The Industrial Internet of Things Connectivity Framework

An Industry IoT Consortium
Foundational Document

2022-06-08

Authors

Rajive Joshi (Lead, RTI), Paul Didier (Cisco), Christer Holmberg (Ericsson), Jaime Jimenez (Ericsson), Timothy Carey (Nokia).

CONTENTS

Figures

Tables

1 INTRODUCTION

Connectivity is the ability of a system or application to communicate with other systems or applications via computer network(s).[1] It is required for communication to take place with others. Ubiquitous connectivity is one of the foundational technologies enabling data and information sharing amongst participating components of an Industrial Internet of Things (IIoT) system.

Communication is the exchange of data and information amongst participants within a functional domain, across functional domains within a system and across systems. The data exchanged may include sensor updates, events, alarms, status changes, commands, configuration updates and any other piece of information needed for the system to operate.

Connectivity enables participants to communicate. Communication is a cross-cutting function across the functional domains defined by the Industrial Internet Reference Architecture,[2] as shown in Figure 1-1.

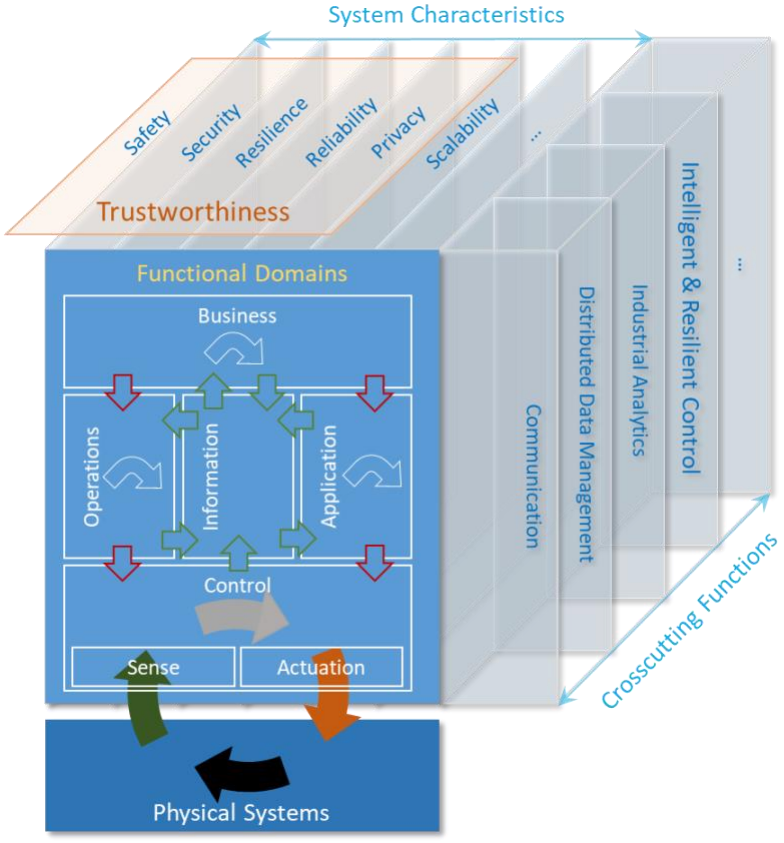


Figure 1-1: Communication is a cross-cutting function in the Industrial Internet Reference Architecture. Connectivity is the ability to communicate. Connectivity implements communication.

[1] See [IIC-IIV]
[2] See [IIC-IIRA]

The communication cross-cutting function enables exchange of information between endpoints in an IIoT system and the physical systems of devices and controllers attached to Operational Technology (OT). In summary, the exchange of information includes: the transfer of data over networks, message exchanges between endpoints, and data encoding and meaning.

1.1 PURPOSE

The IIoT landscape is replete with proprietary and industry-specific communication and connectivity technologies and standards optimized for a narrow set of domain-specific use cases in vertically integrated systems. These domain-specific technologies, though optimal in their respective domains, can be a hindrance to interoperability within and between IIoT systems. The overarching goal of IIoT communication is to unlock data and information in these isolated systems ("silos") and enable data sharing and interoperability between previously closed components and subsystems (brownfield) and new applications (greenfield), within and across industries. The sharing of data, designs, architectures, through proper capabilities is essential to creating new value streams and unleash the potential of a global IIoT marketplace.

This document provides an overall definition of the IIoT communications stack, maps the rich landscape of IIoT connectivity technologies, defines an open connectivity reference architecture, and helps practitioners navigate their way to categorize, evaluate, and determine the suitability of a connectivity technology for the system at hand. Specifically, it addresses the following questions:

- What defines an IIoT communication stack and what are the layers of that stack?
- What core functions to expect from each layer?
- What are the typical considerations and trade-offs at each layer?
- How to open up communication to participants using a domain-specific connectivity technology?
- What are core connectivity standards and what are expected from them?
- How to categorize a given connectivity technology?
- How to evaluate a given connectivity technology?
- How to determine suitability of a connectivity technology against system requirements?
- How to determine the most appropriate core connectivity standard?

1.2 BACKGROUND

The internet protocol suite[3] comprises four layers[4] as shown in Figure 1-2. The link layer provides protocols for hosts to access the computer networks. The internet layer provides protocols to exchange packets between hosts. The transport layer provides end-to-end communication services for applications by using transport protocols. The application layer provides protocols

[3] See [IETF-RFC1122]
[4] See [Clark-2017]

for message transfer, data encodings and formats (syntax), and support for different data models including semantics and metadata. The internet protocol suite does not further subdivide the application layer.

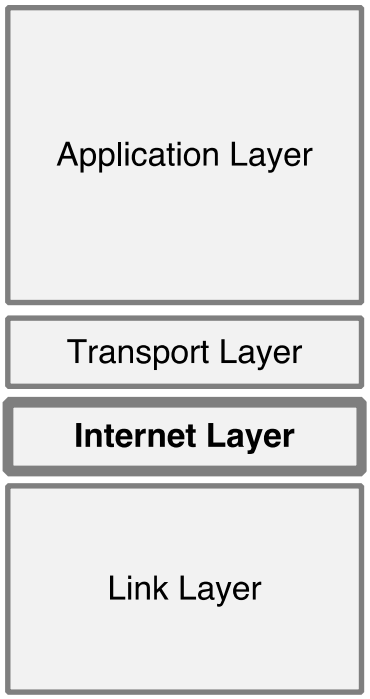


Figure 1-2: The Internet Protocol Suite

1.3 SCOPE

The scope of this document is to give a definition of the communication cross-cutting function, introduce the *IIoT communication stack* and its layers that take into consideration the needs of IIoT systems, and to provide an overview and assessment of the open connectivity framework standards applicable for IIoT solutions.

The communication cross-cutting function includes the following elements:

- the transmission of data on a physical medium,
- the transfer of data units over a network segment or networks,
- providing message exchanges carrying data units between endpoints,
- providing proper data structures (syntax) and
- and information meaning (semantics).

The communication cross-cutting function depicted in Figure 1-3 occurs at three levels of interoperability:[5] technical, syntactic, and semantic. Technical interoperability refers to the exchange of opaque blobs of data. Syntactic interoperability refers to the exchange of structured data. Semantic interoperability refers to the exchange of structured data with the intended meaning.

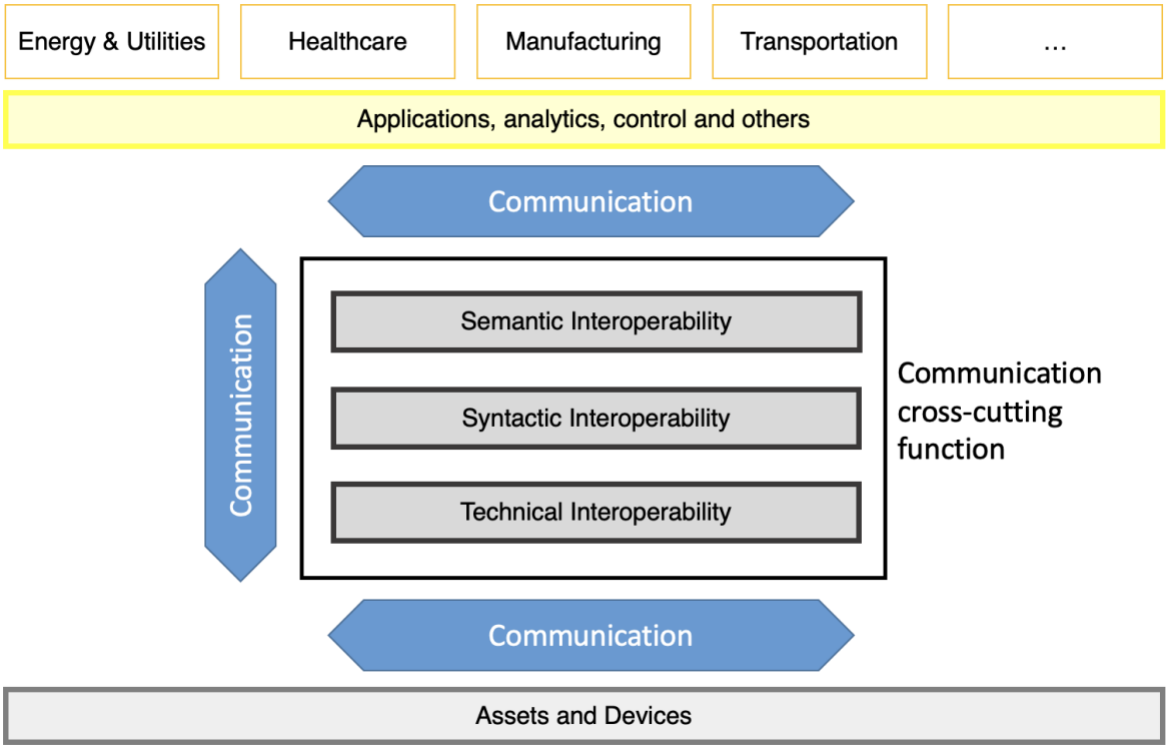


Figure 1-3: Communication occurs at three levels of interoperability in IIoT systems: technical, syntactic and semantic. We use the term "*connectivity framework*" to refer to the portion of the communication cross-cutting function that provides the abstraction of syntactic interoperability and forms the foundation for semantic interoperability.

As indicated in the figure, communication can be seen as taking place along three main paths. Communication can take place between different applications, between an application and a subsystem in an IIoT system, e.g. providing analytics, and between different IIoT subsystems especially in a distributed IIoT system. Communication can also take place between different assets (e.g. OT) and devices in deployments like a process plant. Finally, communication can take place between the assets and devices and subsystems of an IIoT system as well as towards IIoT applications.

We use the term "*connectivity framework*" to refer to the portion of the communication cross-cutting function that provides the abstraction of syntactic interoperability and elements of technical interoperability and forms the foundation for semantic interoperability. This document

[5] The ability of two or more systems or components to exchange data and use information.

provides technical details of the communication cross-cutting function needed to provide syntactic and technical interoperability.

There is a large set of networking standards available, a large set of protocols defined and in use across the layers of the IIoT communication stack model, as well as different standards on syntax, semantics, data and information models. These are in use concurrently, and a general key feature of an IIoT system is to cater for this diversity effectively and efficiently, i.e. to provide the necessary interoperability support at different layers. In this document, we cover connectivity aspects up to the level of syntactical interoperability and include technical assessments of several IIoT connectivity standards.

1.4 SUMMARY

The connectivity challenges in IIoT include meeting diverse requirements, working over many transports, and connecting a dizzying array of "things" from small devices to huge, intelligent networks of complex subsystems. And challenges are both business and technical: we shall consider the business, functional, usage and implementation viewpoints.

The connectivity framework strives for broad applicability across the IIoT and the power to handle challenging, unique applications. We introduce the notions of a *connectivity* gateway[6] and *core connectivity standards*. There are two types of gateways: *core gateways* that connect core domain independent standards, and *non-core gateways* that connect a domain-specific connectivity technology to a core connectivity standard. By domain, we intend different verticals as identified by IIC. Rather than building many bridges between many standards, each core connectivity standard need only connect to the other core connectivity standards through core gateways. The many domain-specific connectivity technologies need then interface to only one of the core connectivity standards. This strikes a balance between allowing any connectivity technology so requiring many complex bridges and allowing only one core standard that cannot span the IIoT.

The IIoT communications stack model identifies the core functions and key considerations at each layer from an IIoT perspective. We define an assessment template worksheet to understand and assess any connectivity technology objectively, and then determine the core connectivity standard closest to the technology under assessment.

Assessment templates for dealing with major IIoT system design challenges introduce connectivity framework standards. These include the Data Distribution Service (DDS) for systems requiring strict data concurrency, OPC Unified Architecture (OPC UA) for systems facing device interchangeability issues, LwM2M for web-like communication patterns and oneM2M for information and communications technology integration with wide area wireless

[6] Gateway is a base term defined in the Industrial Internet Vocabulary [IIC-IIV]: a forwarding component, enabling various networks to be connected. It may be a software component.

telecommunication provider network services. These framework standards cannot be compared on a detailed feature-by-feature level; they represent frameworks of a set of technologies providing various support for connectivity and communication in general.

The architecture integrates other connectivity technologies by interfacing to a core connectivity standard thus addressing the need for interoperability. We also provide assessment templates for common domain-specific connectivity technologies typically used at the network edge.

1.5 STRUCTURE

Chapter 2 defines the IIoT communications stack model and introduces the *Connectivity Framework* (framework) and the *Connectivity Transport* (transport) layers. It clarifies the role of connectivity in enabling syntactic interoperability, i.e. exchanging structured data, in system architecture and introduces the key system characteristics directly affected by connectivity.

Chapter 3 defines the requirements for core connectivity standards and proposes connectivity gateways to bridge a domain-specific connectivity technology to a core connectivity standard and open up hitherto inaccessible endpoints. This approach is tenable with a few core connectivity standards with core gateways for interoperability amongst them, and many domain-specific technologies that can use a gateway to any of those core standards.

Chapter 4 dives into the *connectivity framework* layer. It defines the core functions and the typical considerations and trade-offs to apply when considering a connectivity framework technology.

Chapter 5 dives into the *connectivity transport* layer. It defines the core functions and the typical considerations and trade-offs to apply when considering a connectivity transport technology.

Chapter 6 defines a template for assessing any connectivity technology from a business, usage, functional, and implementation viewpoint. It introduces a worksheet that can be used as a tool to understand, categorize and evaluate any connectivity technology.

Chapter 7 uses the assessment template worksheets to describe the prominent connectivity standards for IIoT. It also describes some of the connectivity standards prominent in specific verticals.

Chapter 8 highlights the standards that meet the requirements of core connectivity and are suitable for serving as core connectivity standards.

Chapter 9 provides guidelines on how to open up domain-specific connectivity technologies, via a core connectivity standard. It recommends completing the worksheets to identify the core connectivity standard closest to the domain-specific connectivity technology. It also makes some suggestions for a core connectivity standard based on the primary functional domain (see Figure 1-1) of applicability for the connectivity technology.

1.6 AUDIENCE

The intended audience of this document is system architects, solution architects, technology evaluators, technology decision makers, business strategists and business investment decision makers.

1.7 USE

The document is intended to be used a guide map, that can be read in its entirety in sequential order. Chapter 7 can be skimmed on the first pass; it serves as a ready reference for a deeper dive into a specific technology.

The worksheet in chapter 6 is meant as a tool for practitioners. It can be used to map out connectivity technologies of interest, in the rich landscape of connectivity.

We do not take a prescriptive approach. Instead, we provide, in the hands of system architects, a tool (see chapter 6, assessment template worksheets) that, given a system's requirements, will help determine the most suitable connectivity technologies and core connectivity standards.

1.8 RELATIONSHIP WITH OTHER IIC DOCUMENTS

The '*Industrial IoT Connectivity Framework*' (IICF) is one of many framework documents, as shown at the IIC Resource Hub[7] (see Figure 1-4).

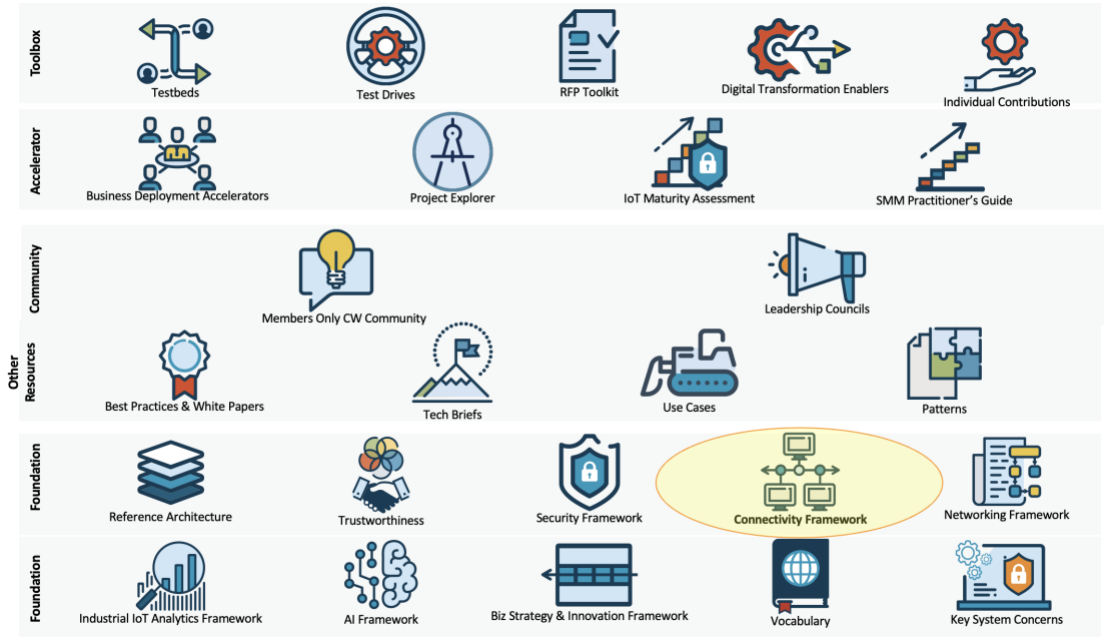


Figure 1-4: IIC technical publication organization. The Industrial IoT Connectivity Framework (IICF) is a foundational document providing practical guidance on connectivity for Industrial IoT systems.

[7] *See https://hub.iiconsortium.org/*

The IICF addresses communications as a key cross-cutting concern of IIoT systems, in particular those related to connectivity, as described by the Industrial Internet Reference Architecture (IIRA).[8] Security considerations for connectivity are described in Industrial Internet Security Framework (IISF).[9] Architectural patterns using connectivity and security are described in the Industrial Internet Reference Architecture (IIRA). This figure also shows how other documents extend from the IICF in covering connectivity related issues in the respective areas. Communications- and connectivity-related terms used in this document and their respective definitions are provided in a common Industrial Internet Vocabulary[10] document shown in Figure 1-4.

2 CONNECTIVITY FRAMEWORK

2.1 IIOT COMMUNICATIONS STACK MODEL

This section defines the IIoT Communications Stack Model. It is based on the *Open Systems Interconnect (OSI) Model* (Wikipedia)[11] and the Internet Protocol Suite (Wikipedia),[12] but has been designed to support the separation of the transport and framework layers defined in this document. It details the core functions and considerations for addressing the needs of distributed industrial sensors, controllers, devices, gateways and other aspects of IIoT systems. This section describes the scope of this document within the layers of the IIoT communication stack model.

Figure 2-1 shows the IIoT communications stack model, and the scope of the communication as a cross-cutting function within the Industrial Internet Reference Architecture (IIRA)[13]. The IIRA communication cross-cutting function, henceforth called the "communication function" in this document, provides the data-sharing mechanisms amongst participants within a functional domain and across functional domains within an IIoT system.

[8] See [IIC-IIRA]
[9] See [IIC-IISF]
[10] See [IIC-IIV]
[11] See [ISO-7498-1], for overview [WKPD-OSI]
[12] See [IETF-RFC1122], for overview [WKPD-IPS]
[13] See [IIC-IIRA]

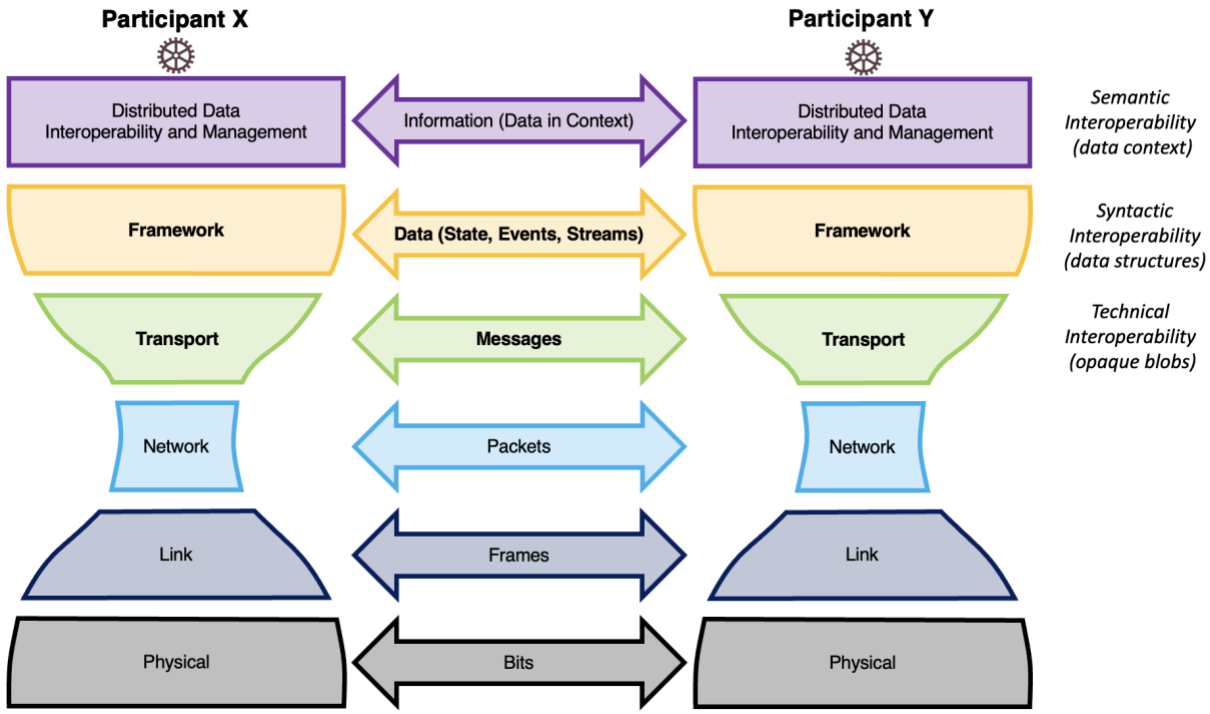


Figure 2-1: Industrial Internet communications stack model. Each layer builds on the capabilities provided by the layer below. The 'Connectivity Framework' layer provides data sharing mechanisms among participants. The 'Distributed Data Interoperability and Management' layer relies on the mechanisms provided by the 'Connectivity Framework' layer to provide meaningful information sharing.

The lowest layer is the *physical layer*, which refers to the exchange of physical signals (electric, optical or other) on the physical media (wired or wireless) connecting the participants. Above it is the *link layer*, which refers to the exchange of *frames* using signaling protocols on the shared physical link between adjacent participants. Above it is the *network layer*, which refers to the exchange of *packets* (bounded length*),* possibly routing them over multiple links to communicate between non-adjacent (remote) participants. Above it is the *transport*[14] *layer*, which refers to the exchange of *messages* (variable length) between participant applications. Above it is the *framework layer,* which refers to the exchange of structured data (state, events, streams) with configurable quality-of-service between participant applications. Above it is the *distributed data interoperability and management* layer that relies on the data sharing mechanism provided by the framework layer.

The Internet Protocol (IP) (Wikipedia)[15] is the prevailing network layer connectivity standard that has given birth to the internet and now IIoT. The IP network layer has enabled independent innovation, both below and above the network layer. The physical, link and network layers have

[14] Note the generalized definition of *transport* layer to mean "transfer" of messages. This contrasts with the transport layer in the Internet protocol suite [IETF-RFC1122], wherein specific protocols are listed.
[15] See [IETF-RFC1122], for overview [WKPD-IPS]

been in use longer—although evolution of IP and non-IP connections and the multitude of wireless-access technologies coming to market create new choices for the IIoT community.

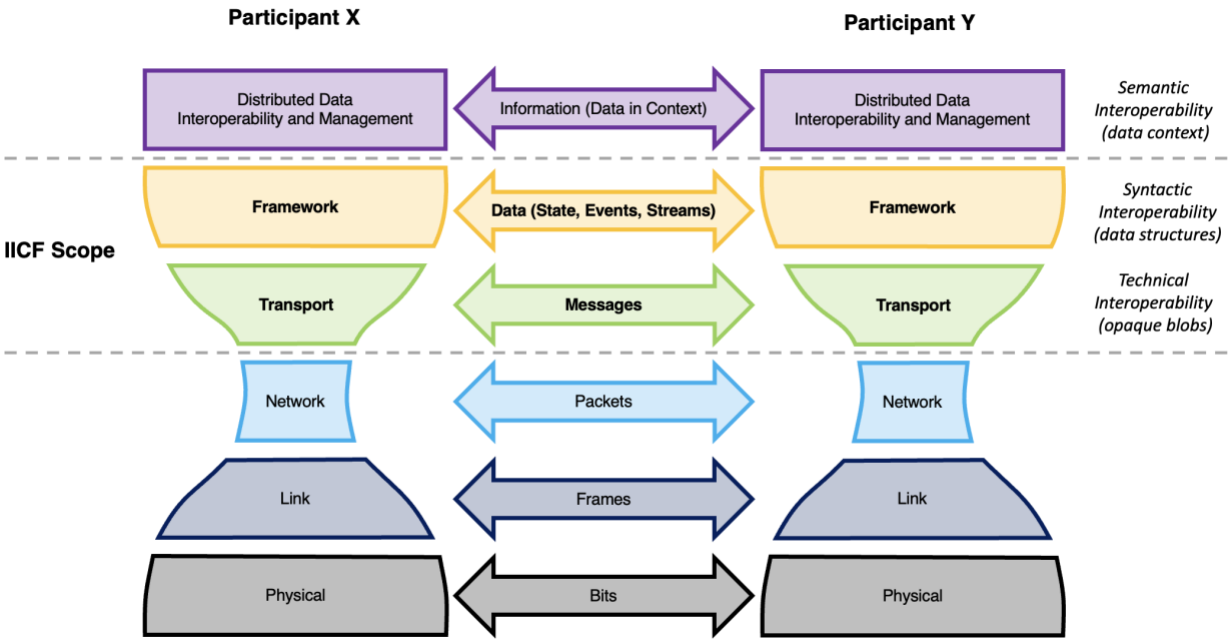


Figure 2-2: The focus of this document (IICF) is on the layers above the network layer, namely the connectivity transport and the connectivity framework layers.

The layers above the network layer have evolved rapidly in the last decade and are not as widely recognized or understood. Therefore, the focus of this document is on the layers above the network layer, namely the transport and framework layers, as shown in Figure 2-2. The remaining layers of the communication stack are covered in other IIC publications[16].

2.2 ARCHITECTURAL ROLE

The communication function in the IIRA supports exchange of data among endpoints in a system of interest. The information, for example, can be sensor updates, telemetry data, control commands, alarms, events, logs, status changes or configuration updates. Fundamentally, connectivity's role is to provide interoperable communications among endpoints to facilitate component integration.

Interoperability in communication can be achieved at various levels of abstraction, from custom integration to plug-and-play interfaces based on open standards. One common classification of interoperability is as follows (see Tolk, Wikipedia[17]):

[16] See [IIC-IINF]

[17] See [Tolk-2007], for overview [WKPD-CI]

Technical interoperability is the ability to exchange information as bits and bytes (e.g. pencil scribbles), assuming that the information exchange infrastructure (e.g. pencil and paper) is established and the underlying networks and protocols are unambiguously defined.

Syntactic interoperability is the ability to exchange information in a common data structure (e.g. using words from a language), assuming that a common protocol to structure the data is used (e.g. the language's alphabet and rules of grammar) and the structure of the information exchange is unambiguously defined (e.g. whitespace, punctuation). Syntactic interoperability requires that technical interoperability be established.

Semantic interoperability is the ability to interpret the meaning of the exchanged data unambiguously as information in the appropriate context.

For IIoT systems, the foundation for semantic interoperability comprises of two functional layers:

The *connectivity transport* layer provides the means of carrying data between endpoints. It provides technical interoperability between endpoints participating in a data exchange.

The *connectivity framework* layer facilitates how data is unambiguously structured and parsed by the endpoints. It provides the mechanisms to realize syntactic interoperability between endpoints. In this context, "common data structure" refers to the structure or schema of the data being exchanged. Familiar examples include data structures in programming languages and schemas for databases. Thus, "connectivity framework" is the portion of the communication stack up to, and including, the framework layer.

The *distributed data interoperability and management* function builds on the syntactic interoperability foundation provided by the connectivity framework and provides *semantic interoperability* required by the dynamic composition and coordination function of the Industrial Internet Reference Architecture (IIRA).[18]

The role and scope of the IIoT communication stack layers are summarized in Table 2-1.

[18] See [IIC-IIRA]

IIoT Communications Stack Model	Correspondence to Levels of Conceptual Interoperability
Distributed Data Management and Interoperability Layer	**Semantic Interoperability**: meaning of the structured data elements is interpreted unambiguously.
Framework Layer	**Syntactic Interoperability**: *Structured data types* shared between endpoints. Introduces a common structure to share data; i.e. a *common data structure* is shared. On this level, a common protocol is used to exchange data; the structure of the data exchanged is unambiguously defined.
Transport Layer	**Technical Interoperability**: *Bits and Bytes* shared between endpoints, using an unambiguously defined communication protocol.
Network Layer	Packets shared between endpoints that may not be on the same physical link. Packets are routed between physical links by a "network router".
Link Layer	Digital Frames shared between endpoints on a shared substratum (link).
Physical Layer	Analog signal modulation between endpoints on a shared substratum.

Table 2-1: Role and scope of the IIoT communication stack layers.

2.3 KEY ARCHITECTURAL QUALITIES

The communication function supports the key architectural qualities of an IIoT system, and they can be used to assess the connectivity choices for concrete architectures. They are described below.

2.3.1 PERFORMANCE

In IIoT systems, high performance connectivity is expected. The spectrum of performance ranges from tight sub-millisecond control loops to supervisory control to analysis at very low frequencies such as daily, weekly or even monthly. The performance characteristic is measured along the following axes:

- Latency is the time it takes for data to go from source to destination ("time of flight"). Jitter is the variation in latency. The data usually has a limited useful lifetime, so low latency is essential. Low jitter is also needed to ensure the application has integrity and system maintains predictable performance. The communication function addresses latency and jitter in the data exchanged between endpoints, possibly in exchange for throughput.
- Throughput is the load on the network as defined by the volume of data flow per unit of time. Bandwidth is the network capacity of a connectivity technology. In some designs, a large volume of data may be exchanged in a short time on an ongoing basis among endpoints; high throughput would be needed.

