IIoT Protocols to Watch

Aron Semle, Sr Solutions Manager, PTC

IoT is alphabet soup. IIoT, IoE, HTTP, REST, JSON, MQTT, OPC UA, DDS, and the list goes on. Conceptually, we've discussed IoT for a long time and understand the basic idea and technical feasibility. Now we're moving forward, identifying use cases and building prototypes. So it's about time to work on that alphabet.

A big challenge in IoT is interoperability. In a recent Nexus survey, 77% of respondents stated that interoperability was their biggest challenge in IoT. Connecting industrial devices to IT and IoT platforms is big business, and it's where a lot of the abbreviations come from. There are many protocols to accomplish this: some that are proprietary and others that are open standards. All are jockeying to be the one and only IoT protocol, but it's clear this will never be the case. These protocols will co-exist—each with their own strengths and weaknesses—and it's our job to understand where and when to use them.

This white paper focuses on the open standards for connecting industry to IT and provides use cases for each.

Client/Server vs. Publish/Subscribe

For the purpose of this discussion, it's important to group protocols into two categories: client/server and publish/subscribe.

Client/server protocols require the client to connect to the server and make requests. In this model, the server holds the data and responds to requests from the client. For example, the client may read a temperature. This requires the client to know about the server in advance and be able to connect.

Publish/subscribe protocols require the devices to connect and publish data to a topic on an intermediary broker. Consumers can connect to the broker and subscribe to data from the topic. For example, a device can sample the temperature every minute and publish once an hour. An application subscribed to the data stream receives an hour's worth of one-minute samples every hour. This model decouples the producer of the data from the consumer of the data.

For example, you know that your server in the field has an IP address of 55.55.55.55 and is listening on port 1234. The client can connect and make requests.

Publish/subscribe protocols are a better fit when your infrastructure is unknown. For example, if a remote device changes networks or has intermittent connectivity, it's easier for the device to call home when it's online and publish its data.

In terms of pros and cons, client/server protocols are more interoperable and secure because they are based on point-to-point connections. They are, however, less scalable, because point-to-point connections are harder to manage and more resource intensive.

> Client/server protocols are best used when you understand your infrastructure… Publish/Subscribe protocols are a better fit when your infrastructure is unknown”

In contrast, publish/subscribe protocols are more scalable because the decoupling of producers and consumers allows each to be added and removed independently. That said, securing these protocols is more complex because there are more pieces involved. They can also have interoperability issues given the lose coupling of the producer and consumer. For example, changing the message format that a producer sends requires all consumers to adapt to the new message type.

Now that we understand the basic categories, let’s look at client/server and publish/subscribe protocols in more detail.

Protocols

The protocols we’ll discuss have the potential to connect together industrial devices with IoT platforms. It may go without saying, but if you’re trying to connect two applications and both support HTTP, try HTTP first. If that doesn’t work or if your environment doesn’t support it, keep reading. We’ll describe each protocol and when to use it. Here is the short list that we’ll cover:

- OPC UA
- HTTP (REST/JSON)
- MQTT
- CoAP
- DDS
- AMQP

OPC UA

OPC Unified Architecture (OPC UA) is the next generation standard from the OPC Foundation. Classic OPC is well known in the industrial space and provides a standard interface to communicate with PLCs. OPC UA aims to expand OPC’s interoperability to the device and enterprise levels.

OPC UA is a client/server protocol. Clients connect, browse, read, and write to industrial equipment. UA defines communications from the application to the transport layer, making it very interoperable between vendors. It’s also highly secure, and uses two-way message signing and transport encryption.

OPC UA has a wide install base in the industrial space. It is a good solution for tying PLC and sensor data into existing industrial applications like SCADA and MES systems, where OPC and OPC UA connectivity are already available.

> The focus on HTTP in IoT is around Representational State Transfer (REST), which is a stateless model where clients can access resources on the server via requests."

OPC UA is new to the IT space, however. Some people in IT are intimidated by the complexity of UA compared to other IT protocols. A lot of this complexity is a result of OPC UA being an industrial protocol, but this perception has led to slow adoption by IoT platforms and the open source community.

