
# A Practical Guide to Using the Industrial Internet Connectivity Framework

**Authors:**
**Stan Schneider, PhD.,** CEO
Real-Time Innovations, Inc.
stan@rti.com

**Rajive Joshi, PhD.,** Principal Solution Architect
Real-Time Innovations, Inc.
rajive@rti.com

## INTRODUCTION

The Industrial Internet of Things (IIoT) is a vast, expanding space. The Industrial Internet Consortium's (IIC) primary mission is to resolve the resulting confusion and thus guide the industry. The IIC developed the Industrial Internet Connectivity Framework (IICF) to help designers understand the many standards and choose the right one for their applications.

The IICF focuses on the layers above the network packet exchange. It offers profound insights into architecture, standards, and use cases. It also presents a way to build an expansive IIoT network in the future by linking a few "core connectivity standards" that address different regions of the connectivity space.

This paper first outlines the IICF's key insights into system architecture, including the newly-defined IIoT connectivity stack. With that architecture, we explain why the IICF sorts the various standards into the stack based on their provided interoperability. Finally, the IICF offers deep analysis of six key standards and technologies: DDS[1], OPC UA[2], oneM2M[3], RESTful HTTP[4], MQTT[5], and CoAP[6]. There are of course many others; the IIC chose these because they have the greatest traction.

The most surprising conclusion: the IIoT space is so big that the connectivity technologies essentially do not overlap. Thus, by understanding the use cases, architectures and target end users, it is possible to select a best-candidate connectivity standard for most problems.

Finally, the paper then distills the IICF results into a simple tool that designers can use to select the right technology. The tool is a simple set of yes-or-no questions that assess the fit for each technology, as well as technical and business justification for those questions.

## THE IIOT SPACE AND THE IICF

The Internet of Things (IoT) combines many technologies, encompasses vast use cases, and attracts vendors of all sorts who want to join the fray. This generates confusion, nowhere more apparent than in its most important aspect: connectivity.

[1] The Data Distribution Service (DDS) is a series of standards managed by the Object Management Group (OMG), https://portals.omg.org/dds/

[2] OPC Unified Architecture (OPC UA) is a standard managed by the OPC Foundation, https://opcfoundation.org

[3] oneM2M is a standard managed by the oneM2M consortium, http://www.onem2m.org/. OPC UA is also known as IEC 62541

[4] The Hypertext Transfer Protocol (HTTP) is a standard, RFC 7231, managed by the Internet Engineering Task Force (IETF). https://tools.ietf.org/html/rfc7231

[5] MQ Telemetry Transport (MQTT) is a standard managed by the Organization for the Advancement of Structured Information Standards (OASIS). http://mqtt.org/

[6] The Constrained Application Protocol (CoAP) is a standard, RFC 7252, managed by IETF. https://tools.ietf.org/html/rfc7252

### Connectivity Technologies Do Not Overlap

The IICF includes the deep insights of many experts, including those from the top industry consortia, many companies and most of the important standards. Its most surprising conclusion: the IIoT is so big that the technologies don't really overlap. The impression that there is overlap is mostly just confusion.

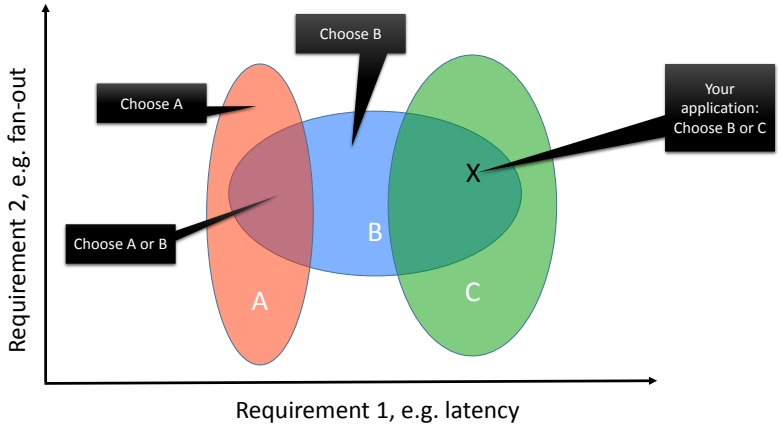


***Figure 1: Misconception.*** *Before the IICF, people (including the authors) assumed that competing standards meet overlapping requirements in the IIoT connectivity space. This designer's application (X marks the spot), fits B or C; either should work.*

Designers may think they can choose any standard and succeed. But this implies the IIoT connectivity solution space overlaps, as in the *Misconception* figure.