In practice, the operational settings that optimize for high throughput are not the same as those that optimize for low latency. Therefore, the communication function should support achieving the right balance as per the requirements of the data flow.

In industrial internet applications, particularly at the edge, low latency and jitter are generally more important to performance than throughput and bandwidth. Automation and control of real-world processes require short reaction times or tight coordination to maintain effective control. Industrial devices in the control domain do not produce large amounts of data in short periods and therefore do not require high bandwidth connectivity. Rather, the data needs to be communicated quickly and consistently (with low latency and jitter).

2.3.2 SCALABILITY

Physical things communicate using connectivity endpoints. Therefore, the communication function should support *horizontal scaling*, by which we mean the ability to accommodate an increasing number of connectivity endpoints, reaching internet scale.

2.3.3 RELIABILITY

The needs of the application data, like strict order of data delivery and data loss rates, determine the required level of reliability for connectivity.

The physical medium used for communicating might be constrained, especially in case of wireless connectivity mechanisms. The medium might not be constant, it might have limited transmission capabilities, and the transmission might be unreliable resulting in packet loss, delays and latency. For more information, please refer to *The Industrial Internet of Things Networking Framework (IINF)*.[19]

A connectivity framework needs to be able to consider such constraints. The framework either needs to provide mechanisms that ensure data reliability (e.g. using message re-transmission or message queuing) or to be able to function correctly even in the case of packet loss.

2.3.4 RESILIENCE

Because many IIoT systems will operate continually in a real-world environment, the communication function should be available (in the logical view), even when there is a temporary physical disconnection. When a broken connection is restored, data exchange should be automatically restored so that the latest updates are available to the consumers along with any relevant missed updates.

The communication function should support graceful failure or disconnection of endpoints, by, for example, confining the loss of data exchange only to disconnected endpoints.

2.3.5 SECURITY

Security considerations for IIoT systems are described in detail in the Industrial Internet Security Framework (IISF).[20] The chapter “Communications and Connectivity Protection” describes the following functional building blocks: physical security of connections, communicating endpoints

[19] See [IIC-IINF]

[20] See [IIC-IISF]

protection, information flow protection, network configuration and management, network monitoring & analysis and cryptographic protection, as shown in Figure 2-3.

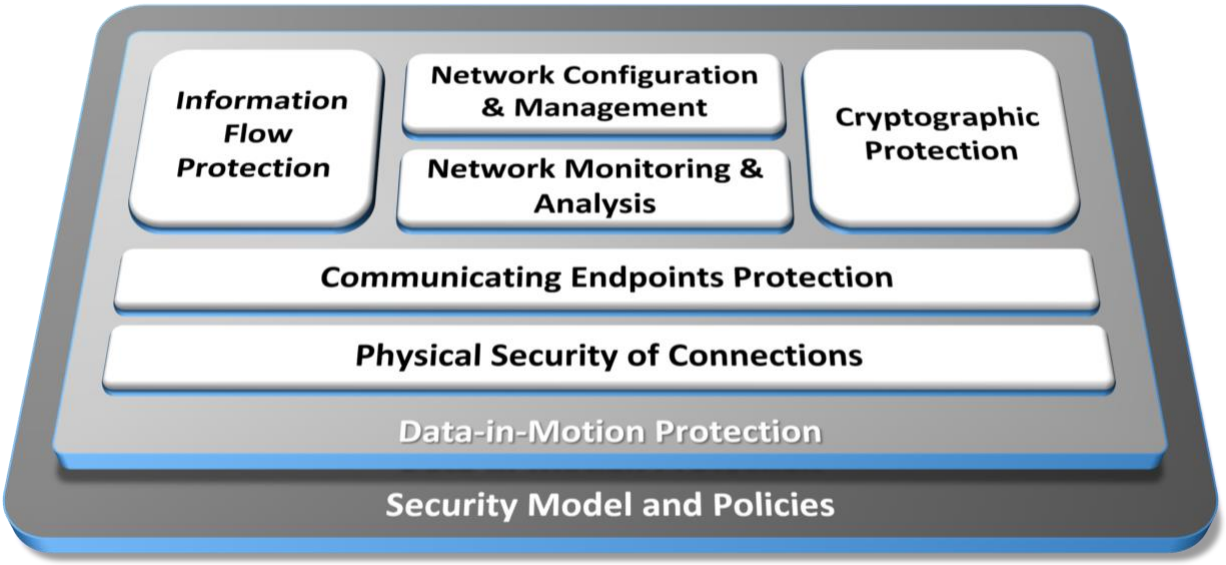


Figure 2-3: Communications and connectivity protection building blocks described in the Industrial Internet Security Framework.[21]

The security policies govern connectivity-endpoint data-exchange as part of a broader protection strategy. For example, they specify how to filter and route traffic, how to protect exchanged data and metadata (authenticate or encrypt-then-authenticate) and what access control rules should be used.

Cryptographic protection of connectivity endpoints relies on:

- explicit endpoint data exchange policies,
- strong mutual authentication between endpoints,
- authorization mechanisms that enforce access control rules derived from the policy and
- mechanisms for ensuring confidentiality, integrity, and freshness of the exchanged data.

Adequate cryptographic protection should be considered for each of the layers shown in Table 2-1.

2.3.6 LONGEVITY

Connectivity components, especially those in the network layer and below, are built into the hardware and hence are not easily replaceable. Where possible and feasible, the connectivity software components should support incremental evolution including upgrades, addition and removal of components. The communication function should also be able to support incremental evolution of the data exchange solutions during the lifecycle of a system.

[21] See [IIC-IISF]

2.3.7 INTEGRATION AND INTEROPERABILITY

IIoT systems comprise components that are often systems in their own right. The communication function should support the integration and the interoperability of system components, isolation and encapsulation of data exchanges internal to a system component, and hierarchical organization of data exchanges. In dynamic systems, the communication function should also support discovery of system components and the discovery of relevant data exchanges for system composition.

2.3.8 OPERATION

IIoT systems generally operate non-stop in a real-world environment. To support a system's operational needs, it should be possible to monitor, manage and replace connectivity elements dynamically. Monitoring includes health, performance and service-level characteristics of the communication function; management includes configuring and administering the capabilities; dynamic replacement requires replacement of hardware and or software while a system is operating.

2.3.9 SAFETY

A high degree of assurance is required in life- and mission-critical systems[22] to avoid unintended consequences during system operation. The communication function should be able to support safety evaluations and provide evidence required to make informed safety assessments.

3 CONNECTIVITY REFERENCE ARCHITECTURE

3.1 IIOT CONNECTIVITY CHALLENGE

The goal of the industrial internet is to enable seamless information sharing across domains and industries. Historically, there have been a plethora of domain-specific connectivity technologies, tightly integrated and optimized to solve domain-specific connectivity needs. IIoT systems typically include integration of brownfield technologies to preserve the legacy investments and greenfield technologies to spur innovation.

Figure 3-1 shows the challenge of building applications that require information exchange across different connectivity technologies. To facilitate information exchange, one has to build bridges to each of the other connectivity technologies. Given N connectivity technologies, this requires building $N*(N-1)/2 = O(N^2)$ bridges. That quickly becomes impractical for large N (> 3 or 4). The result is information silos, making it impossible to realize the vision of the Industrial Internet to create new value stream from heretofore locked up information flows.

[22] For example, autonomous vehicles or medical systems

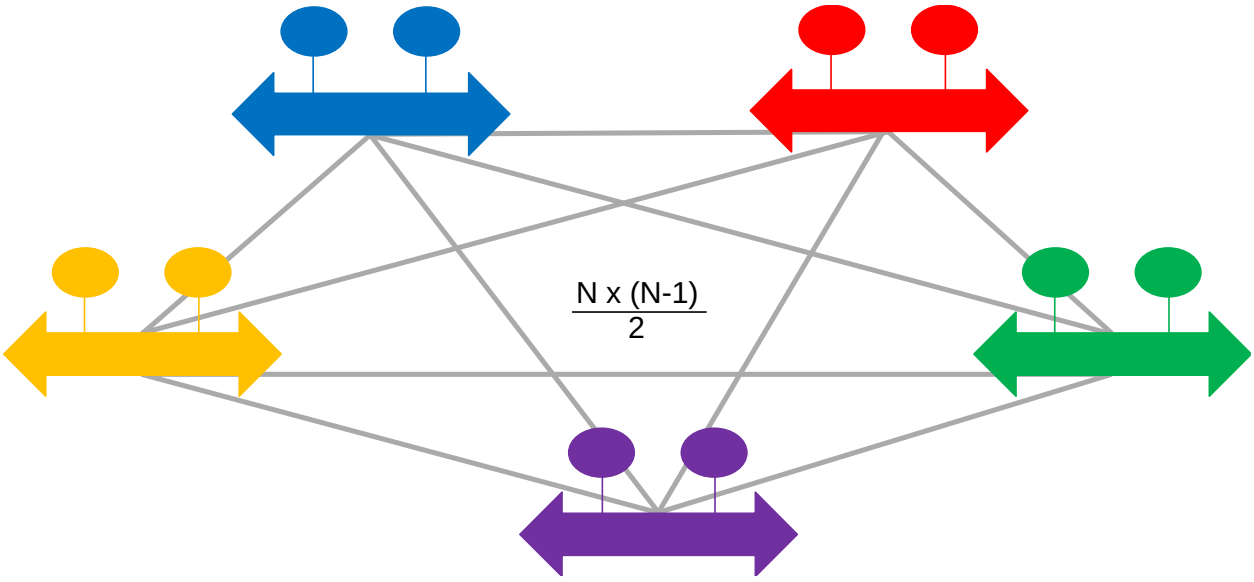


Figure 3-1: The fundamental N² (N-squared) IIoT connectivity challenge. Each new connectivity technology requires building a bridge to all the existing connectivity technologies, to facilitate information exchange between endpoints in different connectivity technologies. This approach does not scale beyond a few (small N) technologies and results in information silos.

In this document, we use the term *"domain-specific"* connectivity technology to refer to a connectivity technology that is especially suited to a particular application area. Domain-specific connectivity technologies include emerging technologies, optimized for certain use cases.

We accept that an IIoT system may require multiple connectivity technologies. Mandating a single connectivity standard across all domains and across all industries is neither realistic nor feasible. We need connectivity architectures that can address the diversity of IIoT systems, while tacking the N² challenge and enabling the vision of the industrial internet.

The rest of this section describes a connectivity reference architecture that achieves near linear scalability, O(N), with respect to the number of connectivity technologies. It accomplishes this by defining a small set of connectivity core standards. *Standardized core gateways* bridge the connectivity core standards. Domain-specific connectivity technologies need a gateway to just one of the connectivity core standards to participate in an information exchange with the rest of the IIoT ecosystem.

3.2 Connectivity Core Standards

New connectivity technologies will need to be integrated with legacy technologies during a system's lifetime. A connectivity architecture shall allow a plethora of connectivity technologies to interoperate within an industry and across industries to support the vision of an IIoT that spans industries.

A *connectivity gateway* bridges one or more connectivity technologies, as shown in Figure 3-2.

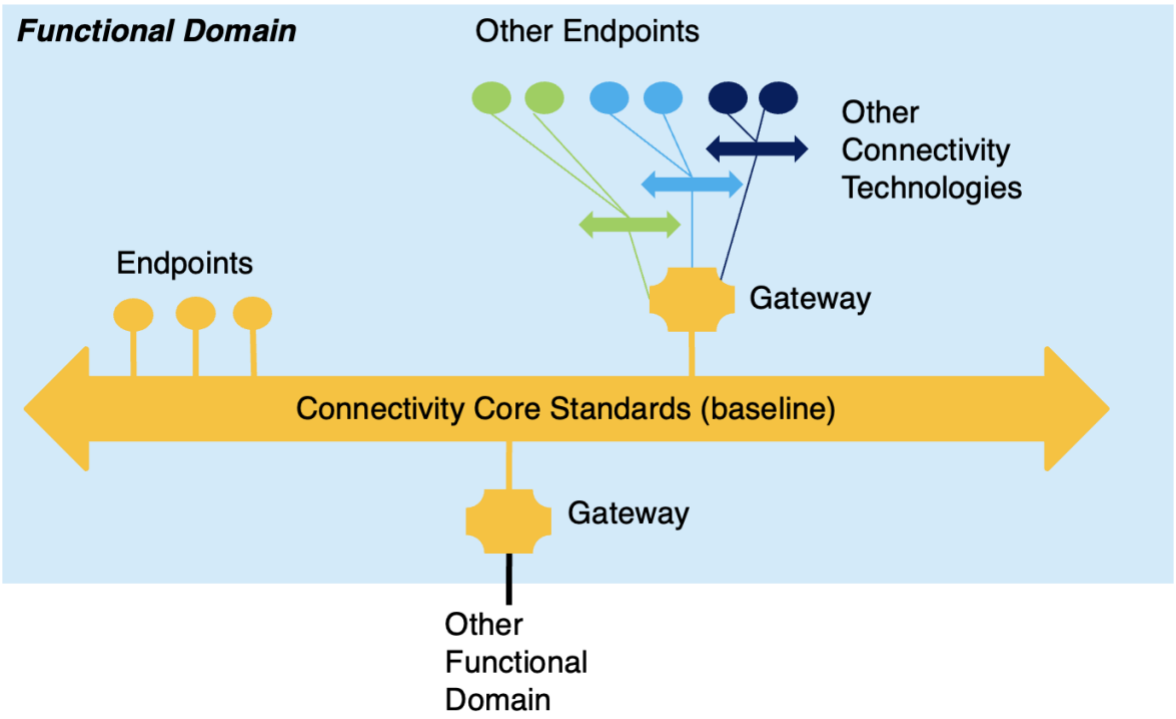


Figure 3-2: Connectivity gateway concept. A connectivity core standard technology (baseline) is one that can satisfy all of the connectivity requirements for a functional domain. Gateways provide two functions (1) integrate other connectivity technologies used within a functional domain, (2) interface with connectivity core standards in other functional domains.

To keep the connectivity architecture manageable, a connectivity technology standard is chosen as the baseline within a functional domain and referred to as the "connectivity core standard" (see Figure 3-2). Gateways are used to bridge other connectivity technologies within the domain and to the connectivity core standards used in other functional domains. Connectivity between functional domains, often implemented in a tiered manner, can be intermittent. Connectivity gateways can mitigate this intermittent connectivity. Applications are simpler and easier to maintain if logic is not needed to react to failed data exchanges.

As shown in Figure 3-2, some endpoints can connect directly to a core standard. Other endpoints and subsystems connect through gateways. A core standard then connects them all together, allowing multiple connectivity technologies to be integrated without having to bridge between all possible pairs, so avoiding the dreaded N-squared bridging problem (see Figure 3-1). Each domain-specific connectivity technology needs only a gateway to just one connectivity core standard.

Connectivity gateways enable incorporation of new connectivity technologies. They provide a stable foundation anchored in the "best-of-breed" technologies available today yet can pivot in the future to a new baseline core standard that better satisfies the requirements.

There are several kinds of connectivity gateways:

- *Framework gateways* expand the logical span of communications across connectivity framework technologies. They preserve the syntactic structure of data but may change the technical representation.
- *Transport gateways* expand the logical span of communications across transport technologies. They do not make any logical changes to the byte sequence (payload) and are transparent to it.
- *Physical/link/network gateways* convert the communications between different physical, link, and networking technologies.

In practice, connectivity gateways may span multiple layers of the communications stack (see Figure 2-1).

3.3 CORE GATEWAYS

Using a gateway to a core connectivity standard, a domain-specific endpoint can communicate with endpoints on other domain-specific technologies also connected via gateways to the core connectivity standard (see Figure 3-2). Core connectivity endpoints can directly communicate with each other, and via gateways with domain-specific connectivity endpoints.

Different functional domains may have different choices of core connectivity standards, due to different priorities on technical requirements, tradeoffs and ecosystems. To enable communication between different connectivity core standards, standardized gateways are needed. A standardized gateway between core connectivity standards is referred to as a *core gateway*. It allows domain-specific endpoints connected to one core standard to communicate with domain-specific endpoints integrated over another core standard, as shown in Figure 3-3. Also, it allows endpoints on the two core connectivity standards to interoperate.

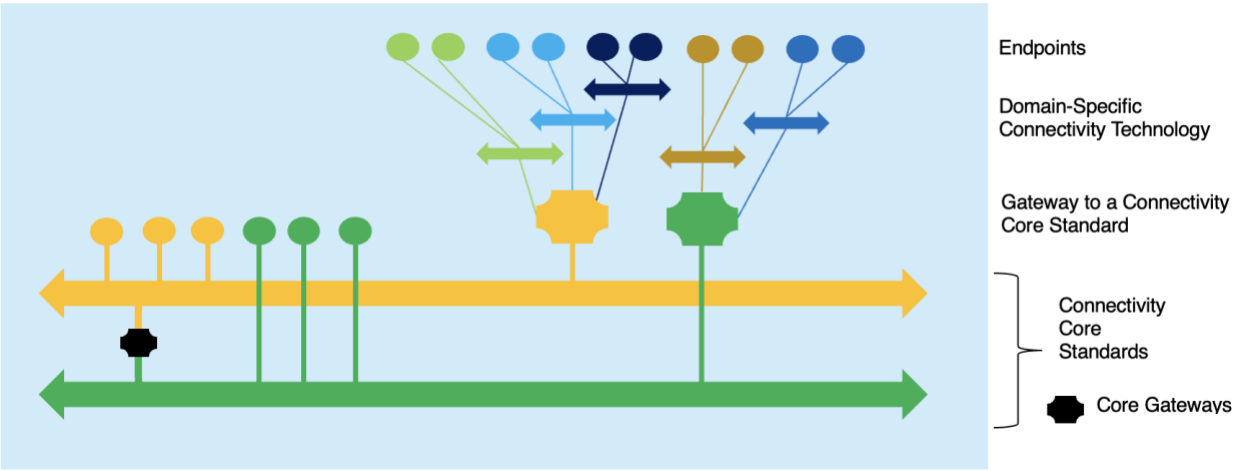


Figure 3-3: A standardized gateway between core connectivity standards can allow domain-specific endpoints connected to one core standard to communicate with domain-specific endpoints integrated over another core standard.

To realize the goals of communication across functional domain and horizontal interoperability across industries, a standardized Core Gateway shall be defined between each of the core connectivity standards, as shown in Figure 3-4.

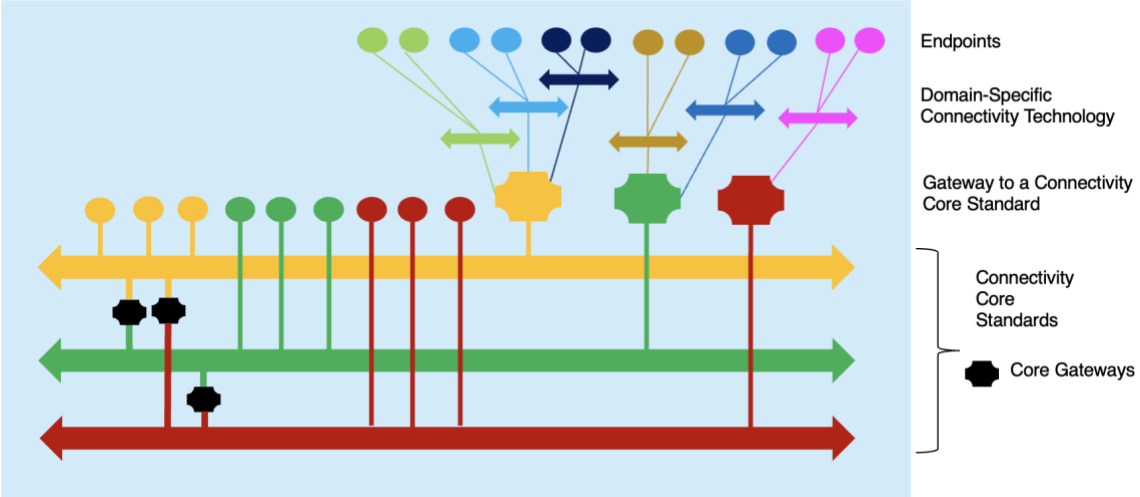


Figure 3-4: Each core connectivity standard requires a standardized gateway to all other core standards. Each additional core standard creates increasing complexity and interoperability challenges. By restricting the design to a few core connectivity standards, we cover the needs of IIoT systems across the functional domains and attain the goal of horizontal interoperability across industries.

Let K be the number of core standards. Then the number of core gateways that will be standardized is K * (K - 1) / 2, as shown in Figure 3-4. If N is the total number of connectivity technologies (see Figure 3-1), then only additional (N-K) gateways are needed with the introduction of core standards. The total number of gateways required becomes K*(K-1)/2 + (N-K) vs. the original N*(N-1)/2 shown in Figure 3-1. Assuming K << N, the number of gateways goes from $O(N^2)$ to O(N), which is much more tractable.

Each additional core standard creates increasing complexity and interoperability challenges with the square of the number of core standards. A few (small K) core connectivity standards should suffice to cover the needs of IIoT systems across the functional domains and industries to attain the goal of horizontal interoperability.

3.4 Core Standards Criteria

A connectivity core standard should align with the priorities on the requirements, engineering tradeoffs and ecosystems in its functional domain. It should not inhibit providing seamless interoperability between domain-specific endpoints connected to it via gateways. This means meeting both the functional requirements and the non-functional requirements of reliability, performance, scalability, availability, security and safety. Below, we define the criteria for qualifying as a connectivity core standard.

A connectivity core standard shall:

- provide syntactic interoperability,
- be an open standard with strong independent, international governance, 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 and
- have standards-defined Core Gateways to all other connectivity core standards.

A connectivity core standard shall provide syntactic interoperability (see section 2.2). It is not simply sending opaque blobs. Applications can get the data and discover the data types to parse and manipulate it as structured data unambiguously. So, an application will, for instance, know that the data it received is a structure with three floating-point number fields and a string field. The communications stack (see section 2.1) does not provide semantics—the interpretation of the fields, such as the units, ranges, and context are important for IIoT systems, but outside the scope of connectivity, and covered by the Distributed Data Interoperability and Management layer in the Industrial Internet Reference Architecture.

A core connectivity standard shall be an open standard managed by a recognized standards-development organization (SDO). The SDO should provide independent, international governance. There should be support for validating or certifying or testing interoperability of implementations adhering to a specification from the SDO.

A core connectivity standard shall be stable and deployed in systems across multiple industries. It should not qualify until it has been fielded and has operational proof points in fielded systems. Connectivity standards that are not proven deployed across multiple industries or in fielded systems can be considered a common connectivity standard in one or more specific industries. We should strike a balance between leading the industry and lowering risk; we set that balance at the point of deployed applications across industries.

A core connectivity standard shall have commitments from SDOs to build standards-based core gateways to the other core connectivity standards. This ensures syntactic interoperability between the core connectivity standards.

A core connectivity standard should support all the core functions of a connectivity framework. It should be fast, flexible and impose minimal overhead. It should be a proven, well-established technology, and be open and extensible to future needs of the most demanding IIoT systems.

Specifically, it should meet the following technical criteria:

- the connectivity framework functional requirements described in section 4.1, within each functional domain and across functional domains,
- the non-functional requirements of performance, scalability, reliability, resilience, within and across functional domains,
- security and safety requirements within and across functional domains

and the following business criteria:

- not require any single component from any single vendor (consistent with the internet model) and
- have readily available, professionally supported Software Development Kits (SDKs) from multiple vendors, ideally including both commercial and open source.

The technical and business criteria ensure that an endpoint can use a gateway to any core connectivity standard to communicate with other endpoints connected via a gateway to another core connectivity standard.

The design of specifying only a few core standards with core gateways amongst them mitigates the "N-squared" problem (see Figure 3-1). The core connectivity standards bear the burden of mapping to all other core connectivity standards. Core connectivity standards allow all other domain-specific connectivity technologies (standard or non-standard) prevalent within a domain to continue to be used, while providing a pathway for an open architecture to communicate with the larger IIoT ecosystem. Domain-specific connectivity technologies will need a gateway to one of the core connectivity standards. Those gateways can be products, hardware or software, standard or not. There is a practical need to limit the number of core standards to just a few, and judiciously allow for new ones to be added, if there is clearly no significant overlap with existing core standards. Otherwise we would be back again to an N^2 problem (see Figure 3-1) amongst the connectivity core standards.

4 CONNECTIVITY FRAMEWORK LAYER

The *connectivity framework* layer provides a logical data exchange service to the endpoints participating in an information exchange. It can observe and "understand" the data exchanges and use that knowledge to optimize data delivery. It is a logical functional layer on top of the connectivity transport layer (see Figure 2-1) and should be agnostic to the technologies used to implement connectivity transports.

The key role of the connectivity framework layer is to provide syntactic interoperability amongst the endpoints. Data that is exchanged is structured in a common, unambiguous data format, independent of endpoint implementation, and decoupled from the hardware and programming platform. Depending on the application logic behind endpoint, one or more data exchange patterns may be required. There are two predominant data exchange pattern styles: publish-subscribe (see section 4.1.6) and request-reply (see section 4.1.7).

A key benefit of the connectivity framework is to abstract and hide the implementation of the various functions so that the applications that use the connectivity framework won't need to know the implementation, just use its capabilities. It reduces the cost of development and increases productivity and quality.

4.1 CORE FUNCTIONS

The key connectivity framework functions include a data resource model, publish-subscribe and request-reply data exchange patterns, data quality of service, data security and a programming API. These are summarized in Figure 4-1 and described below.

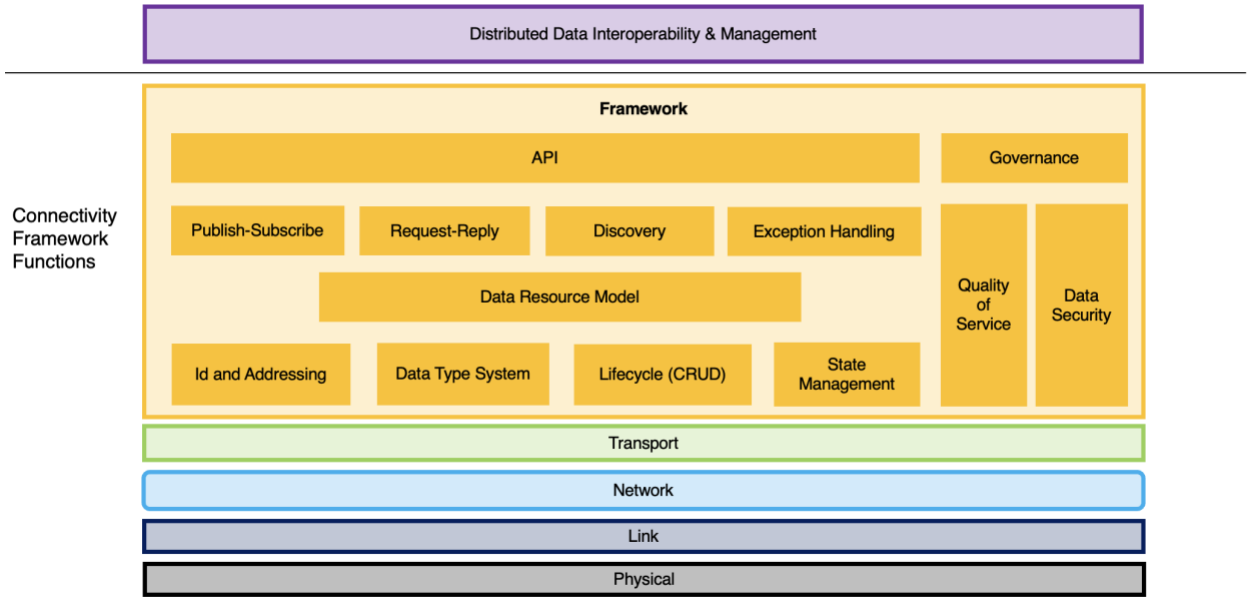


Figure 4-1: Connectivity framework layer functions.

4.1.1 DATA RESOURCE MODEL

Connectivity frameworks provide a way of representing data objects[23] that can change state over time. A data object is a structured collection of fields, as in a programming language. It may be hierarchical and may be statically or dynamically typed.

A connectivity framework distributes changes to data object amongst the participants.

Data models for different application areas or industries are usually mapped into the abstract data objects provided by a connectivity framework.

4.1.2 ID AND ADDRESSING

A connectivity framework provides the means to identify and address each data object. The *id* is used to address a data object and read and write fields in the data-object representation. It could be:

- an explicit id field in the data-object representation,
- an implicit id based on specially marked fields in the data-object representation or

[23] a.k.a. data resource or data item or data point or tag

- a uniform resource identifier (URI) within the namespace of a device or application or network endpoint.

4.1.3 DATA TYPE SYSTEM

To ensure syntactic interoperability, a communications stack (see section 2.1) will provide a way to describe the data syntax. A *data type* is a syntactic constraint placed upon the interpretation of data. It is not possible to connect systems without sharing or mapping data types, either implicitly (e.g. in code) or explicitly.

A connectivity framework provides a *data type system* for representing data objects as structures in a programming environment and for formatting data to be communicated on the wire. The data type system may be *object-oriented*, like the data types found in statically typed programming languages (e.g. C, C++, C# or Java), or *object-based* like the dynamic data types found in dynamically typed programming languages (e.g. JavaScript, Python, Lua).

The data type system should provide a means of managing the evolution of data types. This includes versioning and assignability rules across versions, so that applications using newer versions of a data type can communicate with applications using older versions of a data type to the maximum extent possible.

The data type system also defines the serialized data format in communication (in motion) and in storage (at rest), and operations to serialize from a programming language representation into the serialized format and to de-serialize back into the programming language representation.

Requiring explicit data syntax definitions using a data type system enables functional standardized gateways (see section 3.2). Also, generic tools can collect and traverse syntactically meaningful data. Explicit data typing allows intelligent or standard technologies that can map data representations.

4.1.4 DATA RESOURCE LIFECYCLE (CRUD)

A connectivity framework should provide a means to manage the lifecycle of a data object. These include the following four critical operations, abbreviated as “CRUD”:

- create (C): a means of creating or introducing a new data object,
- read (R): a means of observing the state of a data object,
- update (U): a means of updating the state of a data object and
- delete (D): a means of deleting a data object.

4.1.5 STATE MANAGEMENT

IIoT components need access to high frequency, high-volume data beyond when it was initially produced. For example, a component may require the last *n* updates to plan the next action or make a prediction.

A connectivity framework can manage the historical state of the data objects. They can cache the last *n* updates to a data object so that applications can simply examine the historical state.

Connectivity frameworks can get the current state, even though the state may have changed long before a participant joined the system or reconnected. It may maintain a virtual view of the data objects associated with an endpoint and synchronize it when a connection is re-established.

4.1.6 PUBLISH-SUBSCRIBE

A connectivity framework should support the publish-subscribe data-exchange pattern in which a component publishes data on a well-known topic without regard to subscribers, and a component subscribes to data from the well-known topic without regards to publishers. This decouples publishers from subscribers, using only the channel for the topic at hand, so that components are loosely coupled and can be replaced independently of one another. An endpoint may operate in both a publisher and subscriber role. This data exchange pattern is also called the *push* data-exchange pattern.

