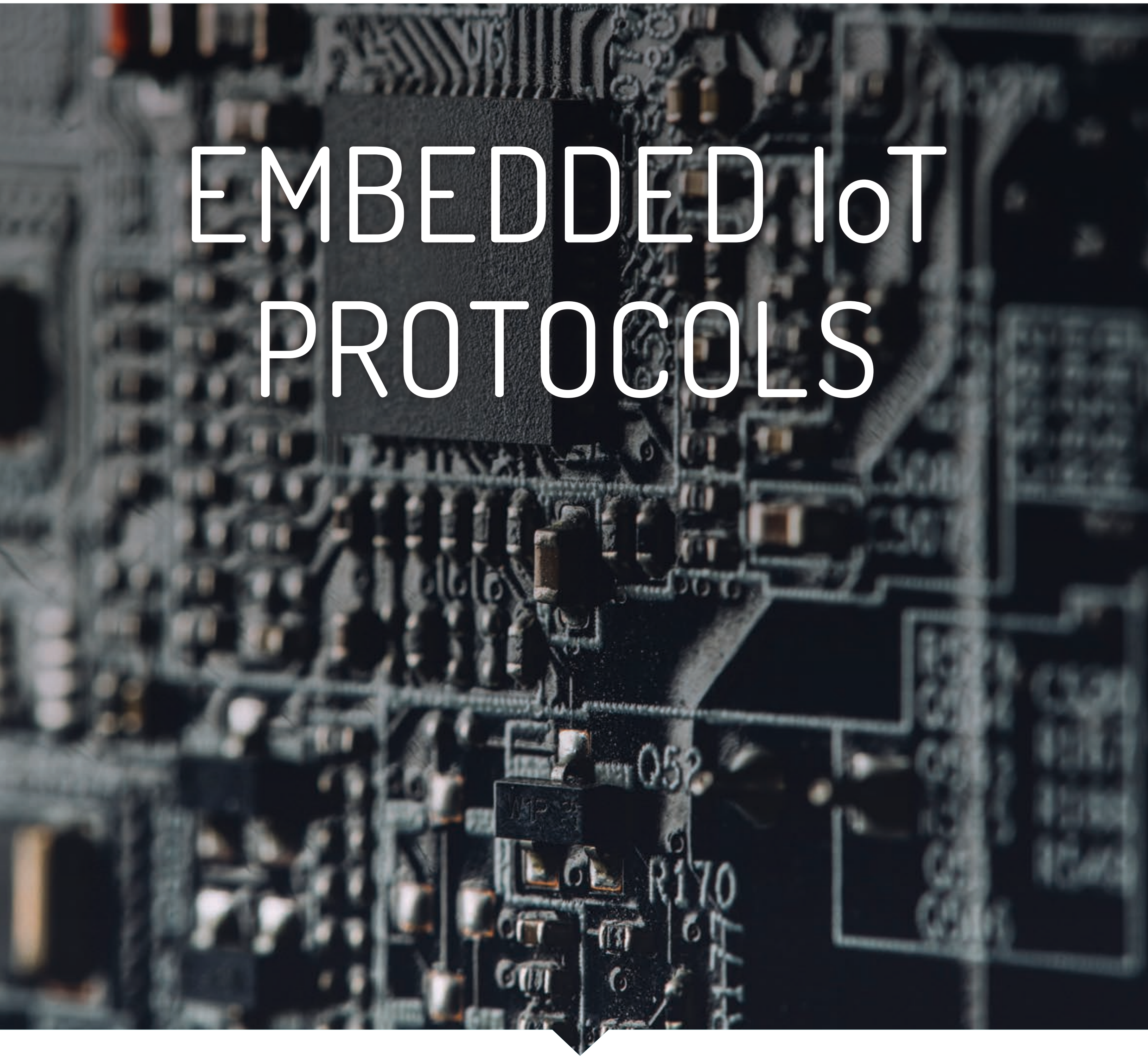
EXOSITE

USA HEADQUARTERS
275 Market St, Suite 535
Minneapolis, MN 55405
+1.612.353.2161

TAIWAN OFFICE
WenXin Road, Section 4
#955, 15F-5
Taichung, 406 Taiwan
+886.4.2247.1623



EMBEDDED IoT PROTOCOLS

WHITE PAPER

RELEASED: JANUARY 2016

Author: Patrick Barrett

EXOSITE

TABLE OF CONTENTS

POWERING THE IoT GENERATION OF BUSINESS

1. INTRODUCTION

What is the Internet of Things (IoT)? It is a global network of physical devices communicating with each other, usually through powerful cloud applications that add data processing, aggregation, and analysis to provide business insights and benefits that would not otherwise be feasible.