The reality is very different. The IIoT covers many industries with very different use cases. The range is breathtaking. There are thousands of companies and uncountable thousands of applications in the IIoT space. The IIoT really is the technological future of the entire world.

The connectivity technologies and standards that target these applications are very different. In fact, the IIoT space is so big that the technology options barely overlap. Today's architecture challenge in the IIoT space is therefore not one of choosing among overlapping standards that may each be able to reasonably solve a problem. The challenge is understanding the technologies, comparing the intended use to the application and choosing the *one* that best addresses the particular challenge. Sure, stretching a technology all out of proportion may make anything work. But, that will result in a lot of extra work and an awkward design. If you look at a more realistic map of the situation, it looks more like the sparse Venn diagram in the *Reality* figure (Figure 2)

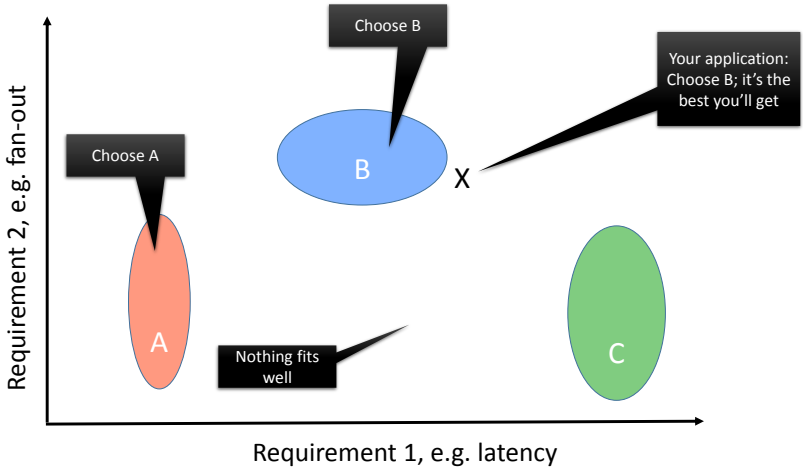


***Figure 2: Reality.*** *The connectivity standards turn out to not overlap. Most applications will not be a perfect fit, and must adapt. Your challenge*

than the overlapping one in the *Misconception* figure (Figure 1).

This may sound distressing, but in reality, the lack of overlap in the IIoT space actually makes an architect's task much simpler. The real problem is not choosing between similar options; it is understanding the different options and overcoming biases.

**The Industrial Connectivity Stack**

The Internet has well-developed layered stack models, most notably the OSI "7-layer" model and the Internet "4-layer" model. However, the IIC experts found that these models did not adequately capture the requirements of *industrial* networks. Thus, the IICF defines a new model, called the IIoT Connectivity Stack Model (See Figure 3).

different challenges than enterprise networks do. Most importantly, industrial systems combine complex, intimately interconnected software modules and devices. Interoperability between the various components is the most demanding requirement on the architecture. With crisp definitions of interoperability, the other requirements can be much better understood. The IIoT Connectivity stack shares layers 1-3 with the OSI model and defines levels 4 through 6 in terms of the interoperability provided.

The layers focus on what is exchanged. At the *network layer*, participants exchange bounded-length packets of information. This is usually implemented by the familiar Internet Protocol (IP).

