay Security (TSL), or previously Secure Sockets Layer (SSL). This and adds protection and privacy to the data, which is something LogiSteps will most likely want for its data.

HTTP/2 was created with several goals to improve HTTP. It adds the ability to allow clients and servers to choose their data protocol. It maintains compatibility with HTTP 1.1. And has improved upon the large overhead of HTTP, thus reducing latency. Of note is that HTTP/2 itself is not secure but does support HTTPS using TLS.

As HTTP is the world standard for internet communication there are many more libraries and clients for Android that LogiSteps could utilize for itself. For example, there is the Google HTTP Client Library for Java, supporting Android 1.5 and higher. There is OkHTTP which works as an HTTP & HTTP/2 client for android and has almost 30,000 stars on GitHub. And there is Retrofit which turns the HTTP API into a Java interface. Retrofit also has almost 30,000 stars on GitHub. Due to the wide spread usage of HTTP and all the libraries there are for it.

## Conclusion

So MQTT and CoAP have their advantages as communication protocols for IoT. However, HTTP is simply a more versatile protocol, and due to it being the global standard there are many libraries that make using HTTP and are very simple and easy to use. As LogiSteps does not require the IoT benefits provided by MQTT and CoAP specifically for the Android communication, LogiSteps will use HTTP, due to the fact that it’s the standard with far more developmental help to be had. However, it should be noted that should LogiSteps remove the Android phone app from the design, and instead send the data directly from the embedded device to the web service, CoAP would be the optimal choice. But on the Android application the benefits provided by HTTP’s widespread use just far outweigh the IoT benefits of the other protocols.

MQTT

<https://mqtt.org/faq>

[https://www.ibm.com/support/knowledgecenter/en/SSFKSJ\_7.5.0/com.ibm.mm.tc.doc/tc00000\_.htm](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.5.0/com.ibm.mm.tc.doc/tc00000_.htm%20)

[https://inductiveautomation.com/what-is-mqtt](https://inductiveautomation.com/what-is-mqtt%20)

[https://github.com/mqtt/mqtt.github.io/wiki/mqtt\_on\_the\_android\_platform](https://github.com/mqtt/mqtt.github.io/wiki/mqtt_on_the_android_platform%20)

Moquette: [https://github.com/andsel/moquette](https://github.com/andsel/moquette%20)

Power: <http://stephendnicholas.com/posts/power-profiling-mqtt-on-android>

<http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https>

CoAP

RFC7252: <https://tools.ietf.org/html/rfc7252>

<http://coap.technology/>

Californium: <https://github.com/eclipse/californium>

<https://projects.eclipse.org/projects/technology.californium>

SpitFireFox: <https://github.com/okleine/spitfirefox>

HTTP

OkHTTP: <https://square.github.io/okhttp/>

<https://github.com/square/okhttp>

Google: <https://developers.google.com/api-client-library/java/google-http-java-client/android>

RetroFit: <http://square.github.io/retrofit/>

<https://github.com/square/retrofit>