A classic example of an IoT product is a light bulb that can be controlled remotely through a connection to the Internet. This could be accomplished directly by a user on a remote device to turn on a light before getting home. It could also be accomplished automatically by intelligence in the cloud that uses data from other sensors around the house and other Internet services to predict what the user will want.

IoT also includes more complex and less obvious examples. For instance, weather forecasting and real-time weather tracking utilize IoT to collect and collate weather data from ubiquitous sensors and devices connected to the cloud. That weather data can then be processed and analyzed to make well-informed predictions and decisions.

In order for computers and IoT devices to interact intelligently in this way, they use protocols in much the same way that people share common languages that allow them to communicate. And, just as many languages throughout the world can express the same meaning but are completely incompatible with each other, computers and connected devices experience the same communication hurdles. Technology is continuously changing; new applications and devices require new protocols in order to communicate efficiently over the Internet, while providing users with the connectivity, security, and ease of deployment for their products and services through the cloud.

However, because IoT systems are usually comprised of low-cost, low-power, low-performance embedded devices, the suitability of protocols that are ideal or, in some cases, even possible to use is somewhat limited. Additionally, the need to communicate securely complicates things further. Potential attackers of IoT systems have substantially more computing power at their disposal thanks to advances in technology and pose a very critical challenge to secure communication, especially with the type of low-powered devices that are generally well-suited for IoT.

This white paper will provide an overview of some of the protocols that are common in IoT, and offer insights into important features that should be taken into consideration when selecting a protocol for use in a connected solution.

2. PROTOCOL FEATURES FOR IOT

At a high level, a good protocol for IoT enables the use of inexpensive, low-power devices and provides high performance in a variety of end-use applications while maintaining a high level of security and reliability. This is a good starting point to describe the needs of all IoT applications; however, the way these are achieved varies greatly depending on the needs of specific use cases. Below is an examination of several features that will provide a good baseline for most general use cases.

2.1 STANDARDIZATION

IoT protocols are like human languages; you can only communicate with others that speak the same language. Selecting standardized protocols for an IoT device increases the number of devices and services with which it can talk. Unfortunately, a single protocol has not been universally accepted for all use cases and probably never will. But in order to interoperate with other devices or avoid implementation of a protocol from scratch, selecting an off-the-shelf, standard protocol can save a significant amount of development time and money

2.2 LOW LATENCY

Latency is the amount of time it takes for a device to react to an external event. In the most basic example, latency could be measured by the length of time it takes a lamp to turn on when a user taps a button in a phone app. Latency can be a big problem for some applications if reactions are user facing. For IoT devices, network latency is by far the biggest component of total system latency. As such, it is one of the most important factors to consider when selecting specific protocols, as latency can have a significant impact on the performance of the overall system.

2.3 LOW DATA OVERHEAD

Data overhead is the amount of data that is transmitted above and beyond the actual information being conveyed from a sender to its recipient. In a real-world system, overhead will never be zero, but it is important to select a protocol that is not unnecessarily verbose for a variety of reasons.

First, keeping communications as lean as possible can help ensure systems remain economical. Many IoT devices use cellular connections for communication, where users pay for every byte of information that the device uses. This means there is a direct relationship between cost and data use.

Second, low data overhead can help avoid detrimental effects on performance even in cases where a device is not on a metered connection. Based on the general prediction that there will be billions of IoT devices developed in the next few years, it is reasonable to expect that a single house or business may soon contain hundreds of IoT devices. If individual devices consume large amounts of data, it could adversely affect the Internet connection they share, slow the performance of IoT applications and other Internet users, and result in a poor user experience.

2.4 LOW POWER USE

Low power use is closely related to low latency and low data overhead. The more time a device spends sending/receiving data or waiting for responses, the more power it draws from the battery. As a result, users are forced to replace or recharge batteries more often.

In addition to limiting the latency and data overhead to improve power use, it is possible to select protocols that are specifically designed to work around very aggressive sleeping of nodes and other power-saving features that make more traditional protocols unworkable.

3. PROTOCOLS

The Internet is made up of many different protocols, each with their own purpose. The standard model of the Internet breaks these protocols into layers based on the functions that they provide. This section will identify some of the protocols that make up the Internet as a whole, including relevant protocols from the Internet layer, transport layer, and application layer. It will also discuss the trade-offs of various application-layer protocols that are well suited for use in IoT, including the effects of the various transport- and Internet-layer protocols on which they depend.