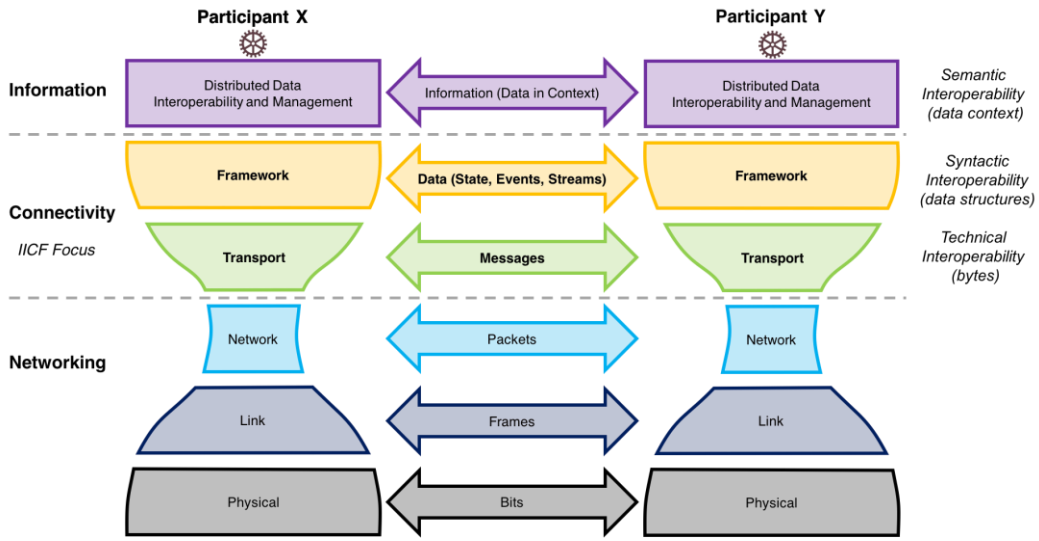


***Figure 3: IIoT Connectivity Stack.*** *Below layer 3 "Networking", the IIoT stack duplicates traditional Internet stacks. Above, it focuses on clarifying the interoperability that industrial IoT systems need.*

Both the Enterprise Internet and the Industrial Internet function by sharing data. However, industrial systems present

Above it, the *transport layer* exchanges variable-length *messages*. Participants share opaque sequences of bytes. Participants using

this layer can share information, but interpretation of that information is completely up to the applications. This enables basic communications, but makes it difficult to interoperate between devices and software that is not designed together.

The next layer, called the *framework layer,* adds structure to the data exchange. This allows components to understand how to process the messages, also called "syntactic" interoperability. Participants above this level can use different programming languages, operating systems, and processor architectures transparently. The framework layer also enables configurable quality-of-services (QoS) like reliability, durability, filtering and more. QoS enables control over data delivery, including selecting information and endpoint delivery conditions. Together, these functions enable a "data model" for the system. The data model is the basis for diverse components to work together. Sophisticated implementations can even match some differences in data model, thus allowing a large distributed system to grow incrementally from parts that are not all developed or deployed together. Thus, syntactic interoperability is critical functionality for an industrial Internet.

Once participants can exchange known structures, they must also know how to interpret the information, aka "semantic interoperability". This is the responsibility of the *distributed data interoperability and management* layer. In the current state-of-the-art, semantic definition is only practical within an industry. There are many standards that operate at this level, including the ICE (Integrated Clinical Environment) in the medical industry, OpenFMB (Open Field Message Bus) in the power industry, and others. Semantic interoperability is beyond the scope of the IICF and this paper.

**The Core Connectivity Standard Architecture**

The IIoT space is far too big to expect a single connectivity standard to span everything. Thus, to build an Internet, we will eventually need to connect subsystems based on different standards.

The IICF does this with the concept of a "Core Connectivity Standard" (CCS). The CCS design eliminates the "N-squared" problem by choosing a few standards that together span the space and separately provide key functionality. The design simply defines the few standard bridges (called "core gateways") between core standards. Other connectivity technologies can then interface to the system through any one CCS. This enables practical end-to-end data exchange, as shown in the Core Connectivity Architecture figure.

This design enables a scalable, deeply connected future Industrial *Internet* of Things. Of course, it does introduce the question of what qualifies a standard to be a core standard. The IIC reasonably requires that a connectivity core standard shall:

- Provide syntactic interoperability,
- Be an open standard with strong independent, international governance and with support for certifying or validating or testing interoperability of implementations,
- Be horizontal and neutral in its applicability across industries,
- Be stable and deployed across multiple vertical industries,

- Have standards-defined Core Gateways to all other connectivity core standards.