Things are changing, however: recently, the OPC Foundation open sourced the OPC UA standard to make it more accessible and help increase adoption.

For now, use OPC UA when you need to get PLC and sensor data into existing SCADA and MES solutions, and keep an eye out for OPC UA adoption by IoT platform providers and the open source community.

HTTP (REST/JSON)

Hypertext Transfer Protocol (HTTP) is a connectionless client/server protocol ubiquitous in IT and the web. Because there are countless open source tools that use HTTP, and every coding language has HTTP libraries, it is very accessible.

The focus on HTTP in IoT is around Representational State Transfer (REST), which is a stateless model where clients can access resources on the server via requests. In most cases, a resource is a device and the data that a device contains.

HTTP provides a transport, but doesn't define the presentation of the data. As such, HTTP requests can contain HTML, JavaScript, JavaScript Object Notation (JSON), XML, and so forth. In most cases, IoT is standardizing around JSON over HTTP. JSON is similar to XML—without all the overhead and schema validation—making it more lightweight and flexible. JSON is also supported by most tools and programming languages.

Industry has some experience using HTTP for device and product configuration, but not for data access. As such, many IoT and IT platforms support consuming and providing data in HTTP form, but few industrial platforms do. This is changing as more gateways and PLCs begin to add native HTTP support.

> Many IoT platforms support HTTP and MQTT as their first two inbound protocols for data."

Use HTTP for sending chunks of data, like one-minute temperature readings every hour. Don't use HTTP for streaming high-velocity data. HTTP can do sub-second data, but 100 ms updates over HTTP are difficult. It has a lot of overhead per message, so streaming small messages is inefficient. And always secure communications with HTTPS. The overhead is minimal.

Be aware of interoperability issues with HTTP products. Just because two products support HTTP/REST/JSON doesn't mean they'll work out of the box. Often the JSON formats are different and require minimal integration to get things working.

MQTT

Message Queuing Telemetry Transport (MQTT) is a publish/subscribe protocol designed for SCADA and remote networks. It focuses on minimal overhead (2 byte header) and reliable communications. It's also very simple. Like HTTP, MQTT's payload is application specific, and most implementations use a custom JSON or binary format.

MQTT isn't as widely used as HTTP, but it still has a large market share in IT. There are many open source clients/producers, brokers, projects, and examples in every language.

Use MQTT when bandwidth is at a premium and you don't know your infrastructure. Make sure you or your vendor has an MQTT broker you can publish data to—and always secure communication via Transport Layer Security (TLS).

Does the end application not support MQTT? If so, there are a lot of open source tools for getting MQTT data into databases and other formats like HTTP.

Beware of interoperability issues similar to HTTP. Just because two applications support MQTT doesn't mean they are interoperable. The topic and JSON formats may need to be adjusted to make the two products interoperable.

> DDS is a good solution for reliable, real-time data delivery at the edge. Use it for fast M2M communications."

CoAP

The Constrained Application Protocol (CoAP) was created by the Internet Engineering Task Force (IETF) to provide the interoperability of HTTP with minimal overhead. CoAP is similar to HTTP, but uses UDP/multicast instead of TCP. It also simplifies the HTTP header and reduces the size of each request. CoAP is used in edge-based devices where HTTP would be too resource intensive, and is often the third protocol supported by IoT platforms after HTTP and MQTT. Similar to HTTPS, CoAP uses Datagram Transport Layer Security (DTLS) to secure communications.

Use CoAP when HTTP is too bandwidth intensive. Keep in mind that CoAP's market adoption is not as large as HTTP, so it may limit your software and hardware options. There are solutions for converting CoAP messages to and from HTTP that make CoAP solutions more interoperable.

DDS

Data Distribution Service (DDS) is a publish/subscribe protocol that's focused on communication at the edge of the network. DDS is an open standard managed by the Object Management Group (OMG). Unlike MQTT which requires a centralized broker, DDS is decentralized. DDS nodes communicate directly in peer-to-peer fashion using UDP multicast. This removes the need for centralized network management and also makes DDS a faster protocol, reaching sub-millisecond resolution.