3.1 INTERNET PROTOCOL

To be part of IoT, by definition, devices must have a connection to the Internet. This means all IoT devices must be able to talk Internet Protocol (IP). IP defines both how a device is uniquely identified on the Internet and how the series of networks that make up the Internet will get messages from one device to another.

Currently, there are two different versions of IP that are in active use, namely IPv4 and IPv6. The Internet as a whole is slowly migrating away from IPv4 towards IPv6. However, because the Internet is such a large and diverse collection of individually owned networks, it will be many years before support for IPv4 can be ignored.

While there are a large number of differences between IPv4 and IPv6, the vast majority of IoT developers will only notice one major difference: that addresses are much larger with IPv6. Where IPv4 used addresses that were 32 bits, IPv6 uses addresses that are 128 bits, allowing many more devices to be connected to the network.

Embedded developers generally do not need to worry about this layer of protocol, as an IP stack is often supplied by hardware interface vendors either in the hardware or as part of the software development kit (SDK). Developers simply have the option to choose whether or not the hardware they select supports IPv6. If the anticipated lifetime of a device exceeds the next three-to-five years, IPv6 support should be a requirement.

3.2 TRANSPORT PROTOCOLS

The next layer in the stack is the transport layer, which provides the common features that all higher-level protocols need. This prevents the need to re-implement these features over and over in each application protocol. This section contains a brief explanation of the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP) transport protocols. Although a transport protocol is usually never a direct requirement, the background information below provides an understanding of the trade-offs between UDP and TCP when selecting between the application protocols that leverage them.

3.2.1 UDP

UDP is the most basic transport protocol and provides just two features on top of IP. The first is data integrity that, through the use of checksums, ensures that the received data is the same as the data that was sent. The second is application muxing that, through the use of port numbers, allows the network stack to direct individual packets back to the application, or subsystem in an IoT device, that requested them.

This relative lack of features makes UDP incredibly simple to implement on top of IP, adding basically no hardware requirements to the application. However, it does lack some features that IoT applications are very likely to need, which the higher-level protocols will need to make up for.

3.2.2 TCP

TCP is UDP's big brother. At a high level, it provides reliable delivery of streams of data. It ensures that data is delivered exactly as it was sent and in the order it was sent. It also automatically re-requests any data that was lost or corrupted in transport, in addition to many more subtle features.

A TCP stack is much more complicated than UDP to implement and will require more hardware resources to use. However, this burden is often acceptable, as many applications will require some of the features that TCP provides. With the increasing availability of more powerful hardware at lower costs, the burden of TCP's additional features that are not required in a particular application are becoming less of a factor. There are even hardware network interfaces that include a full TCP stack, offloading all of the transport-level requirements from host devices.

3.3 APPLICATION PROTOCOLS

While there are many application protocols that can be used to send arbitrary data between a client and a server over the Internet (e.g., SMTP for email), some protocols are more suitable than others when it comes to IoT deployments. This section divides these protocols into two categories, older web protocols and newer protocols designed specifically for IoT, and discusses examples of each.

3.3.1 THE OLD STANDARDS

The development of the web provided some well-established protocols that can be used with IoT devices, including the Hypertext Transfer Protocol (HTTP) and the Extensible Messaging and Presence Protocol (XMPP). The main benefit of these protocols is that they are well understood and supported, especially for the server-side ecosystem. However, the reuse of existing technology with new applications does require some tradeoffs that should be considered.

3.3.1.1 HTTP

HTTP is the poster child of well-understood and well-supported protocols and is the application-layer protocol that runs almost the entirety of the web. HTTP uses a client-server model to describe how its requests are made. A client (traditionally a user's web browser) makes a request to a server asking for a resource, and the server responds with the current state of that resource. This model works well for traditional web browsing because there is a user directing the browser as to when it should make requests for certain resources. As such, it will be the lowest common denominator in supported protocols for the foreseeable future.

However, it also has its limitations and challenges when it comes to IoT use cases. When IoT devices are involved, there is not always a user to guide the actions, and the device must decide when it should request updates to a resource from the server on its own. And for most applications where a device needs to react to a remote input like an app turning on a light, an IoT device will always need to know the most up-to-date value so as to provide the low latencies that users will expect; nobody wants to wait thirty seconds for their lights to turn on after pressing a button to do so. That means that once a device decides to request an update and the server returns a response, it must repeat the request almost immediately since it has no other way of knowing if a server-side resource has changed.