With that definition, the IIC experts proceeded to define criteria, survey standards and evaluate all the standards against those criteria. The criteria definitions alone are a huge contribution, beyond the scope of this paper. They resulted in an "Assessment Template" with deep analyses of the six standards with greatest IIoT traction: DDS, OPC UA, oneM2M, RESTful HTTP, MQTT, and CoAP. The assessment includes business, usage, functional and implementation viewpoints. It is a unique analysis of connectivity technologies for industrial systems. In the end, the IICF identified four of the six as potential core standards: DDS, OPC UA, oneM2M, and RESTful HTTP.

Of course, this design targets a future world of vastly connected systems. There is a more immediate question: "if I'm starting an IIoT project, what should I use?" Next, we present the IICF guidance and a simple tool to navigate that guidance.

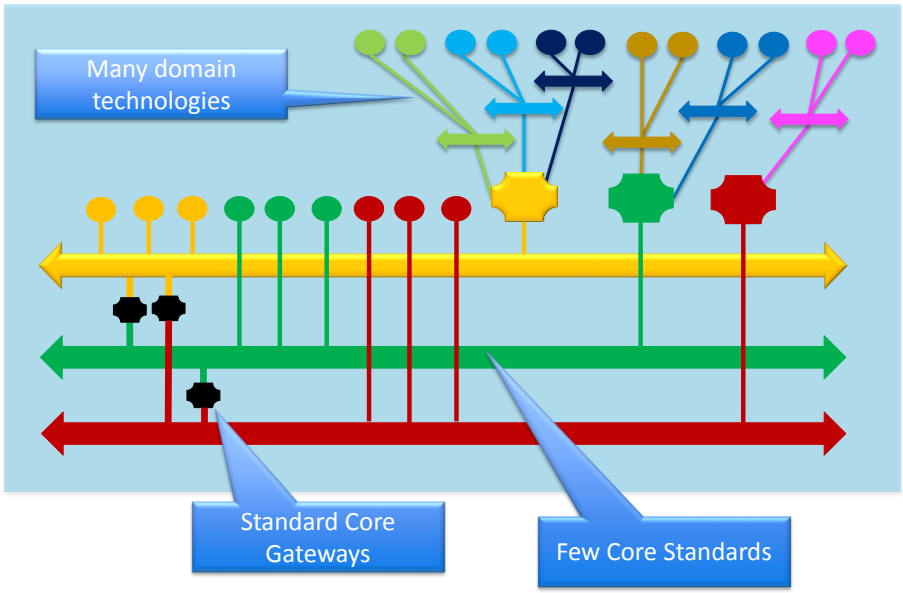


***Figure 4: Core Connectivity Architecture***. *Each core connectivity standard requires a standardized core gateway to connect to the other core architectures. Other domain technologies can then interface to the system via any core standard. This design scales linearly with the number of technologies, thus enabling a true Internet.*

## CHOOSING A CONNECTIVITY STANDARD

In one key table, the IICF outlines the example use cases and application spaces of the various core connectivity options. The intention, and the effect, is to provide useful guidance to architects struggling with the confusing arrays of vendor and advocate claims. The table is reproduced below in the *IIC Use Case Examples* figure.

Let's take this process a bit further. It is possible to ask a few very simple questions for each technology option and quickly narrow the choices. These questions may oversimplify the problem, but they are a great starting point. The IICF identifies four potential "core connectivity standards:"

| System Aspect | Example User | Approach | Targeting Standard |
|---|---|---|---|
| Software Integration and Autonomy | You are a software architect. You are building a system or product line, and you control the architecture. You critically need to integrate components written by different programmers or even entire teams. | A data-centric approach will define the interfaces, capture the dataflow, enable module evolution, and enforce interoperation between teams. This approach also eases redundancy, fast complex data flow, and selective data filtering. | DDS |
| Device Interchangeability | You are a device manufacturer, with the goal of making devices that will sell into many applications. The device offers services, such as configure, start, stop, etc. You have no idea how the device will eventually be used. Your users are likely not software experts; they just want to add or integrate the device into a work cell. | A device-centric approach will allow the device users to write generic software that will interoperate with competitor's devices. | OPC UA |
| Web and Mobile User Interfaces | You are building mobile apps or web browser based applications to provide the human machine interface. You need an easy way to support clean human interaction and access to backend services. | A RESTful approach will make it easy to connect to many types of enterprise systems and UI devices. | Web Services (HTTP/REST) |
| Information & Communication Technology (ICT) Integration | You are building a wide-area wireless system that needs to allow applications and devices to share data and information. The devices use various technology and domain-specific protocols. The applications and devices you integrate rely on leveraging the services provided by the communications provider network. | A common, standard services-layer approach enables applications and device to share data and information without forcing the application to understand multiple protocols implemented on the devices. The applications can thus run in the Platform Tier and seamlessly connect to diverse IoT devices in the field. | oneM2M |