The publish-subscribe data exchange pattern is useful for one-to-many and many-to-many data distribution scenarios, including streaming, alarms and events, command and control, and configuration (Industrial Internet Reference Architecture, IIRA).[24]

The decoupling in space (location) and time (asynchronous delivery) provided by publish-subscribe data exchange pattern achieves the reliability, performance and scale demanded by IIoT systems. It also decreases the likelihood of fault propagation and simplifies incremental updating and evolution.

4.1.7 REQUEST-REPLY

A connectivity framework for IIoT should support the request-reply data exchange pattern. This data exchange pattern uses *requestor*s that can initiate a service request to be fulfilled by endpoints in the *replier* role. An endpoint may operate in both a requestor and a replier role. This pattern is also called a *pull* or *request-response* data exchange pattern.

The request-reply data exchange pattern is useful when working with a sparse subset of large data—for example to query specific data objects or invoke specific services.

The request-reply data exchange pattern permits synchronous or asynchronous exchange of data between endpoints. In synchronous request-reply, a requestor waits for the replies before issuing the next request. In asynchronous request-reply, a requestor can have multiple outstanding requests and replies are processed as they are received.

[24] See [IIC-IIRA]

4.1.8 DISCOVERY

To support more intelligent decisions, the discovery, authentication and access to services (including data exchanges) should be automated.

Connectivity frameworks provide mechanisms to discover the:

- publish-subscribe topics and the associated quality of service,
- request-reply services and their associated quality of service,
- data types associated with the topics and services and
- endpoints participating in a data exchange.

4.1.9 EXCEPTION HANDLING

A connectivity framework should also provide for exception handling, for example when there are disruptions in connectivity. This could happen because of:

- disconnected or intermittent links (at the lower layers),
- switching network interfaces (e.g. between wired and wireless links),
- changes in network configuration (e.g. cable replaced, network ports moved),
- data quality of service needs not met,
- remote endpoint or component failure or
- non-responsive participants.

A connectivity framework should shield the data flows from the impact of such exceptions and should provide a means of informing the applications when an exception cannot be automatically managed by the connectivity framework.

4.1.10 DATA QUALITY OF SERVICE (QOS)

IIoT data exchanges can have varying requirements for how the data is delivered. Those aspects are referred to as the data quality of service (QoS).

A connectivity framework should support these data exchange QoS categories.

Delivery refers to the delivery aspects of the data including:

- *Best-efforts delivery*: An update is sent once, regardless of whether the receivers get it. Also called a *fire-and-forget* scheme, this is a form of “at most once” delivery. It is suitable when high-frequency periodic updates need to be distributed in a system and out-of-order or missing updates can be tolerated.

- *Reliable delivery*: An update is sent and also cached by the sender for later redelivery in case receiver(s) don’t get it in a timely fashion. The amount of caching and timing can be configured based on the application and data flow requirements. Acknowledgements from a receiving endpoint can be automatic at the connectivity framework level or may require explicit response from the application. This is a form of “at least once delivery”. It

is suitable for low frequency status updates, events and notifications and also for commands when updates from a source are expected in-order.

In addition:

- *Timeliness* is the ability of the connectivity framework to establish end-to-end timing constraints, adaptively reconfigure to either guarantee specified timing or minimize timing violations, and to notify the application if a timing constraint has been violated.
- *Ordering* is the ability of the connectivity framework to present the data in the order it was produced or received and collate updates from different sources in the system.
- *Durability* is the ability of the connectivity framework to make data available to late joiners and extend the lifecycle of the data beyond that of the source when so desired and survive failures in the infrastructure.
- *Lifespan* is the ability of the connectivity framework to expire stale data.
- *Fault tolerance* is the ability of the connectivity framework to ensure that redundant connectivity endpoints are properly managed, and appropriate failover mechanisms are in place when an endpoint or a connection is lost.

The underlying transport layer will ultimately bound a connectivity framework's performance and scalability limits. The connectivity framework should introduce minimal overhead in providing the data exchange QoS and should have minimal impact on the overall performance and scalability.

4.1.11 DATA SECURITY

A connectivity framework should provide the ability to ensure confidentiality, integrity, authenticity and non-repudiation of the data exchange, when so desired.

The connectivity framework security mechanisms should provide a means to:

- upon discovery, authenticate endpoints before allowing them to participate in a data exchange,
- authorize permissions (read, write) granted to the endpoints participating in a data exchange, to ensure that endpoints cannot write or read data that they have not been given access to,
- ensure data integrity and trustworthiness of the data delivery, so that received data is not tampered with while stored or in transit and
- selectively encrypt sensitive data flows.

This last point is important since certain high volume data flows may not be sensitive enough to warrant the extra overhead of encrypting and decrypting the data. The decision to encrypt should be based on a risk-impact assessment.

The connectivity framework access-control-model should be sufficiently fine-grained to limit the permissions of each endpoint narrowly to the operations and services needed for performing their intended functions. This enables the application of the *principle of least privilege* that is essential to limit the consequence of security breaches and insider attacks.

The connectivity framework security mechanisms should provide secure logging and auditing capabilities to detect security attacks and assess their consequences.

For more details, please refer to the Industrial Internet Security Framework (IISF)[25].

4.1.12 API

IIoT systems involve multiple software components, developed by multiple parties over time, with a variety of programming languages. Therefore, IIoT software development requires an *Application-Programming Interface* (API) to support the design and implementation of application-specific data exchanges.

Some connectivity frameworks provide standardized APIs in various programming languages (e.g. C, C++, C#, Java, Python, Lua, JavaScript, and so on), to ease the portability of application code from one implementation to another and to decouple the application from the framework implementation. Others define a protocol interface, and let the implementers define the programming API. This makes it harder to switch implementations but allows the APIs to be customized to taste.

4.1.13 GOVERNANCE

A connectivity framework should provide a means to configure, administer, and monitor its operation. These include all aspects of the connectivity framework functions, including data types, data quality of service, data security policies, resource management and timing.

Some connectivity framework standards define the mechanisms for configuration and administration. Others do not standardize on the mechanisms and leave it up to the implementations. Mechanisms may be file based or API based or both.

Monitoring is useful for diagnostics and troubleshooting of an operational IIoT system. It should be configurable to the desired level of detail. Connectivity framework standards may define the mechanisms for monitoring or may leave them up to the implementations.

4.2 TYPICAL CONSIDERATIONS

Typical considerations for choosing a connectivity framework can be grouped into system, data, performance, scalability, availability, deployment and operational considerations. The tradeoffs in each should be carefully evaluated.

[25] See [IIC-IISF]

4.2.1 System Architecture Considerations

4.2.1.1 Peer-to-Peer vs. Broker

Peer-to-peer is a symmetric data exchange pattern between endpoints without any intermediary or broker. A peer-to-peer architecture provides the lowest latency and jitter data exchange between endpoints. It can also avoid startup dependencies, as peers can come up in any order. There is a no single point of failure or vulnerability. However, a distributed peer-to-peer based system requires more careful planning—for example, one may need relays to avoid undue load on extremely resource constrained peers.

On the other hand, a *broker-based architecture* requires running a centralized process on a host in the system. Data exchanges flow through the broker. It needs to be started and run before the endpoints can communicate. A broker can become a choke point and a single point of failure, if not mitigated by redundancy and load balancing. Latency and determinism can suffer, especially as data volume goes up. It can increase vulnerability from a security perspective, but provisioning and management can be centralized.

4.2.1.2 Data-Centric vs. Device/App-Centric

In a *data-centric* architecture, the endpoint application code does not need to be aware of who produces or consumes the data. Data is regarded as a first-class citizen that can be exchanged, stored, transformed and manipulated in its own right, independently of the endpoints that produce or consume it. There is an analogy with databases, which provide a data-centric abstraction for *data-at-rest*. Data-centric connectivity frameworks provide a data-centric abstraction for *data-in-motion.* Integrating new applications requires them to have knowledge of the data-centric abstraction.

In a *device-centric* architecture, the endpoint application code is aware of the devices or application endpoints responsible for producing or consuming data. Devices or application endpoints are regarded as the first-class citizen and are modeled as such; data cannot exist without the context of a device or application. Integrating new types of devices or applications requires integrating new interfaces.

Data-centric connectivity frameworks provide *location, device* and *application independence.* They allow components to be decoupled from one another, developed and integrated independently. Device-centric connectivity frameworks require application components to understand the device context to use the data meaningfully. Each kind of device is integrated separately, and the applications are aware of their behavioral interfaces.

Data-centric connectivity frameworks fit well with multi-party development, simplify the integration effort and reduce the overall effort and time to debug and integrate components into systems. Because the interfaces to the data are explicit in the design, a data-centric approach

results in an *open architecture*. Components become independent of the data, thereby simplifying the system development and evolution and increasing reusability. Data-centric systems also scale well to large systems. However, it can be challenging for IIoT systems integrators to require their vendors use a common data-centric abstraction.

For IIoT systems, open architecture is highly desirable, since it allows multiple data sources to be combined together to generate new value, insights and applications.

Many system designers choose between device-centric or data-centric core connectivity standards based on the relative need of integrating similar brands of devices vs. reducing complexity and easing development of the software. Regardless of the chosen method, connectivity frameworks should allow for both approaches.

4.2.1.3 EXPLICIT VS. IMPLICIT GOVERNANCE

Connectivity framework governance (see section 4.1.13) may be explicit or implicit or a mix. When governance is *explicit*, configuration elements can be controlled independently of the applications; they can be shared and managed through a common repository. When governance is *implicit*, configuration elements are embedded within the application code across the various system components.

Explicit governance allows data architecture evolution and upgrades in a controlled fashion, independently of the application code. This is beneficial for large teams working on safety-critical systems where the development needs to be carefully managed, while enabling multiple sub-teams to work independently.

Implicit governance works best for organic evolution and requires that the data architecture be discoverable via dynamic APIs.

4.2.2 DATA CONSIDERATIONS

4.2.2.1 CONTENT-BASED SELECTION

IIoT systems involve movement of large volumes of data. Components are only interested in a specific subset of the data at a given time, although that interest set may change. Given the data interest set across the components, connectivity frameworks can optimize the data distribution, resulting in lower overall system resource footprint, and so lower system cost.

For IIoT systems, the ability to specify a content-based data subset of interest and automatically optimize the data flows is highly desirable.

4.2.2.2 TIME-BASED SELECTION

IIoT systems typically involve distribution of high frequency data. A component may produce data faster than some consuming components desire or are able to handle. In this situation, time-based filtering in the connectivity infrastructure is required. For example, a sensor may generate

data at a rate of 1000Hz rate, but a user display may not require data at a rate faster than the display refresh rate of, say, 30Hz.

A consumer's desired data rate may change over time or for different data items. By knowing the desired data-rate needs across components, connectivity frameworks can optimize the use of system resources for data distribution. This can result in lower overall system resource footprint and lower system cost.

For IIoT systems, the ability to specify a time-based data subset of interest and automatically optimize the data flows is highly desirable.

4.2.3 Performance Considerations

4.2.3.1 Real Time

"Real time" is more about deterministic response than it is about fast response. Many systems require low average latency, but real-time systems succeed only if they always respond, "on time". This is the maximum latency and can be expressed as the average delay plus the variation or jitter. Even a fast server with low average latency can experience large jitter under load. For real-time operation, the latency needs to be predictable (i.e. the jitter should be consistently small).

4.2.3.2 Latency and Jitter vs. Throughput

Throughput refers to the volume of data distributed per unit time. The throughput demands can vary widely—for example, under load or stress in an emergency situation, there may be a lot more communication compared to normal or steady state operation. Latency and jitter can suffer when throughput demands increase on the connectivity infrastructure without increased capacity. A connectivity framework should be able to meet the latency and jitter requirements for real-time performance.

For IIoT systems, the latency and jitter vs. throughput tradeoffs should be carefully evaluated and the limiting factors for throughput and latency should be understood.

4.2.4 Scalability Considerations

4.2.4.1 Data Objects

When the number of data objects increases, it is no longer practical to send every update to every possible consumer. Connectivity frameworks should support data-object scaling by offering run-time introspection so consumers can choose data objects of interest and configure producer update distribution to a sparser set currently of interest. A producer can also batch multiple data-object updates destined for the same consumer to make the data distribution efficient and scalable.

For IIoT systems, a connectivity framework should effectively handle an increasing number of data objects as more memory resources are added and should support data objects of varying sizes.

4.2.4.2 APPS

IIoT systems comprise independently developed applications, each with evolving interfaces and data formats that should continue to interoperate with older versions of that interface. Forcing all apps to coordinate their update simultaneously to a new interface and data format version is unrealistic; there needs to be version negotiation between components. Interacting teams need tools, processes and eventually architectural support to solve the system-integration problem.

Data-centric connectivity frameworks allow applications to control data-oriented interfaces directly. Applications interact with shared data objects described by explicitly defined data types. Differences in data-oriented interfaces between apps can be automatically detected and adapted to match a participant's expectations, so as to decouple application interface dependencies and allow large projects to evolve interfaces and make parallel progress on multiple fronts.

For IIoT systems, a connectivity framework should support component interface evolution so that new capabilities can be added over time, without impacting the already existing components.

4.2.5 AVAILABILITY CONSIDERATIONS

4.2.5.1 REDUNDANCY

Failure of IIoT systems during operation can have fatal consequences. The system-relevant time period of continuous operation is dependent on the context of the system. For a power plant, the relevant time period could span years. For a medical imaging machine, the relevant time period could be only a few seconds.

But hosts and networks do fail, so redundant infrastructure (duplicate or triplicate or more) and failover mechanisms should be put in place. They rely on the connectivity infrastructure to communicate fault conditions and effect the appropriate state changes. To provide continuous system availability, a connectivity framework should support redundant endpoints and networks, and remove duplicate data transparently when the same update is received over multiple paths.

For IIoT systems, a connectivity framework should support continuous availability over a system-relevant time period.

4.2.5.2 RECOVERY

An IIoT connectivity framework should be continuously available. It should not have single points of failure and it should provide mechanisms for timely detection of system component failures. There should be mechanisms for component state durability and for state recovery and failover.

For IIoT systems, a connectivity framework should provide mechanisms for data durability and state recovery from fault conditions.

4.2.6 DEPLOYMENT CONSIDERATIONS

4.2.6.1 PLATFORM CONSTRAINTS

IIoT system components run on a variety of platforms, from small resource-constrained devices to enterprise-class machines. Platform constraints can be about CPU power, memory capacity, power availability or connectivity capabilities. Generally, the development environment is different from the deployment environment. The memory footprint, CPU, programming language and environments can vary greatly across these hosts. The connectivity platform should be supported on the development and deployment compute platforms used in the system.

For IIoT systems, a connectivity framework should support the operating system, the CPU and the resource constraints on the platform(s) being used.

4.2.6.2 INCREMENTAL UPGRADES

For IIoT systems that have long lifespans, components are upgraded incrementally. Newer or updated components may use newer versions of connectivity framework software. Connectivity framework that supports backwards and forward version compatibility can facilitate the upgrade process.

For IIoT systems, a connectivity framework should provide backwards compatibility of communication protocols and data structures, so that components can be upgraded incrementally.

5 CONNECTIVITY TRANSPORT LAYER

The key role of the *connectivity transport* layer is to provide technical interoperability among the endpoints.

5.1 CORE FUNCTIONS

The key connectivity transport functions include endpoint addressing, modes of communication, network topology, connectedness, prioritization, timing and synchronization and message security. These are summarized in Figure 5-1 and described below.

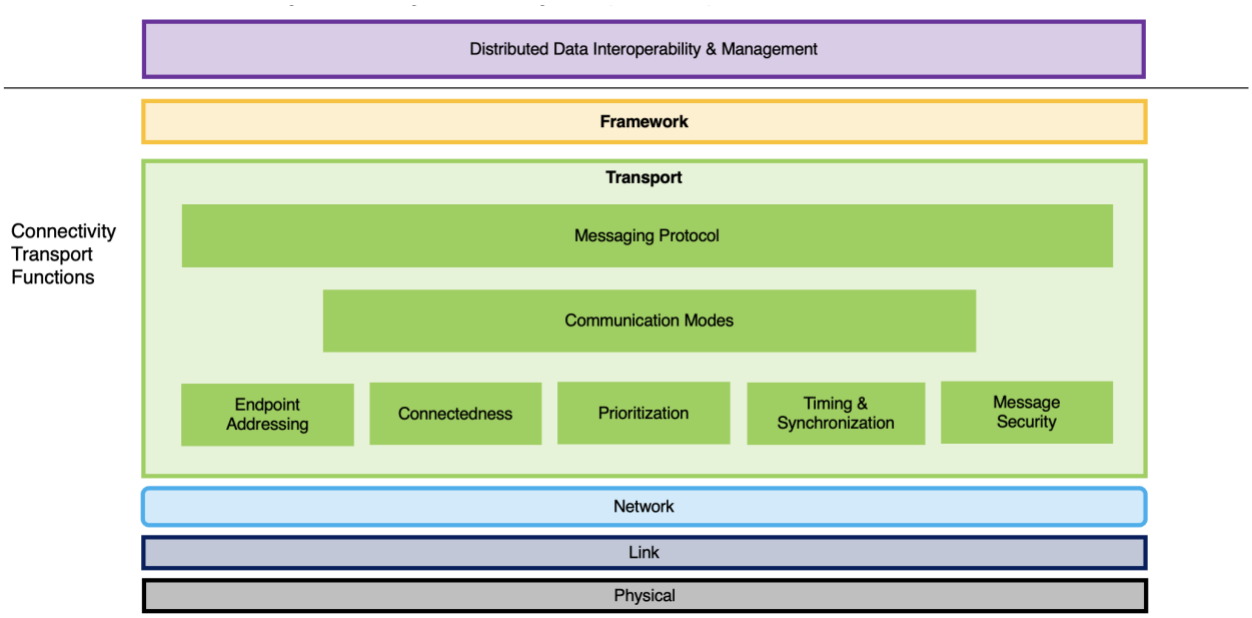


Figure 5-1: Connectivity transport layer functions.

5.1.1 Messaging Protocol

The messaging protocol is the wire protocol that describes the format and behavior of the messages exchanged between the endpoints. A messaging protocol should support the implementation of connectivity framework layer functions (see section 4.1). It may be directly exposed for use by applications, possibly using ad hoc (and unnamed) application-specific connectivity frameworks.

The messaging protocol may include discovery, authentication, session establishment, message retry and acknowledgment, fragmentation and re-assembly of large messages, data encoding and serialization, message reorder and de-conflicting across connectivity transports.

Messaging protocols may be configured and optimized for different network layer configurations. Network layer parameters such as bandwidth, round-trip time and maximum message size should inform the selection of the messaging protocol quality of service.

5.1.2 Communications Modes

A connectivity transport may support the following communication modes:

- unicast—suitable for one-to-one communication between two endpoints,
- multicast—suitable for one-to-many communication between endpoints and
- broadcast—suitable for one-to-all communication between endpoints, where “all” refers to all the endpoints present on the communication transport network at the time of transmission.

5.1.3 ENDPOINT ADDRESSING

Any of the nodes (for example, a device or an application host) in IIoT systems can house one or more components, each with one or more connectivity endpoints. An address identifies a node for network-level communication purposes. This address could be locally unique and possibly globally unique. A node and hence the endpoints residing on it may be reachable over multiple addresses.

The addressing scheme and associated infrastructure should support endpoints on the internet scale.

5.1.4 CONNECTEDNESS

Network layer protocols (see Figure 2-1) offer either connection-oriented or connectionless services for delivering packets across the network. Connectionless services are more common at the network layer. In many protocol suites, the network-layer protocol is connectionless, and the transport layer provides connection-oriented services. For example, in TCP/IP, the Internet Protocol (IP) and the User Datagram Protocol (UDP) layered on top of it are connectionless, while the Transmission Control Protocol (TCP) is connection oriented.

A connectionless transport is best for low latency and jitter applications or when a high degree of scalability is required in a local area network. The connectionless UDP transport has proven itself for use in real-time applications.

A connection-oriented transport is best suited for high-throughput applications in a network with complex topology and high variation of traffic loads, since it provides a “virtual circuit” that reduces the variation in routing path. The connection-oriented TCP transport is battle tested for transiting through *firewalls* and *network address translation* (NAT) routers and connecting across wide area networks. New applications may call for connection-oriented connectivity transports that do not suffer the drawbacks that we find in TCP today, such as unbounded retransmission delays.

When using a connectionless transport, the connectivity framework design needs to handle failures in the transport caused by loss or out-of-order packets. Consequently, designing a connectivity framework based on the connection-oriented transport may preclude it from providing a connectionless data exchange.

5.1.5 PRIORITIZATION

IIoT systems need to ensure that critical data is delivered ahead of non-critical data.

The connectivity transport function can provide the ability to prioritize some messages over others in the data exchange between endpoints.

5.1.6 TIMING & SYNCHRONIZATION

IIoT systems need a way to synchronize local endpoint clocks over a connectivity transport network. Many methods are in use today, including NTP- or PTP-based time synchronization and GPS clocks, and new approaches are in development.

The connectivity transport function may provide the ability to synchronize time across the network.

5.1.7 MESSAGE SECURITY

The security mechanisms provided by the connectivity transport layer should implement and enforce the connectivity-framework-layer data security function (see section 4.1.11).

Transport layer security involves both the messaging protocol and the network layer security. Both should provide mechanisms for endpoint authentication, message encryption and message authentication. Security implemented by each function may provide controls with different granularity and be separately administered.

At the network level, network endpoint security mechanisms can grant access based on policy and enforce security by means of encrypted virtual local area networks (VLANs) and firewalls.

At the messaging protocol level, message-oriented security mechanisms based on policy can enforce permissions by fine-grained cryptographic means. For example, different data flows may be configured to use different cryptographic keys such that permissions granted to an application to access one flow does not allow it to observe a different flow.

There may be multiple transport and network hops between endpoints. End-to-end security is desired, and security should not be compromised when crossing gateways, proxies and bridges between the endpoints.

For more details, please refer to the Industrial Internet Security Framework (IISF).[26]

5.2 TYPICAL CONSIDERATIONS

5.2.1 NETWORK LAYER CONSIDERATIONS

5.2.1.1 TOPOLOGY

A transport may require or preclude a specific network topology. Network topologies in IIoT systems include:

- point-to-point,
- hub-and-spoke,
- meshed,

[26] See [IIC-IISF]

- hierarchical and
- a combination of the above.

Connectivity gateways can be used to bridge multiple networks and topologies, and to form more complex topologies, as needed.

For IIoT systems, a transport should not restrict the network topology.

5.2.1.2 SPAN

A transport communication path may span across different physical geographies. A logical transport layer can span the local area networks (LAN), large geographic distances (wide area networks i.e. WAN), or somewhere in between (metropolitan area networks i.e. MAN).

For IIoT systems, it is desirable for a transport to support a variety of network spans, including, LAN, MAN, WAN and possibly space networks.

5.2.1.3 SEGMENTATION

IIoT systems need a way to separate data from different functional domains over the same network.

The transport may provide the ability to segment a network, to isolate different functional domains and one set of data exchanges from another.

For IIoT systems, it is highly desirable that the connectivity transport be able to support multiple independent and isolated communication paths between the same network endpoints.

6 HOW TO ASSESS A CONNECTIVITY TECHNOLOGY

We apply the Industrial Internet Reference Architecture (IIRA)[27] viewpoints to create an *assessment template* for use in evaluating any connectivity technology. The communication functions (see sections 4.1 and 5.1) constitute the functional viewpoint; the typical considerations (see sections 4.2 and 5.2) constitute the implementation viewpoint. The business and usage viewpoints are described below.

The assessment template is intended to be a tool for understanding any connectivity technology in the context of the IIoT needs. The worksheet is helpful for:

- understanding how a connectivity technology supports specific IIoT functional needs,
- evaluating a connectivity technology's trades-offs for typical IIoT considerations and
- determining a connectivity technology's suitability for a particular use case (once the specific requirements are understood).

[27] See [IIC-IIRA]

The worksheet helps categorize objectively a connectivity technology across the layers of the IIoT communications stack model (see Figure 2-1) based on the functions it supports: is it a connectivity framework (Figure 4-1) or a connectivity transport (Figure 5-1)? Some technologies span multiple layers of the communications stack (Figure 2-1).

Connectivity technologies can be compared objectively, and the most applicable connectivity technology can be easily identified.

The worksheet is described below.

6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.*
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.*
Description	*Short synopsis of the technology.*
Application Domain(s)	*Application domains targeted by the connectivity technology.*
Dependencies	*Possible commonalities with or reliance on other connectivity elements.*
References	*Website and other useful links to the technology.*

6.2	BUSINESS VIEWPOINT	
6.2.1	PURPOSE	*Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.*
6.2.2	PEDIGREE	*Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.*
6.2.3	VARIANTS	*Describe the options and variants from the original generic description of the technology.*
6.2.4	MATURITY	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?*
6.2.5	STABILITY	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced. How easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?*
6.2.6	STANDARDS BODY	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.*
6.2.7	OPENNESS	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?*

6.3 USAGE VIEWPOINT	
6.3.1 ARCHITECTURE	*Summarize the main concepts, and high-level architecture and terminology. Describe the end-to-end information exchange path.*
6.3.2 TECHNOLOGY OPTIONS	*List the choices to be made for using the connectivity technology in a system.*
6.3.3 APPLICATIONS	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.*
6.3.4 TYPICAL USAGE *(Section 2.2)*	*What function or where in the system this technology is typically used?*
6.3.5 OPERATIONS *(Section 2.3.8)*	*Can one monitor, manage and dynamically replace elements of the communication function?*
6.3.6 SECURITY *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?*
6.3.7 SAFETY *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?*
6.3.8 GATEWAYS *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.*

6.4 FUNCTIONAL VIEWPOINT

6.4.1 CORE FRAMEWORK LAYER FUNCTIONS

Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.*
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.*
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.*
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data object's lifecycle? Summarize the salient aspects.*
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.*
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.*
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.*
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.*
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.*
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.*
Data Security *(Section 4.1.11)*	*Does it provide a data object security model? Summarize the salient aspects.*
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?*
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.*

6.4.2 CORE TRANSPORT LAYER FUNCTIONS

Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message-size limitations? What are the usage assumptions? Is it optimized for certain message requirements?*

6.4 FUNCTIONAL VIEWPOINT	
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?*
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?*
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.*
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.*
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.*
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.*

6.5 Implementation Viewpoint

6.5.1 System Architecture Considerations

Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?*
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints to participate in information exchange?*
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?*

6.5.2 Data Considerations

Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?*
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?*

6.5.3 Performance Considerations

Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?*
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?*

6.5.4 Scalability Considerations

Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data object size?*
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?*

6.5.5 Availability Considerations

Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?*
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?*

6.5.6 Deployment Considerations

Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system, the CPU and the resource constraints on the platform(s) being used?*

6.5 IMPLEMENTATION VIEWPOINT	
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?*
6.5.7 NETWORK LAYER CONSIDERATIONS	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?*
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?*
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?*

7 CONNECTIVITY STANDARDS

This chapter lists the prominent connectivity standards for the framework and transport layers, using the assessment template defined in chapter 6, so that the standards can be understood in the context of IIoT needs. As additional standards are identified, they should be added to this list. The assessment template allows us to capture and describe the technologies in a uniform and objective manner.

Figure 7-1 summarizes the prominent IIoT connectivity frameworks and transports. It shows connectivity frameworks that have originated in an industry-agnostic manner for general-purpose use. Some frameworks define their own transport protocols (e.g. DDSI-RTPS), other frameworks use general-purpose transport protocols defined elsewhere (e.g. HTTP, CoAP) and some frameworks use both. All transport protocols covered in this specification run on top of TCP, UDP or both. Each section describing a framework will provide more details on the transport protocols used for that framework. The network (IP) and lower layers (wired and wireless) are also shown for completeness but are outside the scope of this document.

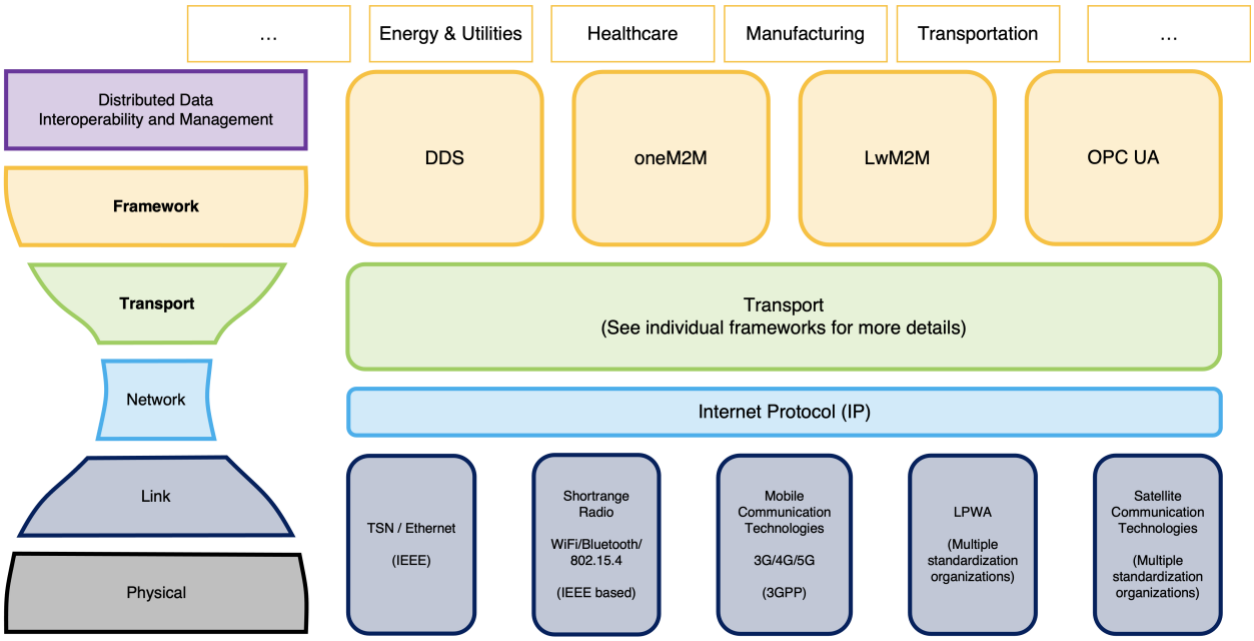


Figure 7-1: IIoT connectivity standards.

The distinction between transport and framework layers is important. To be considered a connectivity framework at a minimum, a connectivity transport would have to be paired with a data type system. For instance, a connectivity transport such as MQTT could be paired with data type system technology such as protocol buffers,[28] and could be used to create a custom connectivity framework. However, there is currently no standard that describes such a pairing.

Some connectivity frameworks, like oneM2M also cover the Distributed Data Interoperability and Management layer. While this is shown in Figure 7-1, layers above the framework are not in the scope of this document.

The connectivity framework and transport standards shown in Figure 7-1 are discussed below. Detailed assessments are provided, starting from Annex A.