Ideally, the server would instead notify the device when a resource changed. Unfortunately, the architecture of the Internet prevents this when using HTTP. Servers are almost always unable to send arbitrary messages to clients without first receiving a request from the client due to the security issues that might arise. So, HTTP is left using a model that is less than ideal because of the amount of data overhead it creates. With a large number of devices operating in this manner, network latency and congestion can quickly become an issue. This problem is exacerbated by the fact that HTTP is a very verbose, text-based protocol that adds a significant amount of overhead in each request. For example, a simple request to read an on/off state takes 412 bytes of data for each request/response when using Exosite's simple HTTP data API.

Also, as a text-based protocol, HTTP is actually very hard for an embedded system to parse correctly. There are problems with encoding, because users must scan for special characters that define the divisions between certain parts of the message. It takes time and excess memory to re-encode the different parts of the messages, and there are also no defined maximums for the different components of the messages. This increases the complexity of implementations and can cause users of a given library headaches depending on how that particular library decides to deal with the problem.

Although these problems seemingly suggest HTTP is not an ideal protocol choice, they can sometimes be accommodated. And, because of external factors like the requirements of some networks or the availability of manufacturer-provided system libraries, HTTP may be a viable option in some cases.

3.3.1.2 XMPP

XMPP, previously known as Jabber, is a protocol originally designed for use in instant messaging. Early IoT developers were interested in XMPP because of its real-time nature. It provides low latency communication back to a single, central server. Exosite offers an XMPP-based API that was developed for use in applications where latency is the highest priority.

However, XMPP has a number of problems that make it somewhat undesirable for embedded IoT applications. As an XML-based protocol, XMPP is very verbose, even more so than HTTP, and has heavy data overhead. A single request/response exchange to send one byte of data from a device to the server is more than 0.5 kB.

There is a draft specification that would compress XMPP using an XML encoding called Efficient XML Interchange (EXI). But even with EXI, the same one byte of data gets hundreds of bytes of protocol overhead from XMPP alone. EXI is also a much harder format to process than other options now available. Because of these inherent problems, it is generally recommended to avoid using XMPP in embedded IoT applications.

3.3.2 THE NEW KIDS

With the rapid growth of IoT, new protocols have been created specifically to meet the needs of IoT systems and devices, including the Message Queue Telemetry Transport (MQTT) protocol and the Constrained Application Protocol (CoAP). These protocols offer the benefit of being designed to be efficient and powerful with the types of workloads found in IoT. However, they do fall behind in the areas of platform library support and general maturity of design when compared to the more established protocols.

3.3.2.1 MQTT

MQTT is a publish/subscribe messaging protocol designed to be very simple, lightweight, and easy to implement. The protocol was originally developed by IBM, although control was recently given to the OASIS consortium. The entire MQTT protocol specification is relatively short and written in a way that makes it easily understood. Someone relatively technical can read the whole protocol specification in a day or two and possibly even implement it in under a week.

However, this brevity of the specification can also be challenging. Some areas are too ambiguous and generally lacking in basic features that would be of substantial benefit in real-world deployments. One of the biggest pain points in MQTT is the absence of useful error-handling. Most error conditions are handled by simply disconnecting the TCP session without any indication about why it happens.

As a result, MQTT works well for small, quickly implemented, one-off deployments of a single device or implementation where the client and server are both controlled. It is not ideal for situations in which a heterogeneous set of clients require different protocol implementations to talk to a single service. Luckily there are better options for this.

3.3.2.2 COAP

CoAP is a new protocol that was recently finalized by the Internet Engineering Task Force in memo RFC 7252. CoAP was designed for use with resource-constrained embedded devices, both in terms of computation and connectivity, while remaining very extensible. It was also designed specifically to accommodate problems that are likely to be encountered in a global IoT device fleet deployment.

The semantics of CoAP were designed to closely model those of HTTP, so developers that are already experienced with HTTP can get up to speed more quickly, and applications developed using HTTP can be directly reapplied to applications using CoAP. However, unlike like HTTP, which is text-based and uses TCP, CoAP is a binary protocol that is transported over UDP. Being a binary protocol reduces its data overhead, while its use of UDP increases its flexibility in communication models and its ability to reduce latencies.

This means CoAP is not limited to just the semantics of HTTP. One of the benefits of using HTTP semantics on top of CoAP's UDP rather than HTTP's TCP is that a device can more easily use the same protocol code to talk to the cloud and other devices on the local network. It can even engage in group communication with IP multicast. This is a boon to applications where devices on the same local network are expected to work together, in addition to working through the cloud. A single protocol and, thus, a single protocol implementation can be used to do both styles of communication, reducing both development time and the resources required on the devices.