DDS is a good solution for reliable, real-time data delivery at the edge. Use it for fast M2M communications.

DDS supports brokers to integrate DDS networks with the enterprise, but in practice it is not well positioned as the integration point between industry and IT as brokers are often secondary to the DDS network.

> It's important to pick the protocol that best fits your needs, and select technology partners that can adapt to these protocols."

AMQP

Advanced Message Queuing Protocol (AMQP) is another publish/subscribe protocol that comes out of the financial services sector. It has a presence in IT, but a limited presence in industry.

AMQP's biggest benefit is its robust communications model that supports transactions. Unlike MQTT, AMQP can guarantee transactions complete—which, though useful, is not always required by IoT applications.

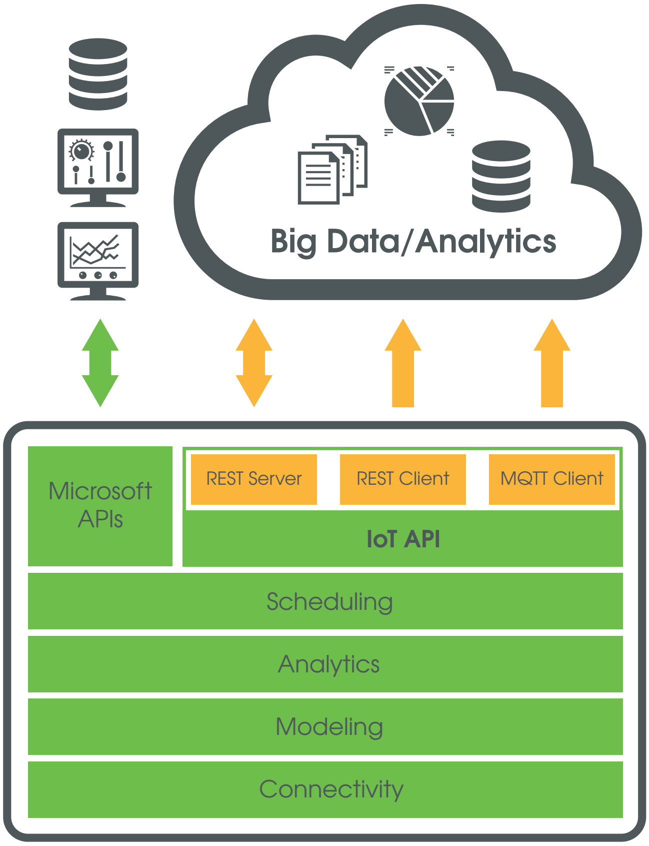
AMQP often gets grouped with IoT protocols and it is one—but its biggest con is that it's a heavy protocol. It was meant for backend IT systems, and not the edge of the network.

Conclusion

OPC UA, HTTP, MQTT, CoAP, DDS, and AMQP all have a place in IoT. Which protocols take majority market share is unclear, but each has its pros and cons.

This will ensure the success of your IoT applications and protect you from the protocol wars.

Be sure to check out Kepware's new IoT Gateway available in the KEPServerEX version 5.19 release. We included support for REST and MQTT, allowing our customers to get PLC data into new IoT platforms and open source tools like Node-RED. Unlock your industrial data to the Internet of Things.



Kepware Technologies is a software development business of PTC Inc., headquartered in Portland, Maine. Kepware provides a portfolio of software solutions to help businesses connect diverse automation devices and software applications and enable the Industrial Internet of Things. From plant floor to wellsite to windfarm, Kepware serves a wide range of customers in a variety of vertical markets including Manufacturing, Oil & Gas, Building Automation, Power & Utilities, and more. Established in 1995 and now distributed in more than 100 countries, Kepware's software solutions help thousands of businesses improve operations and decision making.

J7846–IIoTProtocolstoWatch–EN–1016
First edition published 2015. Second edition published 2016.