7.1 IIoT Connectivity Framework Standards

7.1.1 Data Distribution Service (DDS)

Data Distribution Service (DDS) is an open connectivity-framework standard specifically targeted at IIoT applications. The Object Management Group (OMG) DDS Foundation Portal[29] maintains the DDS family of specifications.[30] The core specifications include Data Centric Publish-

[28] See [GOO-PB]
[29] See [OMG-DDS]
[30] See [OMG-DDS-SPECS]

Subscribe[31], Remote Procedure Calls over DDS,[32] Real-Time Interoperability Wire Protocol,[33] Extensible and Dynamic Type System and Serialized Data Representation,[34] Security Model and Service Plugin Interface.[35]

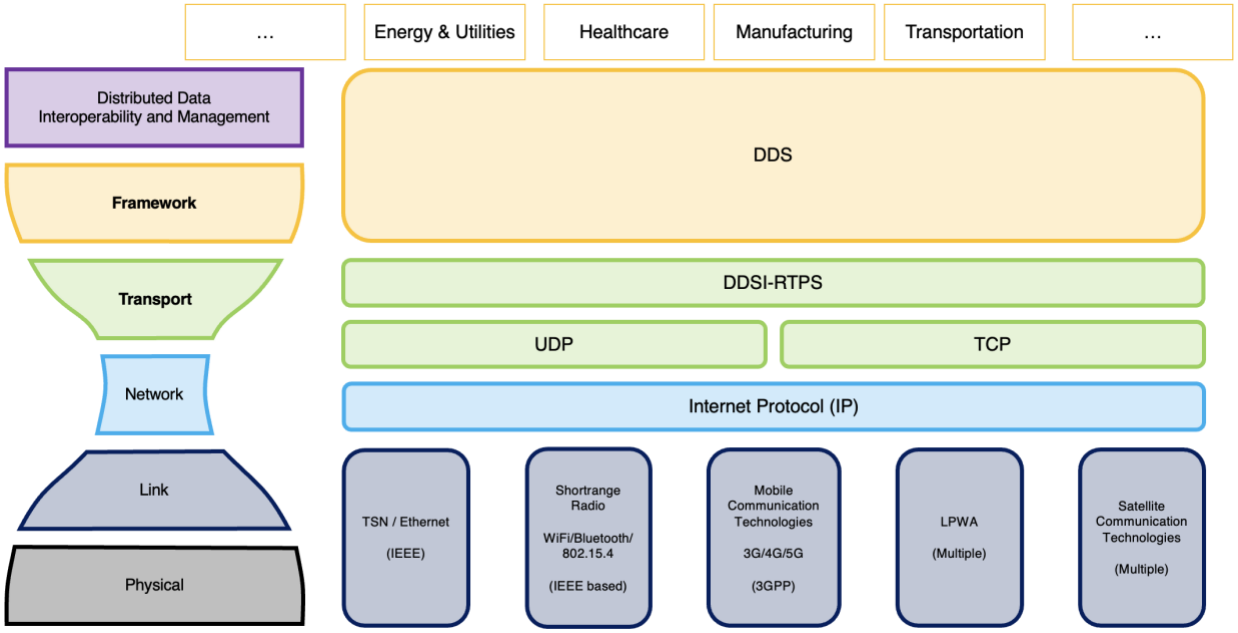


Figure 7-2: DDS IIoT connectivity standard.

DDS is generally used in the control, application, information, operations domains, and sometimes in the business domain (see Figure 1-1). DDS's main purpose is to connect components (devices or gateways or applications) to other components to enable real-time systems and system-of-systems. Components interact with a shared data space, and never directly with each other. Therefore, it is referred to as a *data-centric* middleware standard. It has roots in high-performance defense, industrial and embedded applications.

DDS implements direct component-data-component communication via a relational data model. DDS is also referred to as a *databus* because it is the data-in-motion analog to a database that manages for "data-at-rest". Both a database and a databus implement the "data-centric" abstraction; applications interact with the infrastructure, not directly with each other. The difference is that a database saves *past* data that can later be *searched* by relating properties of the stored data. A *databus* manages *future* data by *filtering* on properties of the incoming data. Data centricity makes a database essential for large storage systems. Data centricity makes a

[31] See [OMG-DDS-DCPS]
[32] See [OMG-DDS-RPC]
[33] See [OMG-DDSI-RTPS]
[34] See [OMG-DDS-XTYPES]
[35] See [OMG-DDS-SECURITY]

databus a fundamental technology for large IIoT software integration and autonomous operation.

The DDS wire protocol is The Real-Time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification (DDSI-RTPS)[36] connectivity transport standard, as shown in Figure 7-2. DDSI-RTPS is independent of the underlying networking technology. DDS implementations support both Wide Area Network (WAN) and Local Area Network (LAN) deployments. Many applications use the UDP transport. DDS can also run over TCP, shared memory, backplane connections, hypervisor networks and many specialized interconnects. DDS does not require any special transport properties such as reliability support. DDS can provide reliable communication over unreliable networks. The DDS-TSN[37] specification defines a set of mechanisms to allow DDS applications to be deployed on, and leverage, Time-Sensitive Networks (TSN). It defines the mapping of the DDSI-RTPS protocol to a TSN network.

Similar to the way a database controls access to stored data, a databus controls data access and updates by many simultaneous components. At its core, DDS is built around a data-centric publish-subscribe data exchange pattern. However, the standard also defines a request-reply data exchange pattern, and vendors offer queuing. The key abstraction is that applications interact with the databus, not directly with other applications participating in that interaction. DDS offers precise data-centric quality-of-service (QoS) control, reliable multicast, configurable delivery, multiple levels of data durability, history, component and network redundancy, automatic discovery, connectivity management and network agnostic fine-grained data-centric security. In addition, one-to-many and many-to-one communications is a key strength. DDS offers powerful ways to filter and select exactly which data goes where, and “where” can be thousands of simultaneous components. To support small, edge devices, there are lightweight versions of DDS that run in constrained environments. The DDS databus ensures ultra-reliable operation and simplifies application code. It does not require servers, greatly easing configuration and operations while eliminating failure and choke points.

A DDS-based system has no hard-coded interactions between components. The DDS databus automatically discovers and connects publishing and subscribing components. No configuration changes are required to add new components (e.g. a smart machine) to a system. Components can be developed or sourced from independent parties. DDS overcomes problems associated with point-to-point system integration, such as lack of scalability, interoperability and the ability to evolve the architecture. It enables plug-and-play simplicity, scalability and exceptionally high real-time performance.

DDS is commonly used for system integration and for building autonomous systems, because of the flexibility, reliability and speed necessary to build complex or real-time applications. DDS is a proven technology for reliable, high performance, large-scale IIoT software systems across many

[36] See [OMG-DDSI-RTPS]

[37] See [OMG-DDS-TSN][OMG-DDSI-RTPS]

vertical industries. IIoT applications using DDS include wind farms, hospital integration, medical imaging, autonomous planes and cars, rail, asset tracking, automotive testing, smart cities, communications, data center switches, video sharing, consumer electronics, oil & gas drilling, ships, avionics, broadcast television, air traffic control, SCADA, robotics and defense.

DDS supports extremely resource-constrained devices with sleep/wake cycles to have access to the shared data space over limited-bandwidth networks. The DDS For Extremely Resource Constrained Environments[38] specification defines an XRCE protocol between a resource constrained, low-powered device (client) and an agent (the server). The XRCE Protocol enables the resource constrained device to communicate with a shared DDS data space and publish and subscribe data.

DDS gateways exist for many other connectivity technologies, including DNP3, C37.118, Modbus, HLA, JMS and so on. The DDS-WEB specification[39] defines a standardized gateway for Web Services. The OPC UA/DDS Gateway[40] specification defines a standard, vendor-independent, configurable gateway that enables interoperability and information exchange between systems that use DDS and systems that use OPC UA. The OPC Foundation is developing an OPC UA-DDS pubsub profile[41] with the goal of adding DDS as an additional publish-subscribe communication option to OPC UA. Work is underway at oneM2M, investigating an interworking gateway between oneM2M and DDS, a DDS protocol binding for oneM2M, and DDS based direct exchange of data between oneM2M entities.[42]

For more details, and to determine the suitability of DDS for a specific set of system requirements, please refer to the assessment template (see Annex A).

7.1.2 OPC FOUNDATION UNIFIED ARCHITECTURE (OPC UA)

OPC Unified Architecture (OPC UA[43]) is a connectivity framework standard used in the manufacturing industry. OPC UA is designed to support multiple transports. Currently transport mappings are defined for TCP with an OPC UA Binary encoding connectivity transport standard or the HTTP connectivity transport, as shown in Figure 7-3. Work on a Web Socket transport mapping has started. All current transport mappings require the TCP transport. The OPC Foundation maintains the OPC UA family of specifications.

[38] See [OMG-DDS-XRCE]
[39] See [OMG-DDS-WEB]
[40] See [OMG-DDS-OPCUA]
[41] See [OPC-DDS]
[42] See [ONEM2M-27]
[43] See [OPC-UA]

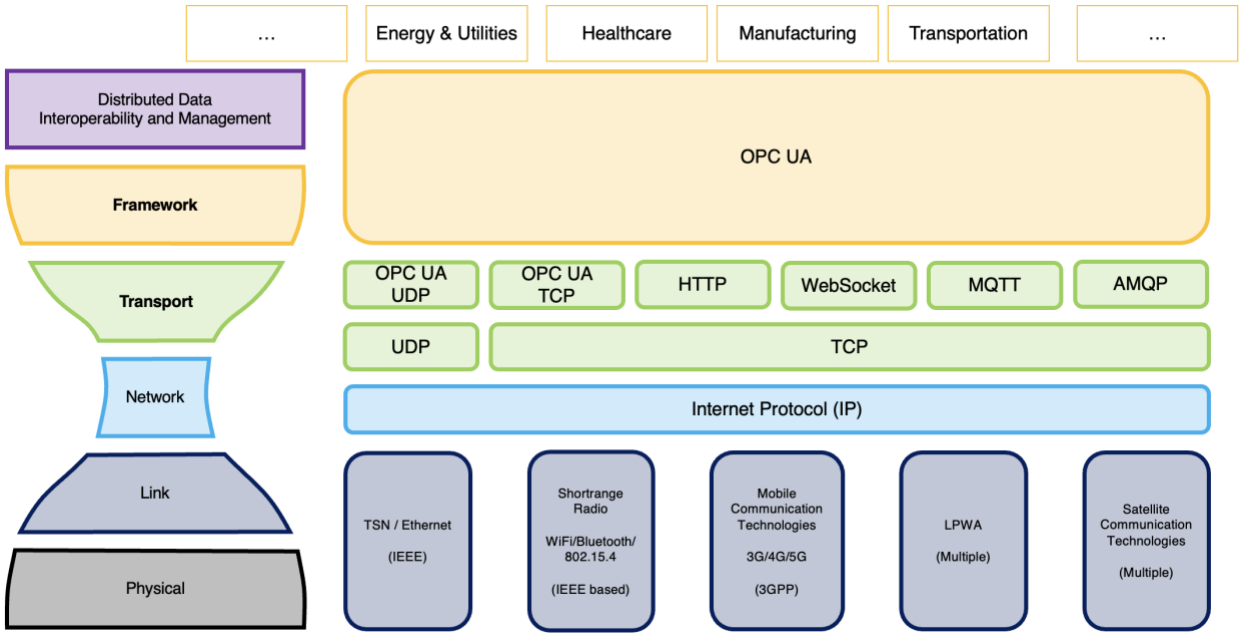


Figure 7-3: OPC UA IIoT connectivity standard.

At its core, OPC UA targets device interoperability and access from Human-Machine Interfaces (HMI), historians, maintenance systems, Manufacturing Execution (MES) systems and Enterprise Resource Planning (ERP) systems. Before OPC UA (or its predecessor OPC), applications simply accessed devices directly through proprietary APIs provided by their vendors. Unfortunately, this meant that applications became dependent on the particular device they controlled. Worse, higher-level applications such as HMIs had no easy way to find, connect to, or control the various devices in factories. OPC UA is generally used in the operations domain and is also being applied to information and application domains (see Figure 1-1).

OPC UA is an evolution of the classic OPC (Object Linking and Embedding for Process Control) standards. It unifies the various original OPC specifications and is an evolution from an API to a network protocol. Adapters are available to bridge between OPC UA and classic OPC. OPC is operational in thousands of factories globally. Traditionally, OPC was used to configure and query plant-floor servers (usually programmable logic controllers (PLCs). Actual device-device communication was then affected via a hardware-based fieldbus[44] such as Modbus or Profinet.

OPC UA retains some of that flavor; it connects and configures plant-floor servers. The UA version adds better syntactical data typing (see section 4.1.3) and semantic information modeling capabilities. There are many companion specifications that define information models for various device types. For example, Field Device Integration (FDI) defines a model that represents all

[44] A "fieldbus" is the name of a family of industrial *computer network* protocols used for real-time distributed control.

fieldbus device types. A remote client such as a graphical interface can browse the device data controlled by a server on the floor. By allowing this introspection across many servers, clients can build a directory with cross-references of all the devices on the floor. Additionally, OPC UA also addresses the specific needs of device-device communication and therefore no longer relies on additional fieldbus solutions. Its scalability allows for implementation on devices with restricted hardware resources, such as sensor and actuator devices.

OPC UA divides system software into clients and servers. The servers usually reside on a device; they provide a way to access the device through a standard “device model”. There are device models for dozens of types of devices from sensor to feedback controllers. Each manufacturer is responsible for providing the server that maps the generic device model to its particular device. The servers expose an object-oriented, remotely callable API that implements the device model.

Generic device models are central to the OPC UA architecture. For example, the object model for a motor starter includes methods for setting parameters, reading data and operating the starter. Thus, applications can control a starter directly without being dependent on the manufacturer’s particular implementation.

OPC UA is developing a “pub-sub” capability. This will provide direct device-to-device connection. There will be several “profiles” using different underlying protocols. The UDP profile supports multicast for efficiency. It targets simple implementation and does not attempt advanced functions like reliability or quality-of-service control. Another profile is designed for connection to cloud-based data analytics. There is work on a DDS profile that will provide more sophisticated pub-sub functionality.

OPC UA targets all kinds of manufacturing, including automotive, oil and gas, pharmaceuticals, food & beverage, medical machines and machine tools. It connects applications at the factory-floor level as well as between the factory-floor and the enterprise IT cloud.

For more details, and to determine the suitability of OPC UA for a specific set of system requirements, please refer to the assessment template in Annex B, Assessment Template: OPC UA.

7.1.3 oneM2M

oneM2M is a relatively new standard (2015) managed by a partnership of regional international standards industry organizations in the telecommunications industry.

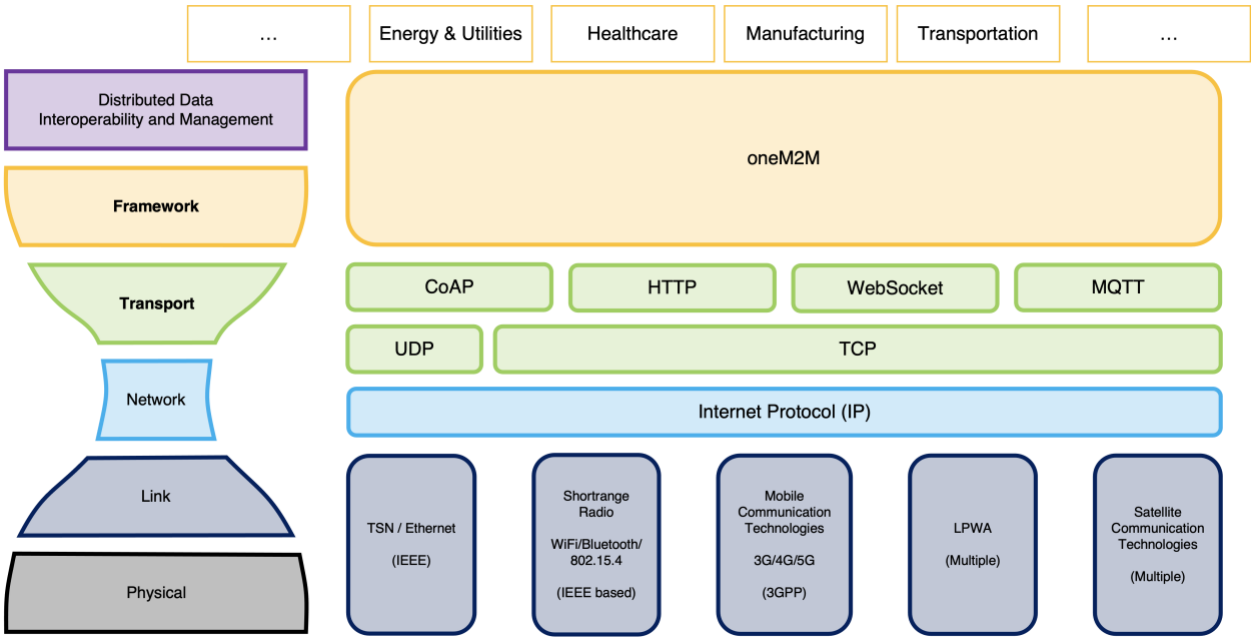


Figure 7-4: oneM2M IIoT connectivity standard.

oneM2M provides a common service layer that sits between applications and connectivity transport. It offers functions that IoT applications across different industry segments commonly need. Those functions are exposed to applications via RESTful APIs.

oneM2M standards comprise a horizontal platform architecture that fits within a three-layer model comprising of applications, middleware services and networks. oneM2M's connectivity standards permit applications that are hosted on connected machines and devices, enterprise systems and mobile devices to communicate with each other in an efficient, secure manner. The oneM2M horizontal platform is scalable as the common service elements are able to be deployed on hosts, at the proximal network edge or within the enterprise cloud.

Connectivity services provide capabilities that allow for efficient communication between application endpoints. It provides interworking mechanisms that adjust the underlying network (e.g. mobile, wireless) quality of service to meet the needs of current application data exchange. Currently, the oneM2M service layer can use HTTP, CoAP, MQTT and WebSockets for connectivity transports. DDS is being explored as another option for providing real-time connectivity between the oneM2M service layer entities. Also, an interworking gateway between oneM2M and DDS and OPC UA is under investigation.

Typical usage of oneM2M includes registration and subscription of devices and applications, service charging and accounting, management of application and devices and monitoring.

oneM2M has commercial deployment pilots for smart city applications. It is suitable for large-scale consumer IoT applications. oneM2M is also actively targeting other IIoT application domains, including telematics and intelligent transportation, home automation, utilities,

healthcare and industrial automation. In all these domains, oneM2M provides semantic enablers (in the scope of the Distributed Data Interoperability and Management layer). As a key part of the telecommunication industry's existing initiatives and its new "5G" initiative, oneM2M provides enablers that focus on the connection between device and the cellular network.

For more details, and to determine the suitability of oneM2M for a specific set of system requirements, please refer to the assessment template in Annex C, Assessment Template: oneM2M.

7.1.4 LIGHTWEIGHT MACHINE-TO-MACHINE (LWM2M)

LwM2M is a framework for remote device management and data transport, developed by Open Mobile Alliance (OMA) SpecWorks, which was formed in 2018 when OMA and the IPSO Alliance merged. LwM2M was designed specifically for battery, CPU and connectivity constrained sensor networks.

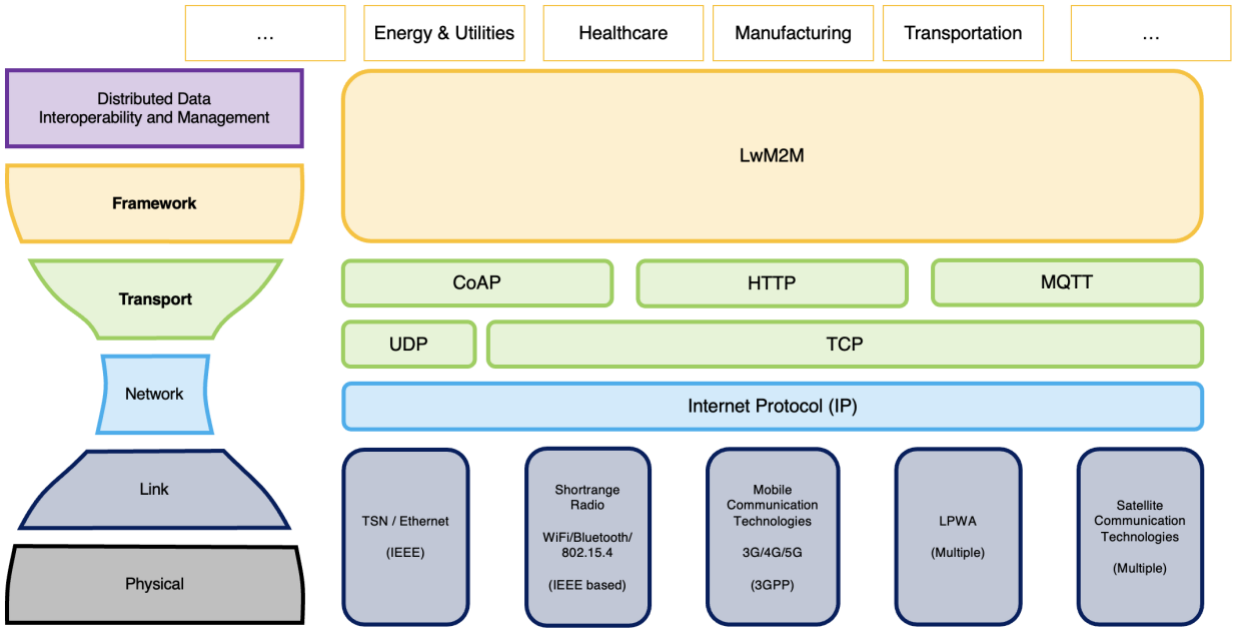


Figure 7-5: LwM2M IIoT connectivity standard.

LwM2M defines two types of nodes: *LwM2M clients* and *LwM2M servers*. A LwM2M client is typically a constrained device (e.g. a sensor) and a LwM2M server is typically a network server that manages such devices.

LwM2M is built on top of the CoAP protocol and inherits the REST design approach and use of open internet standards from CoAP. LwM2M supports both fixed and wireless network technologies and also non-IP-based transport mechanisms (e.g. SMS). Note that a LwM2M Client and server can act as both a CoAP client and a CoAP server. A LwM2M client acts as a CoAP client when it registers itself with the LwM2M server, while the LwM2M server acts as a CoAP client

when it manages, and reads data from, a LwM2M client. LwM2M also uses security mechanisms defined for CoAP and supports both transport layer security (DTLS and TLS) and application layer security (OSCORE).

LwM2M provides device management support in the form of device bootstrapping, device configuration, firmware update, diagnostics, connection management and control. The bootstrapping feature allows a LwM2M Client to fetch its security credentials (from a LwM2M bootstrap server) when it is connected to a network. The control feature allows a LwM2M server to wake up a client using SMS, request a LwM2M client to perform a reboot, disable the client, and request the client to perform a registration.

LwM2M servers read data from clients using the CoAP read and observation features. Using the read feature, the LwM2M server reads a value once. Using the observation feature, the server creates an observation for the resource providing the data, and the LwM2M client will then provide the data value by sending notifications to the server when the state or data of the observed resource changes, until the server cancels the observation. When a LwM2M server has read data from a LwM2M client, it can use that data e.g. to trigger actions on other LwM2M clients.

The resources exposed by LwM2M devices consist of *objects* and *resources*. An object typically represents a feature or function provided by the device. Objects are also used for managing the LwM2M device itself, for example, an object is used to manage the security properties of a LwM2M client. Resources represent values or actions associated with objects. Resources can be specific to a single object or can have semantics that are applicable across multiple different objects and can therefore be reused across objects. Such resources are referred to as *reusable resources*. Different range of resource IDs identifies whether resources are reusable.

Resources are reusable and can be used in multiple objects. Each object definition describes which resources are associated with an object, and whether the support of a specific resource is mandatory or optional when used for that object.

OMA SpecWorks maintains an object and resource registry and provides the procedure for registering new objects. Objects and resources have been registered by OMA. In addition, objects and resources can be registered by third-party SDOs, vendor companies and individual. Vendor companies can also reserve a bulk of object IDs, to be used for private objects that are not registered with the OMA SpecWorks registry. Objects used for management purpose are often referred to as “LwM2M objects”.

7.2 IIoT Connectivity Transport Standards

7.2.1 TCP and UDP over IP

In the context of IIoT, the network layer is the internet protocol (IP). The IP suite also provides the UDP[45] and the TCP[46] transport on top of the IP layer. These IP transports provide the foundation for the other connectivity transports and frameworks.

UDP, Universal Datagram Protocol is a connectionless transport (see section 5.1.4) that provides best-effort delivery quality of service. A message is not resent if it is lost in transmission. Messages may be received out-of-order. Messages are sent as quickly as possible, and so it is suitable for low-latency real-time communications. A message shall be less than 64KB long. A connectivity transport or framework on top of UDP should therefore deal with fragmentation by caching and assembling portions of larger messages.

TCP, Transmission Control Protocol is a connection-oriented (see section 5.1.4) transport that provides reliable and ordered delivery quality of service. A message is resent if it is lost in transmission. Messages are delivered in order. This can lead to head-of-line blocking—high priority, time-critical messages may be blocked behind low priority, non-critical messages. Retries hold up all messages in the channel. Thus, message latencies can vary greatly, leading to large jitter, especially when messages are lost in transmission. The connection sequence can be expensive in time and resources. There is no inherent limit on the message size.

The choice between UDP and TCP at the connectivity transport level has significant implications for the connectivity framework and its suitability (see Figure 1-1). As shown in Figure 7-2, Figure 7-3, Figure 7-4, and Figure 7-5, some connectivity frameworks require TCP and inherit its characteristics; some require UDP and inherit those characteristics; and some can use either to support the varying application requirements.

7.2.2 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) is a connectivity transport standard inspired by HTTP but designed to be more lightweight and efficient (see Figure 7-4, Figure 7-5). It was established using the UDP transport. The IETF maintains the CoAP open standard specification. Since its definition alternative transports using TCP with TLS,[47] SMS, and Web Sockets have been developed.

CoAP is generally used in the operations domain (see Figure 1-1). Like HTTP, a client sends a request to a server, specifying a data object, an operation, and a payload. The server replies with failure or success and a response payload. In addition, a client can also register to be notified of any changes in data object. Unlike HTTP, it is suitable for device-to-device queries. However,

[45] See [IETF-RFC768]
[46] See [IETF-RFC793]
[47] See [IETF-RFC4279]

retries and reordering are implemented in the application stack. CoAP is designed to interoperate with HTTP and the RESTful web services through simple proxies.

For more details and to determine the suitability of CoAP for a specific set of system requirements, please refer to the assessment template in Annex E, Assessment Template: CoAP.

7.2.3 MQTT (FORMERLY MQ TELEMETRY TRANSPORT)

MQTT is an open connectivity transport standard, maintained by OASIS. It requires the TCP transport.

MQTT is generally used in the information domain (Figure 1-1). It targets device data collection. As the name indicates, the main purpose is telemetry or remote monitoring. Its goal is to collect data from many devices and transport that data to the IT infrastructure. It targets large networks of small devices that need to be monitored or controlled from the cloud (Figure 7-3, Figure 7-4, Figure 7-5).

MQTT implements a hub-and-spoke architecture. Typically, all the devices connect to a data concentrator server. The protocol generally works on top of TCP, which provides a simple, reliable stream. Since the IT infrastructure uses the data, the entire system is designed to transport data easily into enterprise technologies. MQTT has also been adapted for UDP in a separate protocol called MQTT-SN.

MQTT is suited for many-to-one data collection. It is not commonly used for device-to-device transfer or for one-to-many data distribution. MQTT is a simple protocol with few control options. Most applications don't need to be particularly fast; latency specifications are often measured in seconds.

MQTT targets applications such as monitoring an oil pipeline for leaks or vandalism, that require message feeds from thousands of sensors to be concentrated into a single location for analysis. When the system finds a problem, it can take action to correct that problem. Other applications for MQTT include power usage monitoring, lighting control and even intelligent gardening and agriculture. They share a need for collecting data from many sources and making them available to the IT infrastructure.

For more details and to determine the suitability of MQTT for a specific set of system requirements, please refer to the assessment template in Annex F, Assessment Template: MQTT.

7.2.4 HYPERTEXT TRANSFER PROTOCOL (HTTP)

7.2.4.1 RESTFUL WEBSERVICES

Web services (Wikipedia)48 using Hypertext Transfer Protocol (HTTP) refers to the application-specific connectivity frameworks, primarily devised for human user interaction interfaces. They rely on a RESTful49 style of architecture (Fielding, Wikipedia50) using the HTTP connectivity transport standard to exchange textual data, as shown in Figure 7-3, Figure 7-4, Figure 7-5. It requires the TCP transport. The IETF[51] maintains the HTTP open standard specification; the W3C[52] maintains the web (HTML5) specifications.

Web services using HTTP are generally used in the application domain (see Figure 1-1). Data is represented in textual form (either as JSON or XML) and embedded in a hypermedia (HTML) context. A Uniform Resource Identifier (URI) represents a data object on a server. A client (app) sends a request to a web server, specifying a data object URI, an operation and a payload. The server replies with failure or success and a response payload. The communication is text-based and designed for human speeds. It is not efficient for device-to-device communications and not suitable for real-time communications.

For more details, and to determine the suitability of web services using HTTP for a specific set of system requirements, please refer to the assessment template in Annex D, Assessment Template: HTTP.

7.3 FIELDBUS TECHNOLOGIES

Fieldbus ecosystems are well developed and extensively deployed in many industries. Most originated with special-purpose hardware and protocols. Well-known fieldbuses include Profibus (Profinet), Ethernet/IP, Modbus & Modbus/TCP, HART & HART wireless, and the Foundation Fieldbus family. Each has developed extensive ecosystems of vendors and customers.

The industrial internet will bring benefits of common connectivity standards based on the Internet Protocol (IP). This is a significant transition; today's industrial ecosystems use a wide variety of communication and connectivity standards.

Interoperability between fieldbus variants is, in general, poor. Many of these have been adopting IP-based networking models and Ethernet transports. This is improving technical interoperability. Syntactic or higher levels of interoperability are only available with special point solutions.

[48] See [W3C-WSA], for overview [WKPD-WS]
[49] RESTful means to adhere to a REST communications architecture style.
[50] See [Fielding-2000], for overview [WKPD-REST]
[51] See [IETF]
[52] See [W3C]

Fieldbuses implement parts of the connectivity transport and framework functions. None satisfies all of the connectivity core standards criteria (see section 3.4). However, they are well deployed with extensive experience. Most of these protocols support the following types of communication:

- *Management*: Request-reply pattern, with explicit schemas, for example a RESTful architectural style, used for resource lifecycle communication for management and status of the device/thing.
- *Operational*: Peer-to-peer publish-subscribe pattern, with implicit schemas, often time-series, used for data streaming of key operational data used for sense-control-actuate control loop processing that maintains the system's operational integrity.

This suggests that the core connectivity standards should support both forms of data exchange patterns to support the range of functions in the overlying applications.

Most automation and control applications in operations and being deployed for the foreseeable future will rely upon a Fieldbus-based ecosystem protocol. Most IIoT applications will have to access data from these protocols to acquire data from and actuate decisions or insights into these protocols. Integration is important.