**Table 8-2 Non-overlapping system aspect examples addressed by the potential IIoT connectivity core standards**

*IIC Use Case Examples. The IIoT space is so big that the primary users of the different technologies don't speak the same language. In practice, there are few applications that could reasonably make alternative choices. The real problem in the industry is the lack of understanding of the options. The IIC experts sorted the technologies into very clearly different "boxes". The choice is nearly as simple as identifying which box most sounds like you.*

DDS, OPC UA, oneM2M, and RESTful HTTP. We analyze these in this section. MQTT does not qualify as an IICF "core connectivity standard" because it does not have a standard typing system required for syntactic interoperability. We examine it nonetheless because it has wide awareness.

**DDS**

Here are five questions to answer to decide if you need DDS:

1. Is it a big problem if your system goes down for a short time?
2. Are milliseconds important in your communications?
3. Do you have more than 10 software engineers?
4. Are you sending data to many places, as opposed to just one (like to the cloud or a database)?
5. Are you implementing a new IIoT architecture?

If you answered three out of the five questions "yes," you probably should use DDS.

DDS is a series of standards managed by the OMG that define a *databus*. A databus is data-centric information flow control. It's a similar concept to a database, which is data-centric information storage. The key difference: a *database* searches *old* information by relating properties of stored data. A *databus* finds *future* information by filtering properties of the incoming data. Both understand the data contents and let applications act directly on and through the data rather than with each other. Applications using a database or a databus do not have a direct relationship with peer applications.

The databus uses knowledge of the structure, contents and demands on data to manage dataflow. It can, for instance, resolve redundancy to support multiple sources, sinks and networks. The databus can control Quality of Service (QoS) like update rate, reliability and guaranteed notification of data liveliness. It can look at the data inside the updates and optimize how to send them, or decide not to send them at all. It also can discover and secure data flows dynamically. All of these things define interaction between software modules. The data-centric paradigm thus enables software integration.

So how does this satisfy the five questions?

1. Since it is directly controlling flow, a databus does not require servers. So, there's no single point of failure. The downtime required to reboot a server and remake connections unexpectedly is never necessary. Without direct relationships with peers, redundancy is transparent. If the application is managing a thermostat, optimizing a plant, or assembling parts, a short downtime is not catastrophic. However, if the software is responsible for someone's breathing or the stability of the power grid, even short interruptions cannot be tolerated.
2. Since the databus has full control over how data flows, it can send information directly between peers. Thus, it can deliver in times measured in milliseconds or microseconds. DDS can use multicast intelligently when available. It knows

delivery deadline requirements and can measure if the system is meeting delivery times. So, it can warn applications if the network (or anything else) cannot handle the needed flow rates.

3. Teams of programmers must control interfaces between modules. The databus specifies a full data model. All connectivity frameworks do this to some extent, but the databus specification is more expressive. It includes not only type information, but also QoS such as deadlines, sensor availability and flow rates. So, the interfaces are defined and then enforced at runtime. The databus can also manage the evolution of those interfaces, allowing modules, for instance, that use newer and older versions of an interface to interoperate. That is important for a practical, large IIoT system that must be incrementally deployed and updated.
4. A databus controls flow between many complex applications. It handles a mix of fast and slow components. Its filtering can make the overall flow manageable. Peer discovery delivers data between multiple field-based components. And QoS control guarantees the flows. There are simpler solutions for one-way, one-destination flows such as capturing sensor information to send it to the cloud for analysis.
5. Finally, using a databus requires a completely new architecture. Most DDS designs are building something new rather than optimizing something old. The databus can integrate legacy subsystems via gateways and adapters, but it should not be considered an incremental design change.

Most databus systems do not have all five of these properties. Three of the five indicate that a databus design will be compelling.