Additionally, the use of UDP allows for further optimization of an embedded device's power consumption, without adding latency. Since it is not necessary to keep a TCP connection established, a device can sleep until it actually has something to report and must only remain awake for one round trip's worth of latency with full reliability. For the most power-sensitive applications, devices can be woken up only long enough to send data, without waiting for a response to come back. Occasionally, a device may wait for the response as a "tracer round" to make sure some of the requests are making it to the server and ensure some level of reliability.

Finally, the extensibility of CoAP provides features like the ability to flexibly update the format of the data that a device uses to communicate, which can be critical to businesses that already have devices deployed. For this, CoAP has an option called "Accept" that allows at client to request a format for data that it is requesting from the server. The protocol even has a way to denote which options are safe to ignore. This allows new options to be added to the protocol in a way that existing devices will to continue to work, while adding features of which new devices can take advantage. Because IoT devices may last many years, during which time technologies and business requirements will undoubtedly change, CoAP's extensibility in this and similar ways can enable rock-solid, future-compatible IoT deployments.

With all these benefits of CoAP, it may seem like an ideal choice for all IoT communication needs. However, there are a few factors that must be considered. First, using UDP instead of TCP does have its downsides. UDP does not have the same guarantees that TCP supplies. To overcome this, CoAP takes on the features that are necessary for its specific needs, ignoring those that are not helpful with IoT-style communication.

Additionally, without TCP, standard Transport Layer Security (TLS) (previously known as Secure Sockets Layer (SSL)) cannot be used to secure communication. Datagram Transport Layer Security (DTLS), a newer derivative of TLS that has a few additional semantics added to allow it to work over UDP, must be used. Because of its relative lack of age, it has a limited amount of existing support. For instance, Exosite is not aware of any Wi-Fi hardware modules that have DTLS support built-in, so a software DTLS stack on a host system may be necessary for secure communication.

Also, similar issues exist for CoAP itself, as at the time this document was written, the CoAP specification had only been finalized for a little more than a year. As a result, fewer options are available for existing libraries and solution support as compared to some of the more traditional protocols.

3.3.2.3 WEBSOCKET

WebSocket might be a bit of an unsuspected addition to this list for some. WebSocket is not a protocol that was designed for use in IoT but instead, as the name suggests, for use with the web. It lets web browsers and web servers communicate continuously using a message-based, bi-directional channel.

The biggest benefit of using the WebSocket protocol is its network compatibility. A WebSocket connection is established first as an HTTP request, so if a network can support an HTTP request, it can almost certainly support WebSocket. The server-side library support is also a major benefit to the Websocket protocol, making implementation on a server much easier thanks to the wide deployments of existing WebSocket servers for the web. Like HTTP, WebSocket uses TCP and, thus, can use TLS and take advantage of its wider availability in existing network stacks.

The biggest downside to using WebSocket is the weight of the protocol and the hardware requirements that it brings with it. WebSocket requires a TCP implementation, which may or may not be a problem, but it also requires an HTTP implementation for the initial connection setup. Additionally, Websocket was not designed with the requirements of highly constrained embedded systems in mind, so implementations may not be straightforward. And, unfortunately, there are currently no useful, open source WebSocket implementations targeted at embedded systems.

Again, as it was with HTTP, these problems may not be impossible to overcome. The decision to use the WebSocket protocol will depend heavily on outside factors and those factors may make WebSockets an attractive option.

4. SUMMARY

Because the requirements of individual IoT implementations can vary significantly, it is impossible to suggest a single protocol that should be used in every situation. Instead, this document has provided a high-level overview of the features and benefits of several protocols that should be taken into consideration when selecting one for use in a connected solution.

To help meet the needs of any implementation, Exosite offers several flexible API options, including CoAP, HTTP, and WebSocket. The Exosite IoT platform provides a modular, customizable architecture that enable companies to quickly and easily deploy IoT devices and services that deliver the reliability, security, scalability, and flexibility they require.

EXOSITE

USA HEADQUARTERS
275 Market St, Suite 535
Minneapolis, MN 55405
+1.612.353.2161

TAIWAN OFFICE
WenXin Road, Section 4
#955, 15F-5
Taichung, 406 Taiwan
+886.4.2247.1623

REVOLUTIONIZE YOUR BUSINESS WITH IoT.

Engage with our team of world-renowned experts to learn more.

EXOSITE.COM | +1.612.353.2161