8 CORE CONNECTIVITY STANDARDS

The assessment of the connectivity standards listed in chapter 7 confirms that not all of the connectivity standards need to support the applications across IIoT to the same degree. Some are more suited to one functional domain (see Figure 1-1), while others are applicable across multiple functional domains. Some are vertically focused, specific to certain industries, while others are horizontally focused, and used in multiple industries.

The result of applying the connectivity core standards criteria (see section 3.4) to the IIoT connectivity framework standards identified in chapter 7 is summarized in Table 8-1.

	Core Standard Criterion	DDS	OPC UA	oneM2M	LwM2M
1	Provide **syntactic interoperability**#	✓	✓	✓	✓
2	Open standard with strong **independent, international** governance#	✓	✓	✓	✓
3	**Horizontal** and neutral in its applicability across industries#	✓	✓	✓	✓
4	**Stable** and **deployed** across multiple vertical industries#	Software Integration & Autonomy	Manufacturing	Smart City Pilots*	Remote low power device management
5	Have **standards-defined *Core Gateways*** to all other core connectivity standards#	HTTP, OPC UA, oneM2M*	HTTP, DDS, oneM2M*	HTTP, OPC UA*, DDS*, LwM2M	oneM2M, HTTP*
6	Meet the connectivity framework **functional** requirements	✓	✓	✓	✓
7	Meet **non-functional** requirements of performance, scalability, reliability, resilience	✓	Real-time via TSN	Reports not yet documented or public	Implementation dependent
8	Meet **security** and **safety** requirements	✓	✓	✓	✓
9	Not require any single component from any single vendor	✓	✓	✓	✓
10	Have readily available SDKs both **commercial** and **open source**	✓	✓	✓	✓

green = Gating Criteria * = work in progress, ✓= supported, ✗ = **not** supported

Table 8-1: IIoT connectivity core standards criteria applied to key connectivity framework standards.

DDS is managed by the OMG, has an established standard gateway mapping to web services (and many others), was established in 2004 and is widely deployed in many types of systems in multiple industries. Standardized gateway mappings to oneM2M are in development. DDS is suitable for multiple functional domains (see Figure 1-1). It is widely used in the control domain, and increasingly being used in the information, operation and business domains. Security is a recent addition to the specification, now nearing deployment. For more details on the applicability of DDS see the detailed assessment template mapping in Annex A.

Web Services using HTTP are important in any IIoT system, especially from the IT perspective. HTTP is managed by the IETF, has standard mappings to DDS, OPC UA and oneM2M, and is well established and widely deployed across multiple industries. Web services are suitable for the application and business domains. Web services are typically used for administrative communication on many IIoT devices, but those devices generally use other mechanisms for the operational communication. For more details on the applicability of HTTP see the detailed assessment template mapping in Annex D, Assessment Template: HTTP.

OPC UA is managed by the OPC Foundation and has a standard mapping to web services. A standard gateway mapping of OPC UA is available from the OMG, a standard mapping of OPC UA on top of DDS is under way at the OPC Foundation, and a oneM2M mapping is under development at oneM2M. OPC UA is being deployed. The main adoption being manufacturing. The precursor technology ("Classic OPC") is widespread. Because of that we consider the risk of OPC UA deployment to be low enough in manufacturing to qualify. Other related industries are beginning to deploy OPC UA as well, so its applicability will likely expand in the future. OPC UA is suitable for the operation and application domains. For more details on the applicability of OPC UA see the detailed assessment template mapping in Annex B, Assessment Template: OPC UA.

oneM2M is managed by a partnership of regional international standards organizations to include: ARIB, ATIS, CCSA, ETSI, TIA, TSDSI, TTA, TTC, uses a RESTful style architecture with transport bindings for HTTP and Web Sockets and is scheduled to have established standard mappings to DDS and OPC UA in the next release, and has been recently deployed across multiple industries. As the standard is quite new (released 2015), deployments are also new and less than what one might require for it to be considered stable and proven across multiple industries. For more details on the applicability of oneM2M see the detailed assessment template mapping in Annex C, Assessment Template: oneM2M.

Considering the current state-of-the-art, the criteria for IIoT connectivity core standards, the practicality of maintaining a minimal set of core standards that support many connectivity technologies, and the need to provide effective guidance. The following emerge as potential IIoT connectivity core standards: DDS, OPC UA, Web-Services and oneM2M. These standards span aspects of IIoT systems, as shown in Table 8-2. The set of potential connectivity core standards may expand in the future as other applicable standards mature to meet the criteria and address IIoT system aspects. Table 8-2 shows examples of some applications where these standards are often applied.

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 need to integrate components written by different programmers or 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 and stop. 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 using HTTP
Information & Communications 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 IIoT connectivity standards.

Core gateways enable horizontal data interoperability between components across functional domains, as shown in Figure 3-4. Other connectivity technologies can integrate into the system architecture using a gateway to one of the core connectivity standards. This satisfies the range of IIoT system architecture challenges with minimum complexity.

9 OTHER CONNECTIVITY TECHNOLOGIES

Historically, specialized industrial connectivity technologies (see section 7.3) have evolved to meet the specific needs of a particular application area. The goal of the IIoT connectivity reference architecture (see chapter 3) is to enable endpoints using one connectivity technology to communicate with endpoints using another unrelated connectivity technology, possibly in a different functional domain. Since gateways exist between the core connectivity standards, endpoints from the originally unrelated technologies can now communicate.

A domain-specific connectivity technology needs to provide a gateway to only one of the core standards. However, the choice of the core connectivity standard has a direct impact on the fidelity and the quality of service of the communication, as the core connectivity standards vary widely in their characteristics (see chapter 7). The most suitable core connectivity standard should be selected for the gateway. We recommend filling out the assessment template defined in chapter 6 for the specific technology under consideration, and then picking out the core connectivity standard that is most aligned with the connectivity technology under consideration.

Some guidelines follow, based on the primary functional domain (see Figure 1-1) of applicability for the connectivity technology.

Control domain connectivity technologies will support high reliability, fast real-time performance, scaling to large number of data objects, and rich quality of service.

Operations domain connectivity technologies will support monitoring and management of devices and applications.

Information domain connectivity technologies will support moving large volume and variety of real-time data selectively to feed streaming analytics and real-time decision processes.

Application domain connectivity technologies will support external APIs and user interfaces, including web browsers and mobile handhelds.

Business domain connectivity technologies will support traditional IT applications and data centers.

These guidelines are starting points, and do not substitute for filling out the assessment template (see chapter 6 to select the closest core connectivity standard.

With the gateways to the core connectivity standards in place for the connectivity technologies of interest, the IIoT connectivity architecture enables communication between hitherto isolated endpoints. It can open up new value streams and help realize the full potential of IIoT.

Annex A ASSESSMENT TEMPLATE: DDS

This annex contains the assessment template for the Data Distribution Service (DDS).

A.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.* DDS (Data Distribution Service)
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* Object Management Group (OMG)
Description	*Short synopsis of the technology*. DDS is a connectivity framework for industrial IoT. It enables network interoperability for connected machines, enterprise systems and mobile devices. It provides scalability, performance, and Quality-of-Service required to support IoT applications. DDS can be deployed in platforms including low footprint extremely resource constrained devices, microcontrollers, laptops, desktops, servers, high performance compute, virtualized and cloud native infrastructure. DDS supports efficient bandwidth usage as well as agile orchestration of system components. It provides a global data space for analytics and enables flexible real-time system integration.
Application Domain(s)	*Application domains targeted by the connectivity technology.* DDS is suitable for both IIoT and large-scale consumer IoT applications. DDS specifically targets the IIoT application domains, including autonomous systems, automotive, robotics, transportation, energy, healthcare, industrial automation, smart manufacturing, simulation & test, smart cities, agriculture, military and aerospace.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* Connectivity transport options include: • UDP (optional: DTLS) • TCP (optional: TLS) in progress • DDS-TSN (Time Sensitive Networking) in progress • Shared memory • Custom
References	Website[53] *and other useful links to the technology.* • DDS Foundation • DDS Specifications • DDS Guide

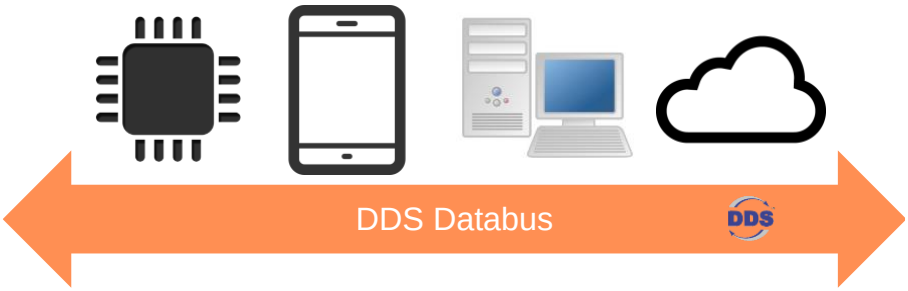
[53] See [OMG-DDS], [OMG-DDS-SPECS], [OMG-DDS-GUIDE]

A.6.2 BUSINESS VIEWPOINT

A.6.2.1 Purpose
(Section 6.2.1)

Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.

The OMG DDS standard helps users reliably and securely harness ever-increasing amounts of device generated data while processing the data in real-time and acting on events as quickly as they occur. As a result, it enables smarter decisions, new services, additional revenue streams and reduced costs. The OMG DDS middleware standard can also simplify the development, deployment and management of IoT applications, speeding time-to-market.



According to the OMG DDS Foundation Portal,[54] based on the use of DDS in thousands of applications, one can predict the need for DDS in new projects. If you answer *yes* to most of the following questions, DDS is likely to be your go-to solution.

- Are the consequences of short downtime severe? If your system goes offline for 5 minutes (or even 5 milliseconds), is it a serious problem? Since DDS does not require servers that could fail and supports redundancy, it makes “fast” reliability and availability much easier. DDS also eliminates struggles with server configuration, startup order or failover to backup servers.
- Do you require sub-second response? DDS direct peer-to-peer messaging can deliver in milliseconds or even a few microseconds.
- Do you have more than a few software modules or software teams? The databus abstraction will define the interfaces, capture the dataflow, enable module evolution and enforce interoperation between teams.
- Do you have to supply data to many modules, or have too much data to send it all to one place? DDS selective filtering makes it easy to find and deliver the right data.
- Are you building a new IIoT architecture? DDS is not usually used in “retrofit” applications. Typical systems are software projects take more than a year to write, last more than three years or go through multiple versions.

These questions help identify your critical performance, reliability, and integration needs. If you answer *yes* to any of these questions, you should evaluate DDS as a solution, since it offers many additional benefits.

[54] See [OMG-DDS]

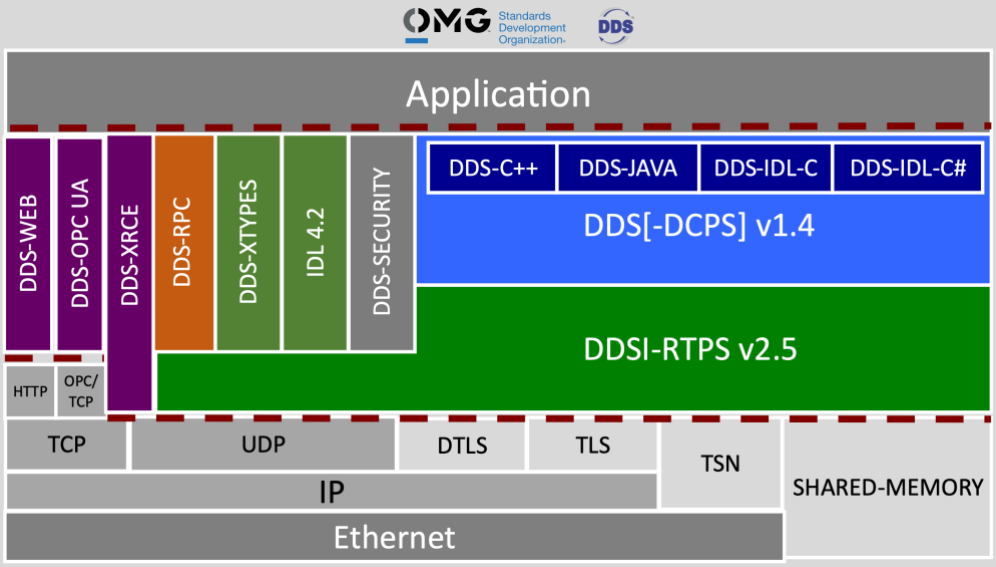
A.6.2 BUSINESS VIEWPOINT

A.6.2.2 Pedigree
(Section 6.2.2)

Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.

DDS was designed for real-time, scalable, continuously available, peer-to-peer real-world systems. A few proprietary DDS systems had been available for several years. Starting in 2001, there was a focused industry effort to create an open standard under the auspices of the OMG, resulting in Version 1.0 in 2004. Since then, DDS has grown into a family of specifications. The essential specifications include:

- DDS v1.4 (2015): defines a Data-Centric Publish-Subscribe (DCPS) model for distributed application communication and integration.
- DDSI-RTPS v2.5 (2021): defines the Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol.
- DDS-XTypes v1.3 (2020): defines Extensible and DynamicTopic Data Types for DDS
- DDS-RPC v1.0 (2017): defines a distributed services framework providing language-independent service definition and service/remote procedure invocation using DDS. Supports automatic discovery, synchronous and asynchronous invocations, and QoS.
- DDS-Security v1.1 (2018): defines the Security Model and Service Plugin Interface (SPI) architecture for compliant DDS implementations.



DDS enjoys an active and vibrant community continuously working to extend its applicability. The full list of the DDS family of specifications can be found at the OMG DDS Foundation website.[55]

Multiple independent DDS implementations are available, including both open-source and commercial.

[55] See [OMG-DDS-SPECS], in particular: [OMG-DDS-DCPS], [OMG-DDSI-RTPS], [OMG-DDS-XTYPES], [OMG-DDS-RPC], [OMG-DDS-SECURITY]

A.6.2 BUSINESS VIEWPOINT	
A.6.2.3 Variants *(Section 6.2.3)*	*Describe the options and variants from the original generic description of the technology.* Compliance profiles are defined by the family of DDS standards. Implementations may differ in their support and coverage of the DDS specifications or compliance profiles.
A.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* The core DDS specifications are mature and have been refined through thousands of applications across multiple industries. DDS vendors collaborate and regularly hold plug-fests to show interoperability between independent DDS implementations. DDS has been applied in multiple verticals[56] to realize higher domain-specific interoperable open architecture specifications. These include: • *AUTOSAR Adaptive* standard for automotive application development uses DDS as a network binding since the 18.03 release. • *ROS: Robot Operating System* v2.0 open-source application development. environment for robotics is built on top of the DDS connectivity framework. • *Generic Vehicle Architecture (GVA)*. The UK MoD GVA (DEF STAN 23-09) requires DDS. • *NATO Generic Vehicle Architecture (NGVA).* The NATO version of the GVA (NGVA) standard (*STANAG 4754*) requires DDS. • *Future Airborne Capability Environment (FACE).* DDS is part of all FACE certified conformant Services Segment (TSS). • *Common Image Generator Interface (CIGI)*. CIGI Next will be built on DDS as the interface. • OpenFMB NAESB Standard for the Smart Grid uses CIM extensions over DDS. • *Tactical Microgrid Standards Consortium (TMSC).* TMSC requires DDS for communication. • *MDPnP OpenICE* Integrated Clinical Environment for Medical Device Interoperability. • *EUROCAE ED-133* flight data exchange between air traffic control centers. • *Open Architecture Radar Interface Standard (OARIS).* • SAE *Unmanned* Systems (UxS) Control Segment (UCS) Architecture. The UCS Architecture supports DDS. Most UCS implementations use DDS. • *Joint Architecture for Unmanned Systems (JAUS) over DDS.* • *Layered Simulation Architecture.* • *Navy Open Architecture.* • *Application Management and Systems Monitoring (AMSM)*. DDS is supported as a platform specific model. • *ALert MAnagement Service (ALAMS)*. DDS is supported as a platform specific model.

[56] See [OMG-DDS-VERTICALS]

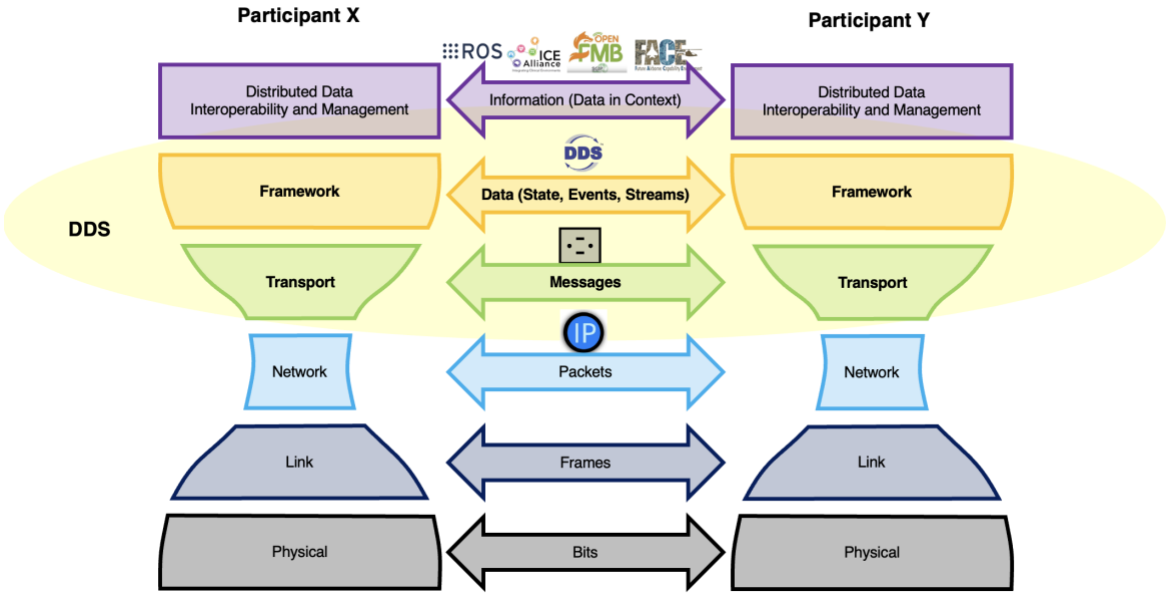
A.6.2 BUSINESS VIEWPOINT	
A.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* DDS is a family of stable and proven connectivity framework standards with increasing adoption across a spectrum of industries. It has been used since 2004 (not counting precursors) across multiple industries. The DDS specifications have been continuously updated to incorporate the lessons from deployments. The core specifications are stable, and easy to use for professionals. The DDS community has continued to innovate actively and expand the breadth and depth of specifications across all aspects of IIoT data connectivity middleware.
A.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* Object Management Group (OMG)
A.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* DDS is an open standard. The specifications are openly available to anyone at no cost. Anyone is free to download and implement them. The specifications development process is open to participation by both vendors and users. Open source and commercial implementations are available. The DDS specifications do not rely on any single component from any single vendor.

A.6.3 USAGE VIEWPOINT

A.6.3.1 Architecture *(Section 6.3.1)*

Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.

DDS provides a "middleware" software layer that abstracts an application from the details of the operating system, network transport and low-level data formats. An application links to a DDS middleware library to participate in a data exchange. The same concepts and APIs are provided in different programming languages allowing applications to exchange data across of operating systems, languages and processor architectures. Low-level details like data wire format, discovery, connections, reliability, timing and QoS management are managed by the middleware layer. It integrates the components of a system together, providing low-latency data connectivity, extreme reliability and a scalable architecture that business and mission-critical IoT applications need.



DDS organizes the data exchange between applications as shared data spaces, called *domains*. An application can participate in one or more DDS domains. A DDS-DomainParticipant represents an application's participation in a data space (domain). Within that data space (domain), a collection of data objects with the same structure (data type) and behavior (QoS) is represented by a named DDS-Topic. A DDS-DataWriter is used to publish updates to one or more data objects on a DDS-Topic. A DDS-DataReader is used to subscribe to updates to one or more data objects on a DDS-Topic.

When a DDS-DataWriter updates data objects on a DDS-Topic, those updates are shared directly with all the DDS-DataReaders associated with that DDS-Topic. The updates are cached locally in each DataReader and DataWriter, in accordance with the QoS configuration. The data paths are direct and peer-to-peer, with no server or broker in the middle. An application can participate in one or more data spaces. A content filter can be used to specify the subset of updates of interest to a DataReader. Only the relevant updates are delivered, as defined by the content filter and the QoS configuration specific to the DDS-DataWriter and DDS-DataReader pair. Applications can retrieve the locally cached updates on a DataReader through a variety of mechanisms, including waiting for updates, listening for status changes or by polling.

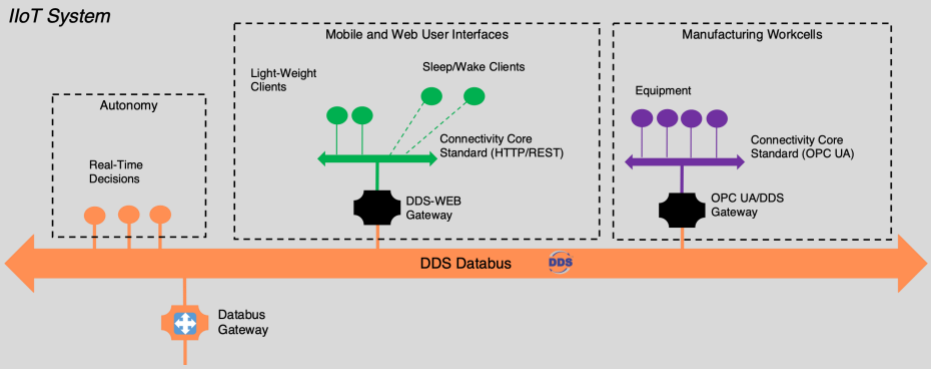
A.6.3.2 Technology Options *(Section 6.3.2)*

List the choices to be made for using the connectivity technology in a system.

Selection of the DDS implementation. Note that the applications participating in a data exchange may use different implementations.

A.6.3 USAGE VIEWPOINT	
A.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* DDS is already proven in mission-critical systems in industries ranging from smart transportation to healthcare to smart energy, and also aerospace & defense. Reasons for using DDS include: • *Ease of integration*: The data-centric approach used by DDS allows the definition of common and extensible data models for seamless Information Technology (IT)/Operational Technology (OT) interoperability. Its anonymous data-sharing abstraction completely hides connectivity and topology details from applications and allows a system to be assembled from independently developed loosely coupled components. • *Performance efficiency and scalability*: DDS implementations can achieve point-to-point latencies that are as low as 30 µsec. and throughput of several million messages per second. It uses an efficient wire protocol, content- and time-based filtering. When properly architected, DDS-based systems can achieve near-linear scalability. • *Advanced security*: The OMG DDS Security Specification defines a comprehensive Security Model and Service Plugin Interface (SPI) architecture for compliant DDS implementations. DDS provides standardized authentication, encryption, access control and logging capabilities to enable secure data connectivity end-to-end in an IoT system. • *QoS-enabled*: A rich set of QoS policies allows DDS to control of all aspects of data distribution, such as timeliness, traffic prioritization, reliability and resource usage. • *Scalable discovery*: For large-scale dynamic systems, DDS offers automatic discovery that provides plug-and-play functionality to simplify system integration and orchestration. • *Applicability*: DDS can transparently address peer-to-peer, device-to-device, device-to-cloud and cloud-to-cloud communication. Implementations are available for embedded, mobile, web, enterprise and cloud applications. • *Future proof*: The OMG DDS middleware specification enables end-to-end vendor interoperability and eases IoT system development and integration through fully open, future-proof APIs with no vendor lock in.
A.6.3.4 Typical Usage *(Section 6.3.4)*	*What function or where in the system this technology is typically used?* • Data plane (data collection, data distribution, data streaming) • Control plane (commands & status, orchestration) • Monitoring plane (remote monitoring & diagnostics) • Management plane (events, analytics & alarms, user interfaces)
A.6.3.5 Operations *(Section 6.3.5)*	*Can one monitor, manage and dynamically replace elements of the communication function?* One can monitor and manage a dataspace simply by adding an application component to participate in dataspace and subscribe to the built-in discovery topics and other monitoring data topics of interest. One can also simply replace a component in the data space by another with an equivalent functional interface and expose additional state for systems level monitoring and management. Implementations are interoperable, and monitoring nodes can easily be added/updated/replaced. Leading implementations provide the ability to replace certain elements of the communication function, such as discovery, networking, and security elements with customized implementations.

A.6.3 USAGE VIEWPOINT

A.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* The DDS-SECURITY[57] specification defines a fine-grained security model at the level of data objects, including per DDS-Topic authentication, encryption, access control, data integrity and logging capabilities to enable secure end-to-end data flows in an IoT system. The DDS security model applied on top of the network layer, and can therefore support secure multicast, when needed. Different security policies may be applied to different topics to match the system security requirements for different data flows. In addition to DDS-SECURITY, DDS the transport layer security mechanisms such as TLS (Transport Layer Security) and DTLS (Datagram Transport Layer Security) can also be used, although they may be unnecessary when the DDS-SECURITY model is used.
A.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* Certifiable DDS implementations are available, including DO-178C Level A for flight safety critical systems, IEC 60601 for class 3 medical devices and ISO 26262 for automotive road functional safety. HIPAA-compliant security is available for the medical industry.
A.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* Standardized gateways are available to the following connectivity standards: • DDS-WEB[58]—access to the data space via RESTful or HTTP technologies • OPC UA/DDS Gateway[59]—access to the DDS data space via OPC UA, and access to data objects on the OPC UA address space via DDS • oneM2M-DDS gateway is under development  Bespoke gateways to many IIoT vertical specific connectivity technologies exist, including DNP3, C37.118, Modbus, HLA, JMS and so on.

[57] See [OMG-DDS-SECURITY]
[58] See [OMG-DDS-WEB]
[59] See [OMG-DDS-OPCUA]

A.6.4 FUNCTIONAL VIEWPOINT

A.6.4.1 Core Framework Layer Functions

Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* DDS provides a flexible resource model. User-defined data types define the structure of data objects. A DDS-Topic is a named collection of data objects that all have the same data type. Topics are user defined, and there can be any number of them.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* DDS provides a flexible user defined scheme for identifying and addressing data objects. A user can mark certain fields of the data type as "key" fields—those are used to identify and address the data objects with a DDS-Topic. A data object is uniquely addressed by a domain id (which identifies the data space), the topic name within the data space and the key fields within that topic.
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* DDS provides a flexible data type system, analogous to that of modern programming languages such as C, C++, Java and .NET. The DDS-XTYPES specification defines extensible and mutable data types that can evolve over a system's lifecycle. Data types are defined in a programming-language-neutral manner, using the OMG Interface Definition Language (IDL[60]) or equivalent XML standards.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data object's lifecycle? Summarize the salient aspects.* DDS provides a means to manage the full data-object lifecycle, including operations to create, read, update and delete data objects.
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* DDS provides a rich set of QoS policies to manage the recent history of data objects. This includes caching user-defined history of data objects independently at DDS-DataReaders and DDS-DataWriters, delivering historical data for late joiners and caching the state in the data space, using the DDS-Persistence Service (an optional part of the DDS specification).
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* The fundamental interaction pattern is publish/subscribe. DDS-DataWriters are used to publish update to data objects; DDS-DataReaders are used to subscribe to data-object updates.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* DDS participants may request the state of data object via a request topic and receive response(s) via a reply topic from one or more participants. The DDS request-reply interaction pattern can be used in conjunction with the publish-subscribe pattern. An application can receive or more replies, receive continuous replies, and get replies from multiple repliers, correlated to the original request. The DDS-RPC specification formalizes the design pattern and defines a distributed services framework providing language-independent service definition and service/remote procedure invocation using DDS. It supports automatic discovery, synchronous and asynchronous invocations and QoS.

A.6.4 FUNCTIONAL VIEWPOINT	
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* DDS provides a means to discover the DDS-DomainParticipants, DDS-DataWriters, DDS-DataReaders and the DDS-Topics in a data space. Discovery is automatic and continuous as applications come and go. An application can access the discovery data simply by subscribing to pre-defined built-in discovery topics.
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* DDS provides a rich set of exception handling capabilities. The data delivery contracts are defined via QoS policies, and when any QoS policy is violated, a corresponding flag is raised to signal the exception to the application. Specifically, the LIVELINESS QoS policy is used to monitor the connectivity. Upon loss of connectivity, an exception is signaled by setting the corresponding flag and by changing the state of the disconnected data objects.
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* The DDS specification defines a rich set of 21+ data QoS policies. These include data delivery (best-efforts vs. reliable), timeliness (deadlines), ordering, durability, lifespan, fault tolerance, history, liveliness, ownership, latency, priority and so on.
Data Security *(Section 4.1.11)*	*Does it provide a data object security model? Summarize the salient aspects.* DDS provides a fine-grained data object security model. It is detailed in the DDS-SECURITY[61] specification. It defines per-topic security policies for authentication, access control (read, write, read-write), confidentiality (encryption), data integrity, data tagging and logging when violations occur. The security policies are applied on top of the data-object resource model and can be activated or updated by reconfiguration (i.e. no code changes) at any stage of the development or deployment cycle.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* DDS provides a standardized APIs in multiple programming languages: • *ISO/IEC C++ 2003 Language PSM for DDS*: defines a C++ API only for the DDS specification. • *Java 5 Language PSM for DDS*: defines a Java API for the DDS specification. • *Other language APIs* for C, Java, traditional C++ and other languages are derived from the DDS API in IDL using the respective IDL to language mappings. • APIs are available for popular languages including Python, JavaScript, Lua, Go and Rust.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* DDS provides standardized APIs to configure the data types, QoS policies, security, resource management, and timing. It also standardizes the APIs to monitor QoS policy violations. Implementations may also support file-based mechanisms for configuration and administration. Monitoring of the endpoint internal state is implementation specific and can be exposed and discovered via standardized DDS APIs.

[60] See [OMG-IDL]

[61] See [OMG-DDS-SECURITY]

A.6.4 FUNCTIONAL VIEWPOINT

A.6.4.2 Core Transport Layer Functions

Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* The DDS messaging protocol (DDSI-RTPS[62]) assumes connectionless (UDP) messaging. The messaging protocol can allow messages larger than the UDP (64kB) limit. The protocol is designed to support widely varying message sizes. DDS implementations can also support TCP. There is an ongoing effort to standardize the mapping under the auspices of the TCP DDSI-RTPS specification.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* DDS supports both unicast (default) and multicast (when available).
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* DDS transport endpoints correspond to a DDS-DataWriter and DDS-DataReader. The endpoints are addressed using a globally unique ID (GUID) for the endpoints. The number of endpoints within a domain is bounded by the number of unique GUIDs.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* DDS does not require a connected circuit between the endpoints.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* DDS provides a means to prioritize messages, for example using the TRANSPORT_PRIORITY QoS policy.
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* DDS does not provide time synchronization services. In systems using DDS, time synchronization is typically accomplished using a separate time synchronization protocol. If Ethernet Time Sensitive Networking (TSN) is used, the timing and synchronization services provided by TSN can be used. The DDS-TSN[63] specification standardizes the mapping and configuration of DDS topics to TSN flows. While DDS does not require a time synchronized network for its operation, it does track the source and reception timestamps on every update, so that applications can take advantage of time synchronization when it is available. DDS also provides APIs for applications to provide their own timestamp and supports timestamp-based ordering, that becomes meaningful in a time synchronized network.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* DDS provides mechanisms for message security. It provides support for authentication of endpoints, message encryption and message integrity. In addition, when using a transport protocol that provides content security (e.g. TLS or DTLS), those mechanisms can also be used; they not required.