**OPC UA**

OPC UA is a standard managed by the OPC Foundation, also documented as IEC 62541.

OPC UA targets device interoperability. Rather than accessing devices directly through proprietary application program interfaces (APIs), OPC UA defines standard APIs that allow changing device types or vendors. This also lets higher-level applications such as human-machine interfaces (HMI) find, connect to and control the various devices in factories.

OPC UA divides system software into clients and servers. The servers usually reside on a device or higher-level Programmable Logic Controller (PLC). They provide a way to access the device through a standard "device model." There are standard device models for dozens of types of devices from sensors to feedback controllers. Each manufacturer is responsible for providing the server that maps the generic device model to its particular device. The servers expose a standardized object-oriented, remotely-callable API that implements the device model.

Clients can connect to a device and call functions using the generic device model. Thus, client software is independent of the actual device internals and factory integrators are free to switch manufacturers or models as needed. So, OPC UA can build and maintain a system from interchangeable

parts, much like standardized printer drivers allow PC system integration. Note that the device model also provides a level of "semantic" interoperability, because the device model defines the generic object APIs in known units and specified reference points.

Determine if you should use OPC UA by answering the following questions:

1. Are you in discrete manufacturing?
2. Must you connect to an Industrie 4.0 system?
3. Are you building a device that will be integrated by control or process engineers or technicians, rather than software engineers?
4. Will your product be used in different applications in different systems, as opposed to one (type of) system where you control the architecture?
5. Are you building equipment for a "workcell" or "skid?"

Three "yes" answers to these five questions point strongly to OPC UA. Let's look at how the technology fits these use case indicators:

1. OPC UA is well-positioned for discrete manufacturing. These applications are characterized by integrations of devices into sets of tightly coordinated subsystems. Since users want to avoid vendor lock-in, a ready supply of devices with interchangeable device models is important.
2. The German initiative Industrie 4.0 recommends OPC UA. Industrie 4.0 is focused on manufacturing, in contrast to the IIC that works on IIoT technical system architectures across verticals. In a sound bite, Industrie 4.0 is about *making things,* while the IIC is about *making things work*. They intersect only in manufacturing systems.
3. OPC UA provides more than connectivity; it also has pre-defined device models and a device integration architecture. Those using and choosing it usually target users who are plant or process engineers, rather than programmers. System integration in manufacturing is usually done between devices, not software modules. OPC UA has very helpful device models that aid interoperability between device manufacturers.
4. OPC UA has a system discovery mechanism called an "address space." This can be rolled up to a site-wide server, for instance and the system connected to a site HMI. This dynamic system building is very useful for providing similar functionality, such as historian storage or HMI viewing, to very different applications. It is appropriate where your users control the system design, not you.
5. Most OPC UA systems end up in workcells or on process skids. These are a stand-alone subsystems, usually incorporating 20 devices or so. OPC UA, and especially the industrial-integration software that supports it, targets workcell integration. The address model and object-oriented nature directly support a hierarchy of these workcells. Users of the other standards rarely characterize their use cases as "workcells."

**OneM2M**

OneM2M provides a common service layer that sits between applications and connectivity transport. Its emphasis is on providing common services, on top of different connectivity standards.

To determine if you should use oneM2M, consider these questions:

1. Do you know what "ICT" stands for and does it describe what you do?
2. Is the cellular network your primary connection technology?
3. Are your target applications largely composed of moving parts?
4. Can the components of the system tolerate intermittent connections and loosely-controlled latencies?
5. Will the system leverage services provided by a communications provider such as a telco?

These questions differ in character from the questions about the previous technologies. OneM2M results from cooperation among many mobile wireless providers. It targets networks of mobile devices that communicate mostly or only through the base-station infrastructure.