[62] See [OMG-DDSI-RTPS]
[63] See [OMG-DDS-TSN]

A.6.5 IMPLEMENTATION VIEWPOINT

A.6.5.1 System Architecture Considerations

Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* The DDS standards are designed to be implemented without a broker, in a peer-to-peer manner. An application, by linking to a DDS library becomes an active participant in the data exchange. Like a networking library, there are no other process dependencies.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints in order to participate in information exchange?* The application code (or business logic) does not have to be aware of other endpoints to participate in a data exchange. Applications interact directly with the databus (data objects organized into DDS-Topics) and never directly with each other.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* Governance can be explicit and shareable, independent of the application business logic. DDS allows system architects to choose the style of governance. The data types are always explicitly defined, the data flows and the quality-of-service configuration may be defined implicitly or explicitly, while the data security configuration is always explicitly defined.

A.6.5.2 Data Considerations

Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* A DDS-ContentFilteredTopic can be used to subscribe to only a subset of data from a DDS-Topic.
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* A TIMEBASEDFILTER QoS policy can be used to subscribe to a subsampled subset of the data.

A.6.5 IMPLEMENTATION VIEWPOINT	
A.6.4.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* The DDS standards are purposefully designed to support extremely low-latency real-time deterministic communications. The latencies are dependent on the underlying networking hardware and the DDS implementation. DDS was specifically designed to support the needs of real-time distributed systems and includes several QoS policies real-time data distribution, such as DEADLINE, LATENCY_BUDGET TRANSPORT_PRIORITY. DDS also can notify applications of delays, allowing the application to adapt to the situation. *DDS-TSN*[64] ensures guaranteed latency and jitter at the Ethernet TSN hardware level. Several DDS implementations have been documented have low latency (< 1ms) and low jitter (µs) on typical LANs, without needing specialized hardware. Even with commodity networking hardware, DDS implementations are being successfully used for closed loop distributed control.
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* The variation of latency and jitter with throughput will be implementation specific, based on the design trade-offs made by that implementation. Leading DDS implementations have been documented to have minimal change in jitter as throughput increases. Implementations can achieve throughput as high as 95% of the theoretical network bandwidth.

[64] See [OMG-DDS-TSN]

A.6.5 IMPLEMENTATION VIEWPOINT	
A.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data object size?* DDS can handle an increasing number of data objects. Every data object is identified by a GUID—the number of unique GUIDs limits the number of data objects in a domain. The port numbers available on a host limits the number of domains. There is no theoretical limit on the data-object size. In practice, it will be limited by the amount of memory available on a host.
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* DDS can effectively support interface evolution for an increasing number of distributed application components. Application components are loosely coupled—they interact with the data, not with each other; thus, the interfaces are data-oriented and can evolve independently. The data types in a data-oriented interface can also evolve through extension or mutation—the rules are defined in the DDS-XTYPES[65] specification.
A.6.5.5 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* DDS can support continuous availability over a defined system relevant time period. DDS accomplishes this by providing a continuous ongoing automatic discovery so that components can be added or removed at any time, by providing the ability to detect faults through configurable QoS policies; by providing local caching, by providing an optional DDS-Persistence Service to cache the data outside of specific application components and by providing QoS policies to repair and recover past data.
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* DDS supports recovery when fault conditions occur. It accomplishes this by signaling exceptions to the application layer, by allowing applications to specify certain QoS policies for recovering past data, by providing access to the automatic discovery data and by providing mechanisms to detect when a component in the distributed system fails.
A.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* DDS implementations are available for most commonly used operating system and CPUs. DDS implementations can run on devices with limited memory resources (<100kB). DDS-XRCE[66] implementations under can fit under 10kB of memory.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* DDS naturally supports incremental upgrades. DDS accomplishes this by automatic ongoing discovery when components are added or removed and encouraging loosely coupled data-oriented interfaces and support for data type evolution over a system's lifecycle.

[65] See [OMG-DDS-XTYPES]

[66] See [OMG-DDS-XRCE]

A.6.5 IMPLEMENTATION VIEWPOINT	
A.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* DDS is agnostic to network topologies. All network topologies can be used with DDS.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* A DDS data space is agnostic to the network constraints and can therefore span both the LAN and the WAN. DDS implementations allow application components to be located anywhere—on the LAN or across the WAN. DDS implementations provide mechanisms to deal with firewalls and other restrictions encountered when going across the WAN.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* DDS can support multiple independent and isolated communication paths between the same network endpoints. DDS accomplishes this by providing DomainParticipants, Topics and a PARTITION QoS policy. These allow a system architect to define fine-grained multiple and independent isolated communication paths between network endpoints, as needed to meet the system requirements.

Annex B ASSESSMENT TEMPLATE: OPC UA

This annex contains the assessment template for the Open Platform Communications Unified Architecture (OPC UA).

B.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.* OPC UA
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* OPC Foundation (OPCF) and IEC 62541
Description	*Short synopsis of the technology*. OPC UA is an industrial communication architecture for platform independent, high performance, secure reliable, and semantic interoperability between sensors, field devices, controllers, and applications at the shop-floor level in real-time and between the shopfloor and the enterprise IT cloud.
Application Domain(s)	*Application domains targeted by the connectivity technology.* Automation for manufacturing, buildings, process control and energy.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* Current technology mapping options include: • TCP for the transport/network layers • HTTP for transport/network layers • TLS for security
References	Website[67] *and other useful links to the technology.*

[67] See [OPC-UA]

B.6.2 BUSINESS VIEWPOINT	
B.6.2.1 Purpose *(Section 6.2.1)*	*Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.* Defines a comprehensive information modeling mechanism, which is fully extensible by specific vertical markets. Defines a standard set of services, which act on the information model. Expose information about the system, its configuration, topology and data context (the meta data) in the collective "address space" of the individual OPC UA servers. Allow this address space to be accessed by authorized OPC UA clients so they can see what is available and choose what to access.
B.6.2.2 Pedigree *(Section 6.2.2)*	*Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.* OPC UA is the next generation of the OPC protocol, which is widely deployed in industrial automation. OPC was introduced in 1996 based on Microsoft DCOM. This specification is now referred as "Classic OPC". OPC UA was first introduced in 2006 (version 1.0). It no longer depends on DCOM and uses Web-Services and Binary TCP protocols instead. OPC UA version 1.03 was released in 2013. It has been endorsed as a key specification for Industry 4.0. OPC UA version 1.04 was released in 2018. Since then, the OPC UA publish subscribe extension, and a large number of information models associated with different industrial verticals, have been released.
B.6.2.3 Variants *(Section 6.2.3)*	*Describe the options and variants from the original generic description of the technology.* The OPC UA specification defines many optional profiles and services, notably: Discovery, View, Query, Attribute, Method, Data Monitoring, Data Access and Events & Conditions.
B.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* A website[68] maintains a list of notable projects that use OPC UA. OPC UA has broad industrial support. Its focus is to allow information to be easily and securely exchanged between diverse platforms from multiple vendors and to allow seamless integration of those platforms without costly, time-consuming software development. There are SDKs from multiple companies that can be used to build OPC UA compliant systems.
B.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* "Classic OPC" has been widely deployed in the industry. The OPC UA specification has been stable for many years as has the standard stack implementations and SDKs.

B.6.2 BUSINESS VIEWPOINT	
B.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* OPC UA enjoys broad industry support. The OPC foundation has over 850 members.[69]
B.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* OPC UA is specified as IEC 62541 standard and therefore allows for individual, royalty-free, implementation according to the standard, certification and technology contribution. OPC UA is an open standard. Access to specifications and developer resources are available with a registration on the OPCF website. Membership requires payment of annual dues at selected membership level. The specifications process is open to participation by both vendors and users. Open source and commercial implementations are available. The OPC UA specifications do not rely on any single component from any single vendor.

[68] See [OPC-CS]
[69] See [OPC-MEM]

B.6.3 USAGE VIEWPOINT	
B.6.3.1 Architecture *(Section 6.3.1)*	*Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.* OPC UA consists of multiple OPC UA-Clients connected to a OPC UA-Server. A OPC UA-Server holds an address space, which is a collection of data objects organized in a linked graph. Requests originate at a OPC UA-Client and are sent to an OPC-Server; the OPC-Server processes the request and sends a reply back to the OPC UA-Client. Requests are addressed to a specific data object in the server's address space. Structured data is used for the request and reply. A OPC UA specification for publish and subscribe architectures is currently under development.
B.6.3.2 Technology Options *(Section 6.3.2)*	*List the choices to be made for using the connectivity technology in a system.* • Selection of SDK used to implement OPC UA clients and servers supporting the desired variants (OPC UA profiles). • Selection of the underlying transport: OPC UA Binary/TCP or XML/HTTP.
B.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* Industrial automation and process control applications. Client-server interactions between components such as devices or applications. Expose the address space of systems and devices to facilitate configuration, browsing and data access.
B.6.3.4 Typical Usage *(Section 2.2)*	*What function or where in the system this technology is typically used?* OPC UA is deployed on devices to allow device configuration and data-access. For existing brown field installations, OPC UA is typically deployed at system boundaries to expose the system address space, support browsing, configuration, monitoring and service invocation. Newer devices and systems are building in OPC UA.
B.6.3.5 Operations *(Section 2.3.8)*	*Can one monitor, manage, and dynamically replace elements of the communication function?* OPC UA discovery services are defined to allow dynamic discovery of components.
B.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* Security is provided at the message and transport level between each client and server. Clients are authenticated via name and password, PKI certificate, of WS-Security Tokens. Each server enforces access control. Servers may support fine-grained access control to individual variable and operations.
B.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* There are currently no safety-certified OPC UA implementations.

B.6.3 USAGE VIEWPOINT	
B.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* • An OPC UA/DDS gateway standard is under development by the OMG. • An OPC UA DDS profile is under development by the OPC Foundation. • An OPC UA gateway standard is under development by oneM2M. • OPC UA clients can connect to OPC UA servers via HTTP. There are commercially available gateways between OPC UA and many industrial protocols such as Modbus, Profibus and Foundation fieldbus.

B.6.4 FUNCTIONAL VIEWPOINT	
B.6.4.1 Core Framework Layer Functions	
Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* OPC UA resources are called *nodes*. They can be individually addressed using a NodeID. Nodes contain data elements, operations and references to other nodes.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* OPC UA nodes have a unique identifier within a server, called the NodeID. Addressing a node also requires addressing the OPC UA server using its network IP address and port, plus the NodeID.
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* OPC UA defines a full data type system. Data-variables within nodes can be simple or complex (structured) data types.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data object's lifecycle? Summarize the salient aspects.* There is no pre-defined resource lifecycle in OPC UA. However, applications can define their own lifecycles and operations to control the resources.
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* The variables within each node constitute the state of the Node. Each OPC UA server manages its own state that is accessible via query and browsing services. They cache the last value of every variable they contain. Those can be queried using the query service. There is also a full set of historical data access services defined.
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* OPC UA supports the publish-subscribe communication pattern.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* OPC UA supports the request-reply communication pattern. It was the initial communication pattern of OPC UA.
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* OPC UA servers can implement a discovery service allowing client applications to discover the nodes they contain. An OPC UA server may provide a global registration and discovery service, allowing discovery of all the OPC UA Servers in a system.
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* Exceptions are supported and communicated via events and alarms.

B.6.4 FUNCTIONAL VIEWPOINT	
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* OPC UA offers limited QoS options beyond the ability to specify an update frequency for the monitored data. There are many QoS features built into the services of OPC UA to provide appropriate QoS.
Data Security *(Section 4.1.11)*	*Does it provide a data object security model? Summarize the salient aspects.* OPC UA authenticates clients using either a username/password, or a PKI X509 certificate, or a WS-SecurityToken. Servers may support fine-grained access control to individual variable and operations.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* OPC UA is a reference architecture specification. A typical user is expected to use software tools to integrate devices adhering to the OPC UA specification. There is no expectation of software development, and therefore no need for standardized APIs.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* OPC UA provides standardized means for configuring and administering the data types, information models and security. Monitoring of OPC UA servers is implementation specific but can be offered and discovered via standardized OPC UA mechanisms.

B.6.4 FUNCTIONAL VIEWPOINT	
B.6.4.2 Core Transport Layer Functions	
Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* OPC UA supports different transport protocols, and different message encodings. The supported transport protocol and message encodings depend on whether the request-reply or the publish-subscribe communication pattern is used. Different transport protocols and message encodings have different characteristics. For the request-reply communication pattern, OPC UA defines an abstract transport protocol, OPC UA Connection Protocol (UACP), that specifies a set of requirements a transport protocol must fulfil, and a set of transport protocols used to implement the UACP. For the request-reply communication pattern, OPC UA supports the following transport protocols: • TCP: OPC UA transport using a TCP connection. • HTTPS: OPC UA transport using HTTPS. • WebSockets: OPC UA transport using WebSockets. NOTE: OPC UA transport using SOAP was deprecated in version 1.03. For the publish-subscribe communication pattern, OPC UA supports the following transport protocols: • UDP: OPC UA transport using UDP. • Ethernet: OPC UA transport using Ethernet (without IP and UDP). • AMQP: OPC UA transport using Advanced Message Queuing Protocol (AMQP). • MQTT: OPC UA transport using Message Queue Telemetry Transport (MQTT). MQTT is widely supported by cloud computing platforms. There are no OPC UA specific message-size limits for OPC UA messages. Possible message-size limits associated with the transport protocol apply. For the request-reply communication pattern, OPC UA supports the following message encodings: • Binary: Binary message encoding. • XML: Message encoding using Extensible Markup Language (XML). • JSON: Message encoding using JavaScript Object Notation (JSON). JSON is widely supported by cloud platforms. For the publish-subscribe communication pattern, OPC UA supports the following message encodings: • UADP: Optimized binary message encoding. • JSON: Message encoding using JSON.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* When the request-reply communication pattern is used, OPC UA supports unicast communication mode. When the publish-subscribe communication pattern is used, OPC UA supports unicast, broadcast and multicast communications modes.

B.6.4 FUNCTIONAL VIEWPOINT	
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* When the request-reply communication pattern is used, OPC UA defines two types of endpoint roles: OPC UA client and OPC UA server. Clients sends request messages to servers. When the publish-subscribe communication pattern is used, OPC UA defines two endpoint roles: OPC UA publisher and OPC UA subscriber. Publishers sends messages that are received by zero, one or multiple subscribers. A given entity can support multiple endpoint roles. An entity that supports the publish-subscribe communication pattern typically acts as both publisher and subscriber. The specification relies on the endpoint-addressing scheme provided by the underlying transport mapping. For IP-based transport mappings, an IP address and a port number identify an endpoint.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* When the request-reply communication pattern is used, OPC UA defines usage of connection-oriented transport protocols. When the publish-subscribe communication pattern is used, OPC UA defines usage of connectionless transport protocols.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* OPC UA does not define means to prioritize messages. However, an OPC UA server or an OPC UA subscriber can choose to prioritize the processing of requests, depending on different criteria, including who has sent the message and the nature and purpose of the message.
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* OPC UA does not provide timing and synchronization services. However, if Ethernet Time Sensitive Networking (TSN) is used as transport protocol, the timing and synchronization services provided by TSN can be used.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* OPC UA supports message security, which can be provided both on the OPC UA layer and on the transport layer when using a transport protocol that provides content security (e.g. TLS or DTLS).

B.6.5 IMPLEMENTATION VIEWPOINT	
B.6.5.1 System Architecture Considerations	
Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* OPC UA does not require a broker, a special process or other network intermediary. However, some of the transport protocols might use brokers (e.g. MQTT). If the multicast communication mode is used, the communication network needs to support multicast.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints to participate in information exchange?* Data is accessible as variables, abstracted away from the physical endpoints. Clients can use OPC UA discovery to locate which server or servers provides a variable of interest. Aggregated OPC UA servers are used to provide applications with a single unified address space and abstract away the physical server providing the variable. Thus, OPC UA is node-centric, which maps closely to a device-centric approach.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* The OPC UA service definition and the information model provide governance. Thus, governance is explicit and shareable.
B.6.5.2 Data Considerations	
Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* The data subset of interest can be specified by content. For example, event data is subscribed to using a filter (essentially a stream filter) that compares the content of each event with a set of criteria provided by the client and only sends the subset that matches. Data subscriptions are also based on filtering.
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* A client and subscriber can request its own sampling rate.
B.6.5.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* Ethernet Time Sensitive Networking (TSN) provides deterministic latency and jitter for real-time applications.
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* Expected to be similar to that of TCP (see section 7.2). Use of binary protocols and direct client-server connections expected to result in throughput limited only by the network bandwidth and CPU of client and server computers.

B.6.5 IMPLEMENTATION VIEWPOINT	
B.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data object size?* OPC UA can handle an increasing number of data objects. In OPC UA, the number of data objects on a server would be limited by the memory on the server and the average size of the data objects in that server. There is no upper limit on the size of a request or reply. Scalability for a server (with respect to the number of clients) is limited by the number of TCP connections it can sustain as well as the number of independent monitor streams it can produce.
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* OPC UA protocol can be extended with new services and data types in a backward-compatible manner. OPC UA is a carefully thought out set of services that address the needs of device integration. Unlike approaches (such as SOAP, REST) where each application defines a new service API for specific purposes, resulting in an explosion of services, once an application supports the service set it needs, it can interface with any device or system.
B.6.5.5 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* OPC UA defines several redundancy features that allow seamless client and server failovers. For example, subscriptions defined with a server can be transferred to a redundant server without the need of the application to recreate a new subscription.
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* The ability to support redundancy and then monitor the current operating state are defined in the standard, and applications can discover these abilities.
B.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* OPC UA implementations are available for a variety of operating systems, CPUs and resource constraints. Different OPC UA profiles have been defined for different types of devices, with different types of requirements and resource constraints.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* OPC UA supports incremental upgrades by allowing OPC UA Servers to be updated independently of the clients when a device is upgraded. A server can be upgraded while a redundant server provides continuous support to its clients with no downtime.

B.6.5 IMPLEMENTATION VIEWPOINT	
B.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* Any OPC UA Client can connect directly to any OPC UA Server. OPC UA is agnostic to network topologies.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* OPC UA can span across both the LAN and the WAN, as long as the servers are accessible via an IP network.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* Strictly speaking, there is no segmentation of the information, but servers can be browsed individually and configured to expose different parts of the system. Servers are free to expose multiple endpoints on the same network segment or different segments. Each endpoint can offer different security requirements and expose different address spaces. In this way, there are no limitations.

Annex C ASSESSMENT TEMPLATE: ONEM2M

This annex contains the assessment template for oneM2M.

C.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology:* oneM2M
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* oneM2M is a partnership project that includes partners from major regional SDOs and other fora (i.e. ARIB, ATIS, CCSA, ETSI, TIA, TSDSI, TTA, TTC, Broadband Forum, CEN, CENELEC, Global Platform, New Generation M2M Consortium, Open Mobile Alliance).
Description	*Short synopsis of the technology*. oneM2M provides a common service layer that sits between applications and connectivity transport. It offers functions that IoT applications across different industry segments commonly need. Those functions are exposed to applications via RESTful APIs. oneM2M standards comprise a horizontal platform architecture that fits within a three-layer model comprising of applications, middleware services and networks. oneM2M's connectivity standards permit applications that are hosted on connected machines and devices, enterprise systems and mobile devices to communicate with each other in an efficient, secure manner. The oneM2M horizontal platform is scalable as the common service elements can be deployed on hosts, at the proximal network edge or within the enterprise cloud. Connectivity services provide capabilities that allow for efficient communication between application endpoints. It provides native QoS as well as interworking mechanisms that adjust the QoS of the underlying network (e.g. mobile, wireless) to meet the needs of current application data exchange.
Application Domain(s)	*Application domains targeted by the connectivity technology.* oneM2M service layer is suitable for both IIoT and large-scale consumer IoT applications. oneM2M specifically targets the IIoT application domains, including telematics and intelligent transportation, home automation, utilities, healthcare, smart cities and industrial automation.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* oneM2M service layer supports direct bindings to the following network layer/connectivity protocols: • CoAP • HTTP • MQTT • WebSockets
References	Website[70] *and other useful links to the technology.*

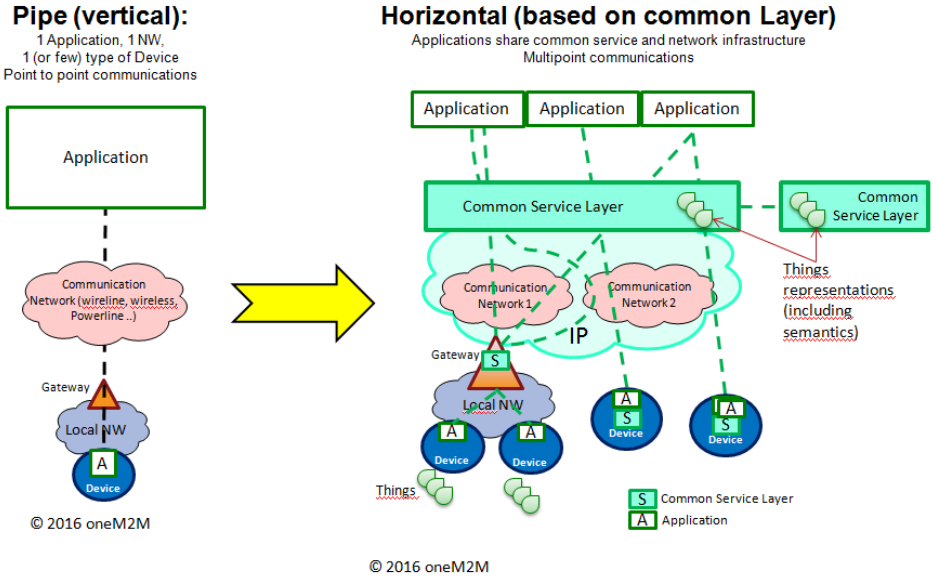
[70] See [ONEM2M]

C.6.2 BUSINESS VIEWPOINT

C.6.2.1 Purpose
(Section 6.2.1)

Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.

oneM2M standards that constitute the horizontal IoT platform allow applications from various previously siloed domains and for applications within a domain to communicate effectively, reliably and securely.



The oneM2M interoperable, platform architecture offers significant strategic benefits by consolidating the resources needed to deliver a variety of IoT applications and opening new service and business opportunities by allowing applications to share resources and data.

C.6.2.2 Pedigree
(Section 6.2.2)

Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.

oneM2M service layer was developed specifically to address solutions in the M2M/IoT community. The core of the specification was initially developed by ETSI as ETSI M2M. The initial specification for oneM2M was published in January 2015 as release 1.0. Release 2.0 of the specification set was published in August 2016.

oneM2M enjoys an active and vibrant community (over 200 member companies) continuously working to extend its applicability. The full list of the oneM2M family of specifications can be found at a website.[71]

Multiple independent oneM2M implementations are available, including both open-source and commercial.

C.6.2.3 Variants
(Section 6.2.3)

Describe the options and variants from the original generic description of the technology.

None. Implementations may differ in their support and coverage of the oneM2M specifications or compliance profiles.

[71] See [ONEM2M-PS]

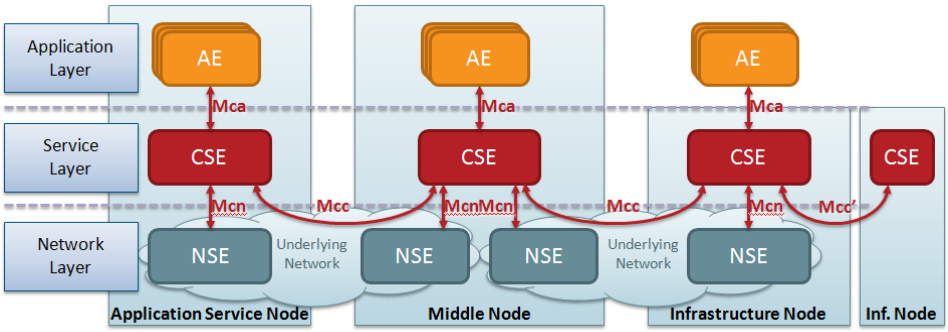
C.6.2 BUSINESS VIEWPOINT	
C.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* oneM2M specifications were published since January 2015. Since then there have been multiple interoperability events and commercial implementations. Certification of oneM2M-compliant nodes is in the process in certain regions.
C.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* Since publication in January 2015, oneM2M has incorporated fixes and clarifications that came from various interoperability and commercial deployments. The rate of fix and clarification requests has dropped off with the majority of the requests being clarification of the standard. As part of its ecosystem development, oneM2M has produced guides to assist application developers use the system. New features are consistently being released with the focus of the work items moving from the core connectivity and service functions to the specifications for new types of gateways needed for interoperability and development of domain-specific resource models.
C.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* oneM2M partner organizations and the ITU publish the specifications for their specific needs. oneM2M collaborates on features with a number of external organizations beyond the partnership organizations.
C.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* oneM2M is an open standard. The specifications are openly available to anyone at no cost. Anyone is free to download and implement them. The specifications process is open to participation by both member companies of the partner type 1 organizations as well as the partner type 2 organizations themselves. Open source and commercial implementations are available. The oneM2M specifications do not rely on any single component from any single vendor.

C.6.3 USAGE VIEWPOINT

C.6.3.1 Architecture
(Section 6.3.1)

Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.

oneM2M standards comprise a horizontal platform architecture that fits within a three-layer model comprising applications, middleware services and networks.



© 2016 oneM2M

Application Entities (AEs) are hosted on nodes (e.g. enterprise server, device). These nodes may be virtualized or physical instances. These AEs communicate with each other by sending requests to a Common Service Entity (CSE) that, in turn routes the request to the target AE while providing services based on the request.

© 2016 oneM2M

CSEs are hosted on the nodes that may be virtualized or physical nodes. In many instances, the AE and CSE share the same node (e.g. device).

C.6.3.2 Technology Options
(Section 6.3.2)

List the choices to be made for using the connectivity technology in a system.

- Selection of the oneM2M deployment: Applications interact with CSEs where the network of CSEs makes up the deployment architecture. The flexibility of placing the CSE on the end device, on the edge of a local network; in the cloud or somewhere in between (e.g. fog) are all deployment options.
- Selection of network layer binding(s) for a CSE.
- Selection of interoperability gateways for a CSE.

C.6.3 USAGE VIEWPOINT	
C.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* oneM2M has several commercial deployment pilots for smart city applications and is actively targeting industrial and intelligent transportation sectors. Reasons for using oneM2M include: • *Ease of integration:* The RESTful architectural approach used by oneM2M allows the definition of common and extensible data models for seamless Information Technology (IT)/Operational Technology (OT) interoperability. • *Performance efficiency and scalability:* oneM2M implementations allow for deployment configurations that place processing at locations where it can be used most efficiently. This allows for localization of messaging traffic to affected area networks. When properly architected, oneM2M-based systems can achieve near-linear scalability. • *Advanced security:* oneM2M defines a comprehensive security model for segment and end-to-end authentication, encryption, access control and logging capabilities to enable secure data connectivity end-to-end in an IoT system. • *QoS-enabled:* Determination of how a CSE treats, in terms of message delivery, is configurable based on a rich set of delivery policies including policies that allow the underlying network to be tuned to the message delivery characteristics of communicating applications. • *Scalable discovery:* For large-scale dynamic systems, oneM2M is in the process of developing automated onboarding and discovery of applications. • *Applicability:* oneM2M can transparently address applications that require data to be exchange between applications that transit through a CSE. Implementations are available for embedded, mobile, web, enterprise and cloud applications. • *Future proof:* The oneM2M specification enables end-to-end vendor interoperability and eases IoT system development and integration through fully open, future-proof APIs with no vendor lock in.
C.6.3.4 Typical Usage *(Section 2.2)*	*What function or where in the system this technology is typically used?* • Registration and Service Subscription (device and application onboarding and discovery): Used in infrastructure CSE. • Discovery of resources: Used in all CSEs. • Service charging and accounting: Used in the infrastructure CSE. • Data plane (data collection, subscription and notifications, data delivery, group management): Used in all CSEs. • Management plane (administration of applications and CSEs, device management): Used in infrastructure CSEs. • Integration with the underlying network layer services: Used in the infrastructure CSEs.
C.6.3.5 Operations *(Section 2.3.8)*	*Can one monitor, manage, and dynamically replace elements of the communication function?* oneM2M service layer provides the capability to monitor and manage applications and CSEs. These components are the building blocks of any oneM2M deployment.

C.6.3 USAGE VIEWPOINT	
C.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* oneM2M service layer defines a security model to authenticate applications and CSEs. All communication can be securely encrypted as a segment or end-to-end using the underlying network layer security mechanisms (e.g. TLS, DTLS). The security model is applied on top of the network layer.
C.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* While certain regions (e.g. Korea) have started certification processes for applications in that region for functional aspects of the oneM2M service layer and resource interaction, the certification process doesn't provide explicit references to which elements are directly related to the safety of a system.
C.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* oneM2M service layer supports interworking gateways with the following connectivity technologies: • OSGi (in progress) • Alljoyn • OIC (Open Interoperability Consortium) • LWM2M (Open Mobile Alliance) DDS is expected in the next release once it is determined whether support will be for a direct binding or through an interworking gateway. OPC UA interworking is expected in the next release.

C.6.4 FUNCTIONAL VIEWPOINT	
C.6.4.1 Core Framework Layer Functions	
Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* Yes, oneM2M service layer provides a data resource model that contains the user-defined structured data objects.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* oneM2M service layer data-objects resources can be identified by the system or by the user. The data-object resources can be semantically annotated by the user to be discoverable by users using various tools (e.g. data queries, ontological queries).
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* oneM2M service layer defines a data type system where the resources defined within the oneM2M system are encoded using XML or JSON encoding formats. User data-object resource can either retain their original encoding structures or can be interworked into abstract data-object resources defined within oneM2M.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data-object's lifecycle? Summarize the salient aspects.* oneM2M service layer is a RESTful resource-based system and provides a means to manage the full data-object lifecycle, including operations to create, read, update and delete data objects.
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* oneM2M service layer data-object resources provide mechanisms to manage the versions and histories of a data object including capabilities (e.g. number of versions, expiration dates, size constraints) for retaining the versions of the data objects. For resources that are communicated via the connectivity layer, oneM2M provides a rich set of policies used to determine when to transmit the data-object resources.
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* oneM2M service layer defines a mechanism where applications can subscribe to the events (e.g. creation, deletion, modification) of data-object resources. As part of the notification procedure, the application can receive the modified data-object resource or can be simply notified of the event.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* oneM2M service layer is a RESTful architecture, which is fundamentally a request-reply architecture. oneM2M provides a rich set of capabilities for communicating the request and receiving the reply
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* oneM2M service layer resources are discoverable using query mechanisms where the elements of the query criteria are defined by the owning applications or their delegates. In addition, oneM2M resources can be semantically annotated for use in ontological queries.

C.6.4 FUNCTIONAL VIEWPOINT	
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* oneM2M service layer provides a rich set of communication exception handling policies that all includes policies of what to do if a communication fails or the recipient is not available.
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* Yes, oneM2M service layer provides a set of communication QoS policies to determine the priority of delivering request (e.g. recipient, time-of-day, capacity limits).
Data Security *(Section 4.1.11)*	*Does it provide a data-object security model? Summarize the salient aspects.* oneM2M service layer provides a data-object security model. It defines security policies for access control (read, write, create, delete, notify), confidentiality (encryption). The owners or their delegated representatives administer security. All communication between endpoint can be authenticated and communication encrypted either end-to-end or by communication segment.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* oneM2M service layer uses RESTful architecture patterns (CRUD), extending the traditional RESTful architectural patterns for subscriptions, notifications and the ability to execute operations.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* oneM2M service layer provides standardized management APIs for configuring, administering and monitoring CSEs and applications.

C.6.4 FUNCTIONAL VIEWPOINT	
C.6.4.2 Core Transport Layer Functions	
Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* The oneM2M service layer runs on top of multiple transport protocols that are based on IP (i.e. CoAP, HTTP, Web Sockets, MQTT). oneM2M messages do not have a defined size limitation but are either limited by the underlying transport protocol's limitation (e.g. HTTP, Content-Length, MQTT 256Meg) or uses the underlying protocols (e.g. CoAP block) mechanisms for message fragmentation and reassembly.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* oneM2M service layer supports both unicast (default) and multicast (when available by the underlying transport) for group-based operations.
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* oneM2M service layer is a RESTful architecture where all resources are addressed using URIs, this includes the endpoints represented by applications and CSEs. The URIs can be defined within the address space of the M2M service provider, or they can be globally unique id (GUID). Endpoints follow the addressing scheme defined in the oneM2M specification.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* oneM2M service layer is a RESTful client/server architecture and does not require a connected circuit between the application endpoints. oneM2M does require endpoints to register with CSE's to which it connects, but the registration is not reliant on a connected connection between the endpoints.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* oneM2M service layer provides communication QoS policies to determine the priority of delivering request (e.g. recipient, time-of-day, capacity limits).
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* oneM2M service layer does not provide a way to synchronize time between endpoints. In oneM2M systems this is typically accomplished using a separate time synchronization protocol.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* oneM2M service layer provides mechanisms for message security. It provides support for authentication of endpoints either end-to-end or endpoint-to-CSE, message encryption (both end-to-end or endpoint-to-CSE) and message integrity.

C.6.5 IMPLEMENTATION VIEWPOINT	
C.6.5.1 System Architecture Considerations	
Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* oneM2M service layer requires applications to connect to a CSE to communicate with other applications. The target applications are not required to be connected to the same CSE as the source applications. CSEs are connected in a hierarchical tree topology with a root CSE (i.e. IN-CSE) that is in the domain of the M2M service provider.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints in order to participate in information exchange?* An application does not have to be aware of other endpoints to participate in an information exchange. Applications interact directly with the data-object resources organized by the owning application and never directly with each other.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* With the exception that oneM2M has defined the domain of data types and resources used by applications and CSE, oneM2M service layer does not require the governance to be implicit and allows system architects to choose the style of governance including how data-object resources are structured and identified.
C.6.5.2 Data Considerations	
Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* oneM2M service layer provides a number of content-filters that can be used for discovery of resources (e.g. time, size, content type, user defined, semantic criteria).
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* oneM2M service layer has a number of filters for time-based selection based on creation, modification and expiration times. In oneM2M there is a specialized content-based resource, called timeSeries, which provides various criteria for collection (sampling) and reporting of data.
C.6.5.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* Real-time data distribution performance of oneM2M deployments have not been documented and made publicly available. Real-time characterization is not currently available.
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* To date, the latency and jitter aspects of throughput of oneM2M deployments have not been documented and made publicly available. Latency and jitter characterization is not currently available.

C.6.5 IMPLEMENTATION VIEWPOINT	
C.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data-object size?* oneM2M service layer can handle an increasing number of data objects. However, the size of the object identifiers is limited by the size of string data type and that defines an upper bound on the maximum number of data objects. There is no theoretical limit on the data-object size. In practice, it will be limited by the amount of memory available on a host or the constraints of the underlying transport.
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* oneM2M service layer can effectively support interface evolution for an increasing number of distributed application components. Application components are loosely coupled—they interact with resources in CSEs not with each other; the interfaces are data-oriented and can evolve independently. Applications are in complete control of how the data-object resources are organized and identified.
C.6.5.5 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* As a service layer oneM2M does not place constraints on the availability of CSEs or applications. CSEs can use standard techniques for redundancy (load-balancers, clusters, virtualized environments).
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* oneM2M service layer can support recovery when fault conditions occur. It accomplishes this by informing applications of errors and by allowing applications to change the behavior of how the system treats communication with endpoints.
C.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* oneM2M service layer does not require a specific type of OS, CPU or even the database management system. oneM2M has been architected to work with constrained devices in mind.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* The oneM2M service layer does not place constraints on the upgradability of CSEs or applications. CSEs can use standard techniques for upgrading the CSE (load-balancers, clusters, virtualized environments).

C.6.5 IMPLEMENTATION VIEWPOINT	
C.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* oneM2M service layer is agnostic to network topologies.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* oneM2M service layer can be used within the LAN or across the WAN. CSEs and application can be located in either the LAN or WAN. The root CSE of the hierarchical tree is usually located within the M2M service providers WAN environment. oneM2M service layer implementations provide mechanisms to deal with firewalls and other restrictions encountered when going across the WAN.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* Since the oneM2M service layer runs on IP-based communication protocols. As such it can support multiple independent and isolated communication paths between the same network endpoints using the underlying IP network's mechanisms for path redundancy and isolation.

Annex D ASSESSMENT TEMPLATE: HTTP

This annex contains the assessment template for Hypertext Transfer Protocol (HTTP).

D.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.* Hypertext Transfer Protocol (HTTP)
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C)
Description	*Short synopsis of the technology*. HTTP is the connectivity transport designed for the world wide web. Its primary goal is to serve the needs of web browsing. It is also used to expose application server interfaces as web services.
Application Domain(s)	*Application domains targeted by the connectivity technology.* World wide web. User interfaces. Collaborative hypermedia applications.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* • TCP • TLS for Transport Level Security
References	Website[72] *and other useful links to the technology.*

[72] See [HTTPWG]

D.6.2 BUSINESS VIEWPOINT	
D.6.2.1 Purpose *(Section 6.2.1)*	*Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.* HTTP is the core connectivity transport of the world wide web. It was developed to support browsing the web of interconnected pages of hypertext markup language (HTML) and associated resources required to render a web page. As a result of its widespread availability, it has been used for exposing application sever interfaces, commonly referred to as web-service APIs.
D.6.2.2 Pedigree *(Section 6.2.2)*	*Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.* HTTP originated in 1990 ~ 92 with informal drafts describing it as a protocol for collaborative hypermedia applications. HTTP/1.0 appeared as an IETF informational RFC in 1996. HTTP/1.1 appeared as a draft standard in 1999. It expanded upon the TCP binding and was finalized in 2014. HTTP/2 appeared in 2015. It further optimized the TCP binding, while preserving the semantics.
D.6.2.3 Variants *(Section 6.2.3)*	*Describe the options and variants from the original generic description of the technology.* None.
D.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* HTTP is a mature technology. It forms the basis of the world wide web.
D.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* HTTP is stable. HTTP/1.x is widely deployed. Toolkits for writing HTTP clients are available in nearly all the popular programming languages. There is a large selection of HTTP server implementations to choose from. Open-source and proprietary implementations are available.
D.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* Internet Engineering Task Force (IETF)
D.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* HTTP is an open standard managed by the IETF. Participation in the standards process is open to all. There are no annual dues and the IETF standards are available free of charge. The specifications process is open to participation by both vendors and users. Open source and commercial implementations are available. The specifications do not rely on any single component from any single vendor.

D.6.3 USAGE VIEWPOINT	
D.6.3.1 Architecture *(Section 6.3.1)*	*Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.* HTTP defines a request-reply application protocol to exchange application state represented as hypertext with embedded resource identifiers. A client can request some action on a server resource and be informed of the outcome of that request. A client request consists of an action method and a resource (path) on which the method is to be applied. The server replies to the request with a status code, which informs the client of the outcome of the method or the reason the method was not performed. Each request or reply message can have associated resource representation metadata header fields, which are name-value pairs and provide additional information about the operation. Some headers are pre-defined, and applications can add their own headers. Each request or reply message can also have an optional body to hold a resource representation, which is hypertext with embedded links to resources.
D.6.3.2 Technology Options *(Section 6.3.2)*	*List the choices to be made for using the connectivity technology in a system.* • Selection of resource representation format. • Multiple implementations choices for client and server libraries are available, including open source and proprietary, in a variety of programming languages. Implementations vary in their quality, performance, scalability, availability and security characteristics.
D.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* HTTP is most commonly known for navigating web pages and building application server interfaces. HTTP based applications are typically driven by a human end user. In the context of IoT, HTTP is also used as a connectivity transport for IT applications.
D.6.3.4 Typical Usage *(Section 2.2)*	*What function or where in the system this technology is typically used?* HTTP is typically used for serving web pages, for exposing application server interfaces and as a connectivity transport layer for some connectivity frameworks.
D.6.3.5 Operations *(Section 2.3.8)*	*Can one monitor, manage, and dynamically replace elements of the communication function?* A component that can serve the same resources can replace a server. A server can support multiple clients.
D.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* HTTP uses transport layer security (TLS) to provide end-to-end authentication, encryption and integrity. HTTP over TLS is referred to as HTTPS.
D.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* There are no known safety-certifiable implementations of HTTP.

D.6.3 USAGE VIEWPOINT	
D.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* Gateways to HTTP are defined by other core connectivity standards: • OMG's Web-Enabled DDS defines a gateway for DDS. It allows HTTP clients to participate in a DDS data space. • OPC UA supports a web service protocol using HTTP. • oneM2M uses HTTP as a connectivity transport option. • HTTP to CoAP forward and reverse proxy.

D.6.4 FUNCTIONAL VIEWPOINT	
D.6.4.1 Core Framework Layer Functions	
Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* HTTP provides a data resource model. A data object is represented via an HTTP resource, formatted as a uniform resource identifier (URI) path string that is meaningful in context of the server. HTTP defines a core set of methods (GET, POST, PUT, DELETE) that can be applied to the resources on a server. Resource representation returned by a server is hypertext that provides the context and links to other resources. To drive the application state, application architects will define the hypertext and the resource organization.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* The HTTP resource URI path string provides a way of identifying and addressing a data object within a server. The server itself is addressed as a network IP address and port number. The result of combing the URI with the network endpoint is a Uniform Resource Locator (URL), expressed as *http://* or *https://* (when TLS is used for security).
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* HTTP does not provide a data type system.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data object's lifecycle? Summarize the salient aspects.* HTTP provides a means for managing data object lifecycles. A client can use the POST or the PUT method to create a data object, the GET method to retrieve the data object's representation, the PUT method to update a data-object's representation and the DELETE method to delete its representation. The server controls which methods are applicable on a data object, via the response for each operation.
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* HTTP provides cache-mechanisms for proxies and clients to maintain responses of previous requests. These responses may contain the representations of the resources. The HTTP specification defines mechanisms for determining the freshness of the caches and provides rules for access control and applicability of a cached response.
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* HTTP does not provide a means to publish and subscribe to the state of data objects.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* HTTP provides a means to request the state of data objects. This is the fundamental means of communicating using HTTP.
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* Data objects can be discovered via the embedded resource links in the hypertext.
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* HTTP does not assume that connection will be continuously available. It supports a request timeout error code, when a server does not receive a complete request within the time it was prepared to wait. When service on a resource is unavailable, a server can inform the client to retry after a certain amount of time.

D.6.4 FUNCTIONAL VIEWPOINT	
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* HTTP does not provide data qualify of service as described in section 4.1.10.
Data Security *(Section 4.1.11)*	*Does it provide a data-object security model? Summarize the salient aspects.* HTTP does not provide a data-object security model.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* HTTP does not provide a standardized programming API. However, libraries are available in most popular programming languages that provide user-friendly APIs.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* HTTP does not define a standard way to configure, administer, and manage a server. Configuration, administration, and monitoring of HTTP servers is implementation specific. It is common practice to use configuration files for administration and log files for monitoring.
D.6.4.2 Core Transport Layer Functions	
Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* HTTP relies on TCP. It required reliable, ordered delivery of requests and responses. It can support partial or chunked delivery of requests and responses. There are no inherent message size limitations.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* Unicast.
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* A transport endpoint is a server IP address and a port number. There is no inherent limitation on the number of endpoints.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* HTTP does not require a connected circuit between a client and server. TCP connections may be torn down after a request-response and reestablished for the next one.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* HTTP does not provide a means to prioritize messages.
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* HTTP does not provide the ability to synchronize time.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* HTTP can use Transport Layer Security (TLS) over TCP to provide message security.

D.6.5 IMPLEMENTATION VIEWPOINT	
D.6.5.1 System Architecture Considerations	
Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* HTTP does not require running a special process or broker to communicate between the client and the server.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints in order to participate in information exchange?* The client application code does not have to be aware of the server implementation details to participate in a data exchange. The server responses indicate the available resources and the methods allowed on them.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* The governance is implicit, embedded in the request and response headers and data exchanged between a client and a server.
D.6.5.2 Data Considerations	
Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* HTTP does not provide a content filtering mechanism to specify a data subset of interest. However, it does support the concept of "content negotiation" between a client and a server. It is left to the server to define the results of the content negotiation.
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* HTTP does not provide a sub-sampling mechanism to specify a data subset of interest.
D.6.5.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* HTTP is not designed to support real-time data distribution. The latency is not deterministic. The use of TCP can result in unbounded latency and jitter.
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* Latency and jitter can suffer as throughput increases. The throughput is limited by the message size, network bandwidth and available memory.
D.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data object size?* HTTP can handle an increasing number of data objects. There is no inherent limitation on the representation size of a data object. A data-object resource representation may be finite or may be unbounded.
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* HTTP can support interface evolution for an increasing number of distributed application clients, since the hypertext is used to decouple the clients from the server state. The hypertext response from a server defines its interface and controls the resources and methods available to its clients.

D.6.5 IMPLEMENTATION VIEWPOINT	
D.6.5.6 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* HTTP can support continuous availability over a system-relevant time period, as evidenced by the WWW.
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* HTTP can support recovery when fault conditions occur.
D.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* HTTP is generally available on a wide variety of operating systems on a variety of CPUs, including embedded devices.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* HTTP facilitates incremental upgrades. Since the hypertext from a server controls the client interface, it can be upgraded any time. This is evidenced by the success of the WWW.
D.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* HTTP is agnostic to network topologies, as it runs above the network layer.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* HTTP can be used over LAN and WAN. It is typically used over WAN, as is evidenced by the WWW. IT infrastructure and firewall are friendly to HTTP.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* HTTP can support multiple independent isolated communication paths between the same network endpoints.

Annex E ASSESSMENT TEMPLATE: COAP

This annex contains the assessment template for Constrained Application Protocol (CoAP).

E.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.* Constrained Application Protocol (CoAP)
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* Internet Engineering Task Force (IETF)
Description	*Short synopsis of the technology*. The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in IoT. The protocol is designed for machine-to-machine (M2M) applications, such as smart energy and building automation.
Application Domain(s)	*Application domains targeted by the connectivity technology.* IoT scenarios where devices are constrained, in memory or CPU or both. Scenarios that require interoperability between web technologies and the general internet with the IoT device domain.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* • UDP (DTLS) • TCP • TLS • Web Sockets
References	Website[73] *and other useful links to the technology.*

[73] See [CoAP] and [Clark-2017]

E.6.2 BUSINESS VIEWPOINT	
E.6.2.1 Purpose *(Section 6.2.1)*	*Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.* CoAP is designed to communicate with resource-constrained devices or with devices across constrained (lossy or low throughput) IP networks. The design goals of CoAP are to provide a generic web protocol that keeps message overhead small, thus limiting the need for fragmentation. It offers features such as built-in discovery, multicast support and asynchronous message exchanges.
E.6.2.2 Pedigree *(Section 6.2.2)*	*Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.* CoAP was developed by the IETF as an internet standard. Work started on 2009 and culminated on RFC 7252 in 2014. There are several other supporting drafts and standards that relate to it. The IETF CoRE working group that maintains and enhances features related to CoAP enjoys an active and vibrant community with member continuously working to extend its applicability. Multiple independent CoAP implementations are available, including both open-source and commercial.
E.6.2.3 Variants *(Section 6.2.3)*	*Describe the options and variants from the original generic description of the technology.* There are no variants as such, but CoAP supports multiple transports UDP/TCP/SMS etc.
#.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* CoAP specifications have only been published since June 2014. Multiple interoperability events and commercial implementations have been deployed prior to the release of the publication as per the IETF process. Some of the many implementations are available at a website.[74]
E.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* The core RFC is solid and stable. The definition of extensions is still ongoing.
E.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* IETF[75] is where the CoAP standard is developed and maintained.

[74] See [CoAP-Impl]

[75] See [IETF]

E.6.2 BUSINESS VIEWPOINT	
E.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* CoAP is an open standard. The specifications are openly available to anyone at no cost. Anyone is free to download and implement them. The specifications process is open to participation by individuals. Open source and commercial implementations are available. CoAP specifications do not rely on any single component from any single vendor.

E.6.3 USAGE VIEWPOINT	
E.6.3.1 Architecture *(Section 6.3.1)*	*Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.* CoAP aims to provide more than plain connectivity or message passing functionality. Like HTTP it brings the RESTful architectural style of the world wide web to the constrained space. Servers make resources available under a uniform resource identifier (URI), and clients access these resources using methods such as GET, PUT, POST, and DELETE. A device (endpoint) will run a CoAP server and often a client too. Clients elsewhere (i.e. other devices, browsers, applications) can request resources on the device and discover new devices and functionality. From a developer point of view, CoAP feels like HTTP. Obtaining a value from a sensor is not much different from obtaining a value from a Web API. For more details, please refer to page 10 of RFC 7252.[76]
E.6.3.2 Technology Options *(Section 6.3.2)*	*List the choices to be made for using the connectivity technology in a system.* CoAP is a client/server model where the options include: • Selection of resource representation format. • Selection of transport layer binding(s): UDP or SMS or TCP and Web Sockets. • Selection of client and server implementation libraries. • Optional: Selection of HTTP proxy (CoAP-HTTP gateway). • Optional: Selection of resource directory server for resource discovery in constrained environments.
E.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* CoAP is a generic REST protocol upon which other technologies have been built. For device management, for example, the Open Mobile Alliance has created LWM2M, which supports management and operations of devices.
E.6.3.4 Typical Usage *(Section 2.2)*	*What function or where in the system this technology is typically used?* The protocol is versatile. It is suited for data collection, managed and unmanaged systems, systems that require scalability and systems that require security.
E.6.3.5 Operations *(Section 2.3.8)*	*Can one monitor, manage, and dynamically replace elements of the communication function?* CoRE specifications typically focus on protocol interactions and do not generally specify how elements of the *communication* functions are managed, monitored or replaced.
E.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* CoAP defines a security model to authenticate and encrypt the interaction between CoAP clients and servers based on the underlying network datagram transport layer (DTLS/TLS) security mechanisms. CoAP specifications provide different types of end-to-end security and analysis of several possible attack vectors; please refer to page 80 of RFC 7252.[77] A robust authentication and fine-grained access control security model for CoAP has been defined by IETF.
E.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* There are no known safety-certifiable implementations of CoAP.

E.6.3 USAGE VIEWPOINT	
E.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* CoAP supports interworking gateways with the following connectivity protocols: • HTTP to CoAP forward and reverse proxy.

[76] See [IETF-RFC7252]

[77] See [IETF-RFC7252]

E.6.4 FUNCTIONAL VIEWPOINT	
E.6.4.1 Core Framework Layer Functions	
Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* CoAP provides a data resource model, following the approach taken by HTTP. A data object is represented via a resource, formatted as a uniform resource identifier (URI) path string that is meaningful in context of the server. CoAP defines a core set of methods (GET, POST, PUT, DELETE) that can be applied to the resources on a server. Resource representation returned by a server provides the context and links to other resources. It is up to the application architects to define the representations and the resource organization, to drive the application state. CoAP servers can expose resource representations in a variety of formats using a variety of data models. CoAP servers can use the *CoRE link format* for the resource representations. The CoRE link format identifies the paths to the resources in the CoAP server and provides annotations to the resources, which includes items like the content-type, interface and the resource type. CoAP clients can use these annotations to better understand the semantics of the resource.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* The CoAP resource URI path string provides a way of identifying and addressing a data object within a server. The server itself is addressed as a network IP address and port number. The result of combing the URI with the network endpoint is a Uniform Resource Locator (URL). The URI schemes depend on the transport protocol. The following schemes have been defined for CoAP: • "coap" (CoAP with UDP transport) • "coaps" (CoAP with DTLS protected UDP transport) • "coap+tcp" (CoAP with TCP transport) • "coaps+tcp" (CoAP with TLS protected TCP transport) • "coap-ws" (CoAP with WebSocket transport) • "coaps-ws" (CoAP with TLS protected WebSocket transport)
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* CoAP does not dictate a specific data type system to define resources. CoAP simply transports resources as a payload on CoAP messages. However, the *CoRE link format* does provide target attributes for CoAP servers to report the content type of the resource.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data-object's lifecycle? Summarize the salient aspects.* CoAP provides a means for managing data-object lifecycles. A client can use the POST or the PUT method to create a data object; the GET method to retrieve the data-object's representation; the PUT method to update a data-object's representation; the DELETE method to delete its representation; the FETCH method to retrieve parts of a data object representation and the PATCH method to update parts of a data object. The server controls which methods are applicable on a data object, via the response for each operation.

E.6.4 FUNCTIONAL VIEWPOINT	
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* CoAP does provide cache mechanisms for proxies and CoAP clients to maintain responses of previous requests. These responses may contain the representations of the resources or links of where the resources are located. The CoAP specification defines mechanisms for determining the freshness of the caches and provides rules for access control and applicability of a cached response.
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* CoAP provides a way to *observe* (subscribe) to resources and get notifications when their state changes. The *OBSERVE* is an extension of the GET method with an additional option that requests the server to keep the representation updated over time. It is up-to to the server to determine how and when to notify the client of updates to resources. There is ongoing work in IETF to develop a publish-subscribe extension for CoAP.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* CoAP provides a means to request the state of data objects. This is the fundamental means of communicating using CoAP. All CoAP requests have an associated response.
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* Data objects can be discovered via the embedded resource links in the response from a server. The *CoRE link format* defines a simple format for exposing the resources offered by a CoAP server and forms the basis for a resource directory.
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* CoAP provides limited exception handling that resolves around the timeout when a CoAP request does not receive a response.
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* CoAP supports two levels of QoS: *confirmable* and *non-confirmable* (best efforts). The CoAP messaging layer provides implementations for confirmable and non-confirmable message delivery over unreliable network layer protocols (e.g. UDP).
Data Security *(Section 4.1.11)*	*Does it provide a data-object security model? Summarize the salient aspects.* Object security for CoAP has been defined by IETF.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* CoAP's API is a generic REST API. It does not provide a standardized programming API. However, libraries are available in most popular programming languages that provide user-friendly APIs.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* CoAP does not define a standardized way to configure, administer, and manage a server. Configuration, administration and monitoring of CoAP servers is implementation specific.

E.6.4 FUNCTIONAL VIEWPOINT	
E.6.4.2 Core Transport Layer Functions	
Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* The CoAP messaging protocol is an IP-based protocol. It supports multiple bindings that are based on IP (i.e. UDP, TCP, SMS, Web Sockets). By default, it works over UDP. CoAP messages size is based on a 32-bit integer, but CoAP messages work best without fragmentation, as such implementations tend to keep message sizes less than the underlying network transport layer payload sizes. A CoAP message, appropriately encapsulated, should fit within a single IP packet to packet to avoid IP fragmentation. When necessary CoAP does provide a mechanism to fragment and reassemble larger messages sizes.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* CoAP supports both unicast (default) and multicast (when available by the underlying transport).
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* A transport endpoint is a server IP address and a port number. There is no inherent limitation on the number of endpoints.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* CoAP does not require a connected circuit between a client and server.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* CoAP does not provide a way to prioritize messages.
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* CoAP does not provide a way to synchronize time between clients and servers.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* CoAP can use UDP transport with Datagram Transport Layer Security (DTLS) protection, or TCP and WebSocket transport with Transport Level Security (TLS) protection, to provide message security. The default DTLS and TLS parameters have been selected to be able to provide a high level of security also when running CoAP on constrained devices.

E.6.5 IMPLEMENTATION VIEWPOINT	
E.6.5.1 System Architecture Considerations	
Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* No brokers are required; it is peer-to-peer oriented. Communication occurs between endpoints acting as clients or servers.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints in order to participate in information exchange?* Clients do not have to be aware of the server behavior to participate in a data exchange. Clients need to have mechanisms for finding and operating on resources much as on the web. Servers can dynamically provide their interfaces to the clients. In practice, depending on the use case, it is feasible to build data-centric (RESTful, dynamic APIs) or device-centric (fixed API) architectures.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* The governance is implicit, embedded in the request and response headers and data exchanged between a client and a server.
E.6.5.2 Data Considerations	
Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* A CoAP client will only receive data associated with the resource to which the CoAP request is addressed. To request a subset of the data associated with a resource CoAP clients commonly use URI parameters. Alternatively, a CoAP client can use the message payload (when using the FETCH method) to indicate in what subset of the data the client is interested. Using the FETCH method and message payload gives more options and flexibility than using URI parameters.
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* When the CoAP observation mechanism is used, i.e. when a single request will trigger periodic responses (notifications), it is possible to specify time-related criteria for the notifications, including minimum and maximum amount of time between two notifications.
E.6.5.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* CoAP was not designed with real-time applications in mind but can support real-time applications based on the deterministic capabilities provided by the underlying transport layer.
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* Compared to HTTP, CoAP endpoints should not experience more latency, as it is constrained and avoids fragmentation at multiple layers. CoAP should have smaller latency and jitter, compared to HTTP when used over UDP.

E.6.5 IMPLEMENTATION VIEWPOINT	
E.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data-object size?* CoAP can handle an increasing number of data objects without constraints, as the limit of the resource identifier is the string size.
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* CoAP is designed with evolution and long-term robustness in mind; it can support future changes or extensions to the servers or clients, much like HTTP.
E.6.5.5 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* Data availability will be determined by the availability of the CoAP Server, if the server is down a cached version of the resource can be accessed but it won't be "fresh". CoAP doesn't provide functionality for redundancy as part of the protocol. This is considered part of the application layer.
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* CoAP does not provide functionality for recovery of fault conditions. This is considered part of the application layer.
E.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* CoAP is supported by several platforms, OSs and hardware. CoAP has been designed to work with constrained devices and networks in mind, and so can be made available on the smallest of platforms.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* CoAP, like HTTP, is designed with incremental updates and long-lasting client and server lifecycles. As a protocol CoAP does not place constraints on the upgradability of CoAP clients or servers. CoAP clients and server can use standard techniques for upgrading the CoAP client or server (load-balancers, clusters, virtualized environments).

E.6.5 IMPLEMENTATION VIEWPOINT	
E.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* CoAP is agnostic to network topologies. CoAP has been used on low power networks that have a central point of connectivity to the outside.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* CoAP can be used within the LAN or across the WAN. CoAP clients and servers can be located in either the LAN or WAN. CoAP implementations provide proxy mechanisms to deal with firewalls and other restrictions encountered when going across the WAN.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* CoAP is an IP-based communication protocol. As such it can support multiple independent and isolated communication paths between the same network endpoints using the underlying IP network's mechanisms for path redundancy and isolation.

Annex F ASSESSMENT TEMPLATE: MQTT

This annex contains the assessment template for MQTT (formerly MQ Telemetry Transport).

F.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.* MQTT (formerly MQ Telemetry Transport)
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* OASIS.
Description	*Short synopsis of the technology*. MQTT is a connectivity transport for lightweight machine-to-machine (M2M) messaging. MQTT uses a centralized broker and supports publish-subscribe communications pattern running on top of TCP.
Application Domain(s)	*Application domains targeted by the connectivity technology.* Telemetry. Connecting remote sensors to data centers. IoT scenarios where small code footprint is required, or network bandwidth is at a premium.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* • TCP • Recent addition MQTT-SN supports UDP • TLS or DTLS for security
References	Website[78] *and other useful links to the technology.*

[78] See [MQTT]

F.6.2 BUSINESS VIEWPOINT	
F.6.2.1 Purpose *(Section 6.2.1)*	*Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.* Provide connectivity to M2M applications where small code footprint is required or network bandwidth is at a premium. MQTT may be considered for applications that exhibit high-cost connections, high latency, variable availability and negotiated delivery guarantees.
F.6.2.2 Pedigree *(Section 6.2.2)*	*Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.* The protocol was created by IBM in 1999 as the MQ Telemetry Protocol (MQTT). In 2010 IBM published the protocol under royalty-free terms. In 2011 IBM contributed the MQTT standard to OASIS and in 2012 the source code to Eclipse. The first OASIS standard version of MQTT (version 3.1.1) was approved in 2014.
F.6.2.3 Variants *(Section 6.2.3)*	*Describe the options and variants from the original generic description of the technology.* MQTT-SN is a variation aimed at embedded devices on non-TCP networks.
F.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* A website[79] maintains a list of notable projects that use MQTT.
F.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* The baseline MQTT protocol is stable and has been available for a long time. The more recent MQTT-SN protocol is not as mature.
F.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* OASIS.[80]
F.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* MQTT is an open standard. OASIS members can participate in its development. The specifications are freely available. Open-source implementations are available. It does not require any single component from any single vendor.

[79] See [MQTT-P]

[80] See [OASIS]

F.6.3 USAGE VIEWPOINT	
F.6.3.1 Architecture *(Section 6.3.1)*	*Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.* MQTT comprises multiple MQTT-Clients connected to a MQTT-Server (or broker). MQTT-Clients publish and subscribe to messages on one or more MQTT-Topics. A message published at a client is sent to the MQTT-Server, which sends it to all the subscribed MQTT-Clients. An MQTT message is an opaque vector of bytes.
F.6.3.2 Technology Options *(Section 6.3.2)*	*List the choices to be made for using the connectivity technology in a system.* • Selection of MQTT versus MQTT-SN. • Selection of the MQTT broker. This is one for a segment of connected applications. • Selection of client libraries (can be different for each client application).
F.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* According to the OASIS MQTT Technical Committee, target applications are sensors communicating to a broker via satellite links, occasional medical device dial-up connections with healthcare providers, home automation and small device scenarios. MQTT also targets mobile applications.
F.6.3.4 Typical Usage *(Section 2.2)*	*What function or where in the system this technology is typically used?* Centralized data collection.
F.6.3.5 Operations *(Section 2.3.8)*	*Can one monitor, manage, and dynamically replace elements of the communication function?* MQTT does not provide standardized mechanisms to monitor and manage a MQTT-Server. However, MQTT-clients can be replaced at any time. The broker routes all messages in the system. To avoid becoming a bottleneck it is deployed so that there is high-bandwidth connectivity to all critical clients. The broker should be specially protected against security breaches and denial of service attacks.
F.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* Security is provided only at the transport level between each client and the broker. There is no end-to-end (client-to-client security). Therefore, if the broker is compromised, all data in the system will be compromised. The broker introduces a potential target to denial-service-attacks.
F.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* There are currently no safety-certified client libraries or brokers.
F.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* Custom application gateways have been developed for MQTT to DDS and HTTP to meet the needs of specific applications.