The following points examine why oneM2M is implied by these questions:

1. Surprisingly, most target users of DDS and OPC UA cannot even correctly define the name of oneM2M's target industry as Information and Communication Technology (ICT). There are, of course, exceptions. But, if you consider yourself in the ICT industry, then you need to consider OneM2M: It was designed for you.
2. The core design of oneM2M is to define services that mobile devices can use to cooperate and integrate. If you are going to use those services, you need to connect to them. They will be running in the platform layer (cloud) connected mostly through the cellular data infrastructure. Other technologies also use IP traffic over the cell network, but they usually also heavily leverage LAN, local wireless or WAN networking technologies in their designs.
3. There is a potential future market for 5G wireless integration of fixed assets such as manufacturing cells. However, this technology is still years away. oneM2M performs best for mobile assets. One especially powerful aspect is that oneM2M abstracts differences in protocols to those devices. Thus, it can integrate different ways to connect to similar devices.
4. Cellular mobile systems are not reliably connected. Thus, applications must not fail when communications are offline for a few seconds or minutes.
5. oneM2M system designers assume a cloud backend in their designs. The core of oneM2M is the standard services layer that is provided by a telco or its partners.

**RESTful HTTP**

REST (Representational State Transfer) over HTTP is the most common interface between consumer applications and web services. REST is an architectural pattern for accessing and modifying an object or resource. One server usually controls the object; others request a "representation" and may then send requests to create, modify or delete the object.

To see if RESTful HTTP is the best candidate for your application, ask these questions:

1. Are you connecting independent devices to a single web service API?
2. Are you building an HMI interface to an IoT device or service?
3. Does your application only need to be fast enough for human interaction?
4. Must your dataflow cross firewalls that you do not control?
5. Is there no device-to-device communication?

Three "yes" answers indicate you will likely be best off with RESTful HTTP. The reasons:

1. RESTful HTTP fundamentally makes it easy to connect a field device to a web service. REST is the most widespread way to build web services, enabling copious offerings to help developers. For instance, the DreamFactory open source project automatically creates APIs from most any database, thus enabling a centralized data-centric approach.
   While most applications use hypertext to present a linked view of a web page to a user, this paradigm is also similar for many IIoT "monitoring" applications. These applications are similar to smart-phone apps, except there is no human user. Instead of a phone, the end entity is a "thing," typically a single device. From a connectivity perspective, the things usually have only a single connection to an IoT platform. This category includes most of the "consumer" IoT, including thermostats, wearables and smart home locks.
   The most important industrial applications are "predictive maintenance" systems that upload device data to the cloud. Cloud systems then analyze the data to predict when a part may fail, allowing proactive repair.
2. HTTP is the most common way to serve information to human users. While most think of HTTP servers as living in the cloud with clients on the edge, this often is not the case in IoT. The server often, in fact, runs on the device itself, providing an intuitive configuration and management interface. Browsers can simply connect directly to the device, giving full access to advanced visual capabilities. This is limited to local connections, so another common configuration is to combine this with the above by having both the device and the user connect to a cloud-based HTTP service. This allows easy remote access to the device.
3. RESTful HTTP is an approach, not a standard with an official type system. However, most web systems use text encodings like JSON or XML. Text encoding is not fast so response speeds may satisfy humans, but not fast machines. Binary systems like Google Protocol Buffers (Protobuf) and the new HTTP/2 standard offer more efficiency, but are not yet widely used. HTTP runs over TCP, which does not deliver low latency.
4. Because all websites offer HTTP services, most IT firewalls allow connections to HTTP's default port (80). Deep packet inspection systems will likely accept HTTP. Thus, using HTTP on port 80 is usually the easiest way to traverse firewalls without special configuration.

5. RESTful HTTP requires that all devices connect to a server. While multiple devices may interact through that server or share a database, applications that share data between devices are rare.

**MQTT**

MQTT is a very simple protocol designed mostly for the "data collection" use case. It does not qualify as a "core connectivity standard" per the IICF guidelines, because it has no standard type system. Without a type system, it cannot offer a standard ability to interoperate at the "syntactic" data-structure level, leaving all data interpretation to the application.

Nonetheless, MQTT enjoys significant awareness. Because of its simplicity, simple questions about your system will help determine if it is appropriate:

1. Do you think of your application as data collection?
2. Is there little device-to-device communications?
3. Is interoperability not a consideration?
4. Do you have many small devices?
5. Is software a minor challenge?

Again, if you answer three or more of these "yes," you should look at MQTT. The reasons:

1. The first "T" in MQTT stands for "Telemetry," or data collection at a distance. This is its main use case.
2. MQTT is designed as a hub-and-spoke design that requires a broker. It does not support direct inter-device communications. Choosing MQTT for device-to-device communications is awkward.
3. Without a type system, MQTT applications are tightly coupled – they must all be aware of the data format. The only way to build interoperability is to implement a way to share types in user code. As a result, most of all MQTT applications are standalone systems. MQTT also does not help with system evolution; version compatibility between components must be provided by the applications.
4. MQTT is by far the simplest technology considered here. If you have many small devices that are simply connected, simple software can handle your challenge.
5. MQTT offers little to ease software development. There is only one QoS setting (reliability). There are no defined services. It offers no data or device modeling. You are therefore going to have to write all of the software from scratch; that is only practical if you have a simple software challenge.

**Comparisons and Overlaps**

Comparing these technologies highlights the stark differences and non-overlapping nature of connectivity approaches.

For instance, OPC UA is object oriented (OO), while DDS is data centric. Those are diametric opposites. The OO mantra is "encapsulate data, expose methods." Data centricity is all about exposing data and there are no user-defined methods. The only methods are defined by the standard.

OO systems work like a sequential programming language. You "call a method" on a remote server, it returns. Then you call the next. It is simple and intuitive for getting

and setting values. However, it is difficult to call many methods in parallel. The new OPC UA "pub/sub" functionality retains this flavor. Applications fundamentally interact with active remote peers. The paradigm is sequential reading and writing device values or streams, organized as a larger "address space."

Software systems built with OPC UA are typically compositions of existing modules like historians and HMIs. Direct OPC UA users are mostly device vendors. End OPC UA users are typically control or process engineers building and configuring systems of devices and existing software modules. OPC UA does not offer software teams integrating custom software help with complex interfaces that need a common system data model.

DDS, on the other hand, directly supports large custom software integrations. It explicitly requires a system data model and then uses that to automatically enforce interfaces. It works well for building and integrating AI modules, custom software development and wide data distribution. Everything is redundant and massively parallel. Most DDS end users are teams of programmers with dozens or even thousands of developers. DDS frustrates non-programmers who want to quickly integrate devices without much new software.

Thus, DDS and OPC UA target vastly different users. For OPC UA, end-user teams with more than a few programmers are rare. DDS is the opposite; most end-user teams have many programmers. That results in vastly different market penetration, depending on the integration challenge. For example, most OPC UA applications are in discrete manufacturing, while DDS has essentially none. That's because discrete manufacturing systems today are built via device, rather than software, integration.

Similarly, MQTT applications mostly target data collection from devices to a central store or analysis function. This is a rare application for either OPC UA or DDS, which work between devices. Also, oneM2M works by offering common services aimed at integrating mobile devices. None of the other technologies target this application.

## THE FUTURE

Combinations of the core standards will make great sense in the future. For instance, future complex software systems can use DDS, but access interchangeable OPC UA devices through a gateway. That design is powerful.

The IICF specifies an architecture for sharing data across connectivity technologies to allow this future, pervasive Industrial Internet. IIoT use cases are evolving from simple monitoring to optimization and finally to autonomy. These increasingly require more powerful integration.

Today's IIoT designs are relatively isolated within industries. Someday, there will also be cross-industry integrations, such as manufacturing systems integrated with transportation and power. More importantly, sophisticated autonomy software will reconfigure workcells, creating a bold new world for component device vendors. Wireless 5G systems will

interoperate with freeway controllers and autonomous vehicles. 5G may even directly control factory devices, eliminating wiring in manufacturing.

However, designers should consider the vastness of the space. Today, there are few concrete needs to bridge the huge gaps between connectivity systems. That doesn't mean the industry is not responding to the obvious need. Gateway standard efforts are active between all the core connectivity standards. A recent demonstration at an IIC testbed shows a bridge between DDS and OPC UA. Long term, these integrations will allow bigger systems combining technologies. For now, designers must understand the vast differences between connectivity standards and choose the one that best fits their problem space.

➢ Return to IIC Journal of Innovation landing page for more articles and past editions.

The views expressed in the *IIC Journal of Innovation* are the contributing authors' views and do not necessarily represent the views of their respective employers nor those of the Industrial Internet Consortium.

© 2017 The Industrial Internet Consortium logo is a registered trademark of Object Management Group®. Other logos, products and company names referenced in this publication are property of their respective companies.