F.6.4 FUNCTIONAL VIEWPOINT	
F.6.4.1 Core Framework Layer Functions	
Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* MQTT does not provide a data resource model. Messages are directed to topics, which represent logical flows or streams. There is no explicit data-resource model in MQTT. A single topic might be used to represent data from multiple resources and the association will be encoded in the data and maintained by the clients.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* MQTT does not provide a way of identifying and addressing data objects. Addressing individual resources within the streams is left to the application code.
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* MQTT does not define a data type system. It transmits opaque data to be interpreted by the applications.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data-object's lifecycle? Summarize the salient aspects.* MQTT does not provide a means of managing a data-object's lifecycle. There is no explicit resource management. This would be implemented by the client applications.
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* MQTT does it not provide a means to manage the recent history of data objects. Given that there is no resource management there is also no state management provided by MQTT. However, the MQTT broker can retain messages to be delivered to late-joining applications and clients could use this to build state management at the application level.
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* MQTT provides a means to publish and subscribe messages on topics, but since there is no data-resource model, applications have to maintain the mapping of messages to state updates on data objects.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* MQTT does not provide a means to request the state of data objects.
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* MQTT does not provide a means to discover the data objects. Discovery is implicit by the fact that applications communicate via a broker. All client applications must connect to the same broker that has full knowledge of the topology of the system.
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* MQTT does not provide a means to handle exceptions when quality of service or connectivity violations happen.

F.6.4 FUNCTIONAL VIEWPOINT	
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* MQTT provides limited QoS support. It includes best efforts and reliable delivery, and a minimal level of durability so that subscribers can receive a special message after a publisher goes offline.
Data Security *(Section 4.1.11)*	*Does it provide a data-object security model? Summarize the salient aspects.* MQTT does not provide a data-object security model. Only username and password authentication are provided by the protocol. Security model is implemented by the broker and is not part of the MQTT standard.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* MQTT does not provide a standard programming API. The API is implementation dependent.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* MQTT does not define a standardized way to configure, administer, and manage a broker. Configuration, administration and monitoring of MQTT brokers is implementation specific.
F.6.4.2 Core Transport Layer Functions	
Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* The MQTT standard is the messaging protocol. Applications are responsible for building the communication framework on top of the MQTT transport protocol. There are no standards for this. MQTT requires TCP. MQTT-SN works over UDP.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* MQTT relies on unicast. MQTT-SN can use multicast but not with security.
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* MQTT endpoints are the MQTT-Client and the MQTT-Server. MQTT uses standard IP host and port number addressing combined with the name of the topic to direct messages. The number of TCP connections on the server host and the memory limits the number of endpoints.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* MQTT is a connection-oriented transport on top of TCP. MQTT-SN is a connectionless transport on top of UDP.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* MQTT does not provide a means to prioritize messages.

F.6.4 FUNCTIONAL VIEWPOINT	
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* MQTT does not provide the ability to synchronize time.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* MQTT can use transport-level security to authenticate a client with the broker so configured and provide integrity and confidentiality of the information: Transport Level Security (TLS) for MQTT and Datagram Transport Level Security (DTLS) for MQTT-SN.

F.6.5 IMPLEMENTATION VIEWPOINT	
F.6.5.1 System Architecture Considerations	
Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* Broker-based. It requires running MQTT-Server, a special broker process, for clients to communicate.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints in order to participate in information exchange?* The application code (or business logic) does not have to be aware of the other endpoints to participate in data exchange. MQTT is only a connectivity transport. Data-centric or device-centric connectivity frameworks can be built with application-specific code.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* The governance is not explicit and shareable. Governance is enforced by the server implementation. The means for this are not standardized.
F.6.5.2 Data Considerations	
Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* The data subset of interest cannot be specified by content. The content is opaque to MQTT.
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* One cannot subscribe to a sub-sampled data subset of interest. MQTT attempts to deliver the published messages to all the subscribers on a topic.
F.6.5.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* MQTT is a TCP and broker-based protocol and is not intended for real-time. Binary protocol offers low overhead, but the use of TCP and relay via a broker provides non-deterministic latency.
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* Implementation dependent, but use of a broker is likely to make latency highly dependent on throughput. Small protocol overhead benefits throughput, but the use of broker limits this to what a single broker can relay.

F.6.5 IMPLEMENTATION VIEWPOINT	
F.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data-object size?* MQTT does not provide a notion of data objects or data-object caching, only the notion of a topic. The number of topics supported by a MQTT-Server will depend on the server memory and the number of clients. The number of clients will be limited by the capabilities of the MQTT broker and the number of connections it can sustain. There are no explicit message-size limits in MQTT, since it runs over TCP. MQTT-SN messages are over UDP will limit message size to what it can fit in a network datagram (64KB).
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* MQTT can effectively support interface evolution for an increasing number of distributed application components, since the message are opaque. However, the applications will need to manage the message versioning and evolution.
F.6.5.5 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* MQTT-Server does not support continuous availability over a defined system-relevant time period. The single point of failure introduced by the MQTT-Server will impact availability.
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* MQTT does not support recovery when fault conditions occur. Broker health should be monitored to ensure system availability. There should be mechanisms in place to re-start the broker in case of malfunction or failure.
F.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* MQTT is available for a number of platforms. Open-source implementations are available and could be built for target platforms.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* MQTT can facilitate incremental upgrades since it is built on the publish-subscribe data exchange pattern. During deployment, the main requirement is to configure all clients to connect to the same broker. The centralization of the configuration on the broker simplifies deployment but it requires provisioning and maintenance of a service that is separate from all client applications that is common to all. Integrating separate applications developed using different brokers requires consolidation of the brokers.

F.6.5 IMPLEMENTATION VIEWPOINT	
F.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* Hub and spoke. The broker (MQTT-Server) is the hub. All messages flow via the broker.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* MQTT can span globally over the WAN, as long as the broker is accessible via TCP.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* Segmentation is tied to the MQTT-Servers. Clients connected to different servers are segmented from each other.

Annex G ASSESSMENT TEMPLATE: LWM2M

This annex contains the assessment template for OMA Lightweight M2M protocol.

G.6.1 GENERAL INFO	
Name	*Common and formal name of the connectivity technology.* LwM2M (Lightweight Machine to Machine)
Contacts	*Responsible standards development organization (SDO), task group or author(s), respective companies and email addresses.* OMA SpecWorks.
Description	*Short synopsis of the technology*. LwM2M is a connectivity, device and application management framework designed for diverse IoT applications. The framework is designed specifically to be suitable for constrained deployments with limited device and network capabilities. Devices can be constrained in different ways, e.g. memory, power supply, CPU power and connectivity. Connectivity constraints can be related to bandwidth, coverage and reliability of the connection. LwM2M is based on open internet standards and principles. LwM2M addresses lifecycle management support of deployed devices and their applications. Both transport- and application layer security is provided. Syntactic and semantic interoperability is provided using standard technologies and an open extensible data model.
Application Domain(s)	*Application domains targeted by the connectivity technology.* LwM2M is suitable for a range of applications across industrial, enterprise, consumer and public IoT, e.g. automation for manufacturing, buildings, process control, energy, agriculture, transport, logistics and smart cities.
Dependencies	*Possible commonalities with or reliance on other connectivity elements.* Current technology mapping options include: • TCP for the transport/network layers • UDP for transport/network layers • DTLS/TLS for security • OSCORE for application layer security[81] • CoAP for message transfer • HTTP for message transfer • MQTT for message transfer
References	Website[82] *and other useful links to the technology.*

[81] See [IETF-RFC8613]

[82] See [OMA-LWM2M]

G.6.2 BUSINESS VIEWPOINT	
G.6.2.1 Purpose *(Section 6.2.1)*	*Give the general motivation and expectation for the connectivity technology. This section provides the business rationale. It communicates the fundamental "why and what" for the project.* LwM2M is a device management framework designed for sensor networks and machine to machine communications over a variety of networks. It provides interfaces for secure bootstrap and registration of devices (LwM2M clients) so that devices can connect to the management server in a secure and cost-efficient manner. In addition, the standard provides interfaces for remote management of devices (e.g. updating firmware images) and service enablement (e.g. configuring the right access control mechanisms). The standard also provides mechanisms to gather information (sensor measurements) from devices. While LwM2M is designed for constrained environments, it is equally applicable to any type of IoT scenario. The protocol is agnostic of the industry domain. LwM2M specifies a simple four-tuple resource model and provides capabilities for versioning and discovery of the resource model.
G.6.2.2 Pedigree *(Section 6.2.2)*	*Describe the derivation, origin or history of the system. The objective is to understand the brief evolutionary context of this technology.* The first version of the LwM2M protocol, version 1.0, was published in 2013. Version 1.1. was published in 2018. The latest version, version 1.2., was published in 2020. The CoAP protocol was published by IETF in 2014. The LwM2M protocol is developed by OMA SpecWorks, which was formed in 2018 when the Open Mobile Alliance (OMA) and the Internet Protocol for Smart Objects (IPSO) Alliances merged. Prior to the merger, OMA had developed successful device management solutions for decades.
G.6.2.3 Variants *(Section 6.2.3)*	*Describe the options and variants from the original generic description of the technology.* The LwM2M specification defines a single profile. However, the specification contains multiple transport and encoding options.
G.6.2.4 Maturity *(Section 6.2.4)*	*Estimate the technology maturity, state of development and condition relative to perfection. How refined are the connectivity concepts, requirements and demonstrated capabilities? Is the technology consistent and uniform?* LwM2M products and services are produced by more than 20 different companies from across tier-one chip manufacturers, cloud providers, telecom vendors and operators. LwM2M is based on open internet technologies. Most of those technologies have been around for a long time and are used in a wide range of applications. There are SDKs that can be used to build LwM2M systems.
G.6.2.5 Stability *(Section 6.2.5)*	*Describe whether the connectivity technology has been in use for long enough that most of its initial faults and inherent problems have been removed or reduced; how easy is it to use for both non-experts and professionals? Has there been a reduction in the rate of new breakthrough advances related to it?* The LwM2M specification is stable. The latest version was published 2020. Each new version contains new functionality but remains backward compatible with previous versions.

G.6.2 BUSINESS VIEWPOINT	
G.6.2.6 Standards Body *(Section 6.2.6)*	*List the relevant organizational bodies developing, coordinating, promulgating, revising, amending, reissuing, interpreting or otherwise producing technical standards and guidelines intended to address the needs of the base of affected adopters.* OMA SpecWorks is responsible for developing the LwM2M framework. However, LwM2M relies on open internet technologies that have been and are developed by the Internet Engineering Task Force (IETF), which is a membership-free standards organization.
G.6.2.7 Openness *(Section 6.2.7)*	*Is it an open standard? Who can participate? Are the specifications freely available? Are open-source implementations available? Does it require any single component from any single vendor?* LwM2M is specified by OMA SpecWorks. Participation in the standardization work requires a membership. Access to the published specifications is freely available to anyone, and the implementation and usage of the specifications is royalty-free. LwM2M relies on protocols and technologies specified by IETF. Participating in the IETF standardization work is open to anyone and does not require a membership. The incorporated IETF protocols are not specific to LwM2M, and many of them have been widely adopted also as part of other frameworks. This ensures mass marked adoption, interoperability and technology longevity. Open source and commercial implementations are available. The LwM2M specifications do not rely on any single component from any single vendor.

G.6.3 USAGE VIEWPOINT	
G.6.3.1 Architecture *(Section 6.3.1)*	*Summarize the main concepts, and high-level architecture, and terminology. Describe the end-to-end information exchange path.* LwM2M consists of multiple LwM2M clients connected to a LwM2M server. A separate LwM2M bootstrap server is also provided, for client bootstrapping and lifecycle management. LwM2M requests and responses are sent between the LwM2M Client and LwM2M Server in both directions, depending on the use-case. Requests are addressed to a specific LwM2M Object in the Client or the Server, using a URI associated with the Object.
G.6.3.2 Technology Options *(Section 6.3.2)*	*List the choices to be made for using the connectivity technology in a system.* • Selection of SDK used to implement LwM2M clients and servers. • Selection of the security mechanism. • Selection of the message encoding. • Selection of the message transport.
G.6.3.3 Applications *(Section 6.3.3)*	*A general statement of the typical applications that rely on this connectivity technology and the reason for using the connectivity technology.* LwM2M is based on concepts, protocols and security mechanisms that have been successfully used on the web for a long time, and that have been designed and maintained by a large community. It does not require any network middleware, function or other infrastructure for message routing between clients and servers over those available for web applications in general. Typical LwM2M applications are devices that exchange data with a network server. LwM2M is designed for, but not limited to, resource-constrained devices.
G.6.3.4 Typical Usage *(Section 2.2)*	*What function or where in the system this technology is typically used?* LwM2M was designed for, but is not restricted to, device management, application management and application data control (sensing, actuation) in constrained environments (devices and networks).
G.6.3.5 Operations *(Section 2.3.8)*	*Can one monitor, manage, and dynamically replace elements of the communication function?* Monitoring can be performed by network monitoring functions that support LwM2M or by monitoring functionality in the LwM2M server. CoAP network proxies can also be used for monitoring.
G.6.3.6 Security *(Section 2.3.5)*	*What are the system security implications of this connectivity technology?* LwM2M supports device bootstrapping, a procedure where a device retrieves credentials (security credentials and address information) needed to establish contact with one or more LwM2M servers. Later, the bootstrapping procedure can be used to update the security credentials, or to redirect a device to another LwM2M server. LwM2M defines a dedicated bootstrap server to which the device establishes contact to fetch the credentials. This procedure is referred to as device-initiated bootstrap. The credentials for establishing contact with the bootstrap server is hardcoded during device manufacturing. As an alternative to using a bootstrap server, the credentials can be hardcoded on the device during manufacturing, referred to as factory bootstrap, or they can be provided e.g. using a SIM card. However, in that case of hardcoded credentials the updating of the credentials would require physical access to the device. LwM2M supports application-layer encryption using OSCORE.

G.6.3 USAGE VIEWPOINT	
G.6.3.7 Safety *(Section 2.3.9)*	*For systems that need it, are certifiable implementations available?* N/A
G.6.3.8 Gateways *(Section 3.3)*	*List of gateways to core connectivity standards and other relevant connectivity technologies.* • oneM2M specifies interworking with LwM2M. • There is ongoing work in IETF to specify interworking between CoAP and HTTP.

G.6.4 FUNCTIONAL VIEWPOINT	
G.6.4.1 Core Framework Layer Functions	
Data Resource Model *(Section 4.1.1)*	*Does it provide a data resource model? Summarize the salient aspects.* LwM2M resources are called *objects* and *resources*. An object can contain one or many resources and multiple instances of a given resource. Each object and resource instance can be individually addressed using a URI. Objects and resources are used to manage both connectivity, devices and applications, as well as application data read/write.
ID & Addressing *(Section 4.1.2)*	*Does it provide a way to identifying and addressing data objects? Summarize the identification and addressing scheme.* In the REST model, all content is treated as a resource. A REST server provides access to the resources that it hosts, and a REST client can read, write and update the resources. LwM2M uses the REST model for identification of objects and resources. Each object and resource are presented as a resource. Standardized URI schemes are used to identify resources, and each Object and Resource is assigned a unique, hierarchical URI. Addressing of objects and resources require knowledge of the IP address and port of the LwM2M devices that host them. A LwM2M Server becomes aware of the IP address and port of a LwM2M Client when the client registers with the server.
Data Type System *(Section 4.1.3)*	*Does it provide a data type system? Summarize the salient aspects.* LwM2M defines a standard template for objects and resources. For resource values, there is a pre-defined set of value type options. A value can either be a simple data type (e.g. integer, string or Boolean) or a link to another LwM2M Object.
Data Resource Lifecycle (CRUD) *(Section 4.1.4)*	*Does it provide a means of managing a data object's lifecycle? Summarize the salient aspects.* LwM2M provides the means to create, read, update and delete data objects.
State Management *(Section 4.1.5)*	*Does it provide a means to manage the recent history of data objects? Summarize the salient aspects.* CoAP does provide cache mechanisms for proxies and CoAP clients to maintain responses of previous requests. These responses may contain the representations of the resources or links of where the resources are located. The CoAP specification defines mechanisms for determining the freshness of the caches and provides rules for access control and applicability of a cached response.
Publish-Subscribe *(Section 4.1.6)*	*Does it provide a means to publish and subscribe the state of data objects? Summarize the salient aspects.* A LwM2M Server can enable publish-subscribe functionality by subscribing to values of resources hosted by LwM2M clients, and then propagate resource value changes across other LwM2M clients.
Request-Reply *(Section 4.1.7)*	*Does it provide a means to request the state of data objects? Summarize the salient aspects.* This is the main communication pattern in LwM2M.
Discovery *(Section 4.1.8)*	*Does it provide a means to discover the data objects? Summarize the salient aspects.* When a LwM2M client registers with the LwM2M server, the client informs the server about the objects supported by the client and how to access them. Once a LwM2M client has registered itself with a LwM2M server, the server can perform a *discover* operation to discover what resources are implemented in a specific object.

G.6.4 FUNCTIONAL VIEWPOINT	
Exception Handling *(Section 4.1.9)*	*Does it provide a means to handle exceptions when quality of service or connectivity violations happen? Summarize the salient aspects.* LwM2M relies on the exception procedures defined for the transport protocols. Flags can be raised when there are exceptions related to quality of service or connectivity or if a transport protocol violation occurs.
Data Quality of Service (QoS) *(Section 4.1.10)*	*Does it support data QoS? Summarize the scope and coverage. Highlight the salient aspects.* LwM2M inherits the QoS capabilities of CoAP and the underlying network layers protocols. When using an unreliable network layer protocol (e.g. UDP), CoAP supports two levels of message delivery QoS: *confirmable* (re-transmission and acknowledgement) and *non-confirmable* (best effort, without acknowledgement).
Data Security *(Section 4.1.11)*	*Does it provide a data-object security model? Summarize the salient aspects.* LwM2M supports access control, where a LwM2M client can specify what operations a LwM2M server can perform on a given object supported by the client. If a LwM2M client is registered with multiple LwM2M servers, the client can specify different access control policies for each LwM2M server.
API *(Section 4.1.12)*	*Is there a standard API? Which programming languages is it available for?* LwM2M does not specify a programming language API. There are software development kits for, for example, Java and C programming languages.
Governance *(Section 4.1.13)*	*Does it standardize the mechanisms for configuration, administration, and monitoring? Summarize the salient aspects.* LwM2M provides a rich set of mechanisms for configuration, administration, monitoring and lifecycle management of LwM2M clients, and for LwM2M servers to monitor clients using event notifications.

G.6.4 FUNCTIONAL VIEWPOINT	
G.6.4.2. Core Transport Layer Functions	
Messaging Protocol *(Section 5.1.1)*	*Does it require UDP or TCP? What are the salient aspects of the messaging protocol? What are the message size limitations? What are the usage assumptions? Is it optimized for certain message requirements?* The LwM2M specification defines usage of both UDP and TCP. The LwM2M specification does not define message-size limitations. The normal message-size considerations for UDP and TCP apply.
Communication Modes *(Section 5.1.2)*	*Which communication modes does it support?* LwM2M supports IP-based and non-IP based transport mechanisms. The IP-based transport mechanisms rely on unicast over UDP or TCP. LwM2M supports the following IP-based transport mechanisms: CoAP, HTTP and MQTT.
Endpoint Addressing *(Section 5.1.3)*	*Describe the transport endpoints. How are the endpoints addressed? What are the limitations, if any, on the number of endpoints?* When IP-based transports are used, endpoint are addressed by an IP address and port number.
Connectedness *(Section 5.1.4)*	*Does it require a connected circuit between the endpoints? Summarize the salient aspects.* LwM2M does not require a connection-oriented transport. However, connection-oriented transport using TCP is supported. LwM2M also supports non-IP-based transports.
Prioritization *(Section 5.1.5)*	*Does it provide a means to prioritize messages? Summarize the salient aspects.* LwM2M does not support prioritization of messages. However, implementations might prioritize the order in which they process requests.
Timing & Synchronization *(Section 5.1.6)*	*Does it provide the ability to synchronize time? Summarize the salient aspects.* LwM2M does not provide the ability to synchronize time.
Message Security *(Section 5.1.7)*	*Does it provide mechanisms for message security? Summarize the salient aspects.* For UDP transport, DTLS provides message security. For TCP transport, TLS provides security. Independent of the transport mechanism, the Object Security for Constrained RESTful Environments (OSCORE) provides application-layer message security.

G.6.5 IMPLEMENTATION VIEWPOINT	
G.6.5.1 System Architecture Considerations	
Peer-to-Peer vs. Broker: *(Section 4.2.1.1)*	*Does the connectivity framework require running a special process or broker?* LwM2M Clients do not communicate directly with each other. Instead, they always communicate with a LwM2M Server. For LwM2M transport over CoAP and HTTP, the communication between the LwM2M client and the LwM2M server is peer-to-peer, and no special processes or brokers are required. For LwM2M transport over MQTT, an MQTT broker (MQTT server) is required. The LwM2M server can be co-located with the MQTT broker, or it can be located in a separate entity, in which case the client-server communication always traverses the MQTT broker entity.
Data-Centric vs. Device/App-Centric: *(Section 4.2.1.2)*	*Does the application code (or business logic) have to be aware of the other endpoints in order to participate in information exchange?* LwM2M servers need to be aware of LwM2M clients to exchange information. LwM2M clients will get the IP address and port of LwM2M servers during the bootstrapping process. If a bootstrap server is used for bootstrapping the IP address and port of the bootstrap server might be factory pre-configured. During the registration process, LwM2M clients will register themselves with a LwM2M server. After that the server can used the discover command to get a list of all objects, object instances and resources supported by the client. For LwM2M transport over MQTT, the LwM2M client needs to be aware of the MQTT broker.
Explicit vs. Implicit Governance: *(Section 4.2.1.3)*	*Is the governance explicit and shareable?* LwM2M provides mechanisms to version control the resource model including LwM2M objects, standardized Internet Protocol for Smart Objects (IPSO) objects and also vendor specific IPSO objects. A schema definition file (called DDF) captures these details. Hence governance is explicit and shareable.
G.6.5.2 Data Considerations	
Content-Based Selection *(Section 4.2.2.1)*	*Can a content-filter specify the data subset of interest?* A LwM2M server initiates observation requests for specific client resources that it wants to observe. The server can provide notification conditions associated with an observation. Such conditions can be related to the frequency of notifications, whether the observed value reaches a threshold value etc.
Time-Based Selection *(Section 4.2.2.2)*	*Can sub-sampling specify the data subset of interest?* A LwM2M server can indicate the frequency of observation notifications sent from a LwM2M client.
G.6.5.3 Performance Considerations	
Real-Time *(Section 4.2.3.1)*	*Does the connectivity technology support real-time data distribution? Is the latency deterministic (smaller jitter is better)?* Current transport mappings rely on TCP or UDP, both of which have non-deterministic latency.

G.6.5 IMPLEMENTATION VIEWPOINT	
Latency and Jitter vs. Throughput *(Section 4.2.3.2)*	*How does the latency and jitter change with throughput? What limits the throughput?* Latency and jitter are implementation specific and inherit the characteristics of the underlying transport. A LwM2M server with a large number of connected devices streaming data could potentially be a bottleneck for throughput, thereby increasing latency and jitter variance when loaded.
G.6.5.4 Scalability Considerations	
Data Objects *(Section 4.2.4.1)*	*Can the connectivity framework effectively handle an increasing number of data objects? What limits data-object size?* LwM2M can handle an increasing number of data objects. The number of data objects hosted by an endpoint, and the size of the hosted data objects, is limited by the memory in the endpoint.
Apps *(Section 4.2.4.2)*	*Can the connectivity framework effectively support interface evolution for an increasing number of distributed application components?* LwM2M is designed based on RESTful principles. Data and functions are represented as resources and functions are implemented by performing actions (e.g. read and write) on those resources. There is a pre-defined set of methods (e.g. GET, PUT and POST) for performing such actions. New methods can be defined if needed.
G.6.5.5 Availability Considerations	
Redundancy *(Section 4.2.5.1)*	*Can the connectivity framework support continuous availability over a defined system-relevant time period?* LwM2M does not place constraints or define how availability of the system is guaranteed. Server-side availability can be greatly increased by configuring multiple redundant LwM2M servers on the client.
Recovery *(Section 4.2.5.2)*	*Can the connectivity framework support recovery when fault conditions occur?* LwM2M relies on the transport layer mechanisms to indicate faults. However, LwM2M does specify mechanisms for retries and also suggests mechanisms to use COAP-confirmable messages at least once every 24 hours to ensure the server is receiving data.
G.6.5.6 Deployment Considerations	
Platforms Constraints *(Section 4.2.6.1)*	*Does the connectivity framework support the operating system (OS), the CPU and the resource constraints on the platform(s) being used?* LwM2M implementations are available for a variety of operating systems, CPUs and resource constraints.
Incremental Upgrades *(Section 4.2.6.2)*	*Does the connectivity framework facilitate incremental upgrades?* LwM2M clients and LwM2M servers can be updated independently.

G.6.5 IMPLEMENTATION VIEWPOINT	
G.6.5.7 Network Layer Considerations	
Topology *(Section 5.2.1.1)*	*What network topologies are allowed?* A LwM2M client can connect directly to any LwM2M server. LwM2M is agnostic to network topologies.
Span *(Section 5.2.1.2)*	*What is the span of the transport: LAN vs. WAN?* LwM2M inherits the LAN and WAN capabilities of CoAP, HTTP and MQTT and can be used within the LAN or across the WAN. LwM2M clients and servers can be in either the LAN or WAN.
Segmentation *(Section 5.2.1.3)*	*Can the transport support multiple independent and isolated communication paths between the same network endpoints?* LwM2M inherits the communication path properties of CoAP, HTTP and MQTT and supports multiple independent and isolated communication paths between the same network endpoints.

Annex H ACRONYMS

API	Application Programming Interface
CoAP	Constrained Application Protocol
DDS	Data Distribution Service
DDSI-RTPS	Data Distribution Service Interoperability Wire Protocol (DDSI)–Real-Time Publish-Subscribe Protocol (RTPS)
DHCP	Dynamic Host Configuration Protocol
DNP	Distributed Network Protocol
DTLS	Datagram Transport Layer Security
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IIC	Industry IoT Consortium
IIoT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
kHz	Kilohertz
MAC	Media Access Control layer
MIB	Management Information Base
MTU	Maximum Transmission Unit
NAN	Neighborhood Area Network
ND	Neighbor Discovery
OMG	Object Management Group
OSI	Open Systems Interconnection
OT	Operational Technology
PHY	Physical Communications Layer
QoS	Quality of Service
REST	Representational State Transfer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language
X-Types	Extensible Data Types

Annex I REFERENCES

[Clark-2017] David D Clark: Designs for an Internet, 2017, retrieved at 2022-06-04 download at *https://groups.csail.mit.edu/ana/People/DDC/ebook-arch.pdf*

[CoAP] CoAP: RFC 7252 Constrained Application Protocol, retrieved 2022-06-04 *http://www.coap.technology*

[CoAP-Impl] CoAP: Implementations, retrieved 2022-06-04 *http://www.coap.technology/impls.html*

[Fielding-2000] Fielding, Roy Thomas: Chapter 5: Representational State Transfer (REST) Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine, 2000, retrieved at 2022-06-04 download at *http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm*

[GOO-PB] Google Developers: Protocol Buffers, retrieved 2022-06-04 *https://developers.google.com/protocol-buffers/*

[HTTPWG] IETF HTTP Working Group home page, retrieved 2022-06-04 *http://httpwg.org*

[IETF] Internet Engineering Taskforce (IETF), retrieved 2022-06-04 *https://www.ietf.org/*

[IETF-RFC768] Internet Engineering Task Force (IETF), Postel J.: RFC 768, User Datagram Protocol, 1980, retrieved 2022-06-04 *https://tools.ietf.org/html/rfc768*

[IETF-RFC793] Internet Engineering Task Force (IETF), Postel J.: RFC 793, Transmission Control Protocol, 1981, retrieved 2022-06-04 *https://tools.ietf.org/html/rfc793*

[IETF-RFC1122] Internet Engineering Task Force (IETF), Braden R.: RFC 1122, Requirements for Internet Hosts -- Communication Layers, 1989, retrieved 2022-06-04 *https://tools.ietf.org/html/rfc1122*

[IETF-RFC4279] Internet Engineering Task Force (IETF), Kronen P., Tschofenig, H.: RFC 4279, Pre-Shared Key Ciphersuites for Transport Layer Security (TLS), 2005, retrieved 2022-06-04 *https://tools.ietf.org/html/rfc4279*

[IETF-RFC7252] Internet Engineering Task Force (IETF), Shelby Z., Hartke K., Bormann C.: RFC 7252, The Constrained Application Protocol (CoAP), 2014, retrieved 2022-06-04 *https://tools.ietf.org/html/rfc7252*

[IETF-RFC8613] Internet Engineering Task Force (IETF), Selander G., Mattsson J., Palombini F., Seitz L.: RFC 8614, Object Security for Constrained RESTful Environments (OSCORE), 2019, retrieved 2022-06-04
https://tools.ietf.org/html/rfc8613

[IIC-IINF] IIC: The Industrial Internet of Things Networking Framework, version 2021-Jul-19, retrieved 2022-06-04
https://www.iiconsortium.org/pdf/Industrial_Internet_Networking_Framework.pdf

[IIC-IIRA] IIC: The Industrial Internet, Volume G1: Reference Architecture Technical Report, version 1.9, 2019-June-19, retrieved 2022-06-04
http://www.iiconsortium.org/IIRA.htm

[IIC-IISF] IIC: The Industrial Internet, Volume G4: Security Framework Technical Report, version 1.0, 2016-Sep-26, retrieved 2022-06-04
http://www.iiconsortium.org/IISF.htm

[IIC-IIV] IIC: The Industrial Internet of Things Vocabulary, version 2.3, 2020-Oct-05, retrieved 2022-06-04
http://www.iiconsortium.org/vocab/index.htm

[ISO-7498-1] ISO/IEC standard 7498-1:1994, retrieved 2022-06-04
http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip

[MQTT] MQTT protocol, retrieved 2022-06-04
http://www.mqtt.org

[MQTT-P] MQTT Use Cases, retrieved 2022-06-04
https://mqtt.org/use-cases/

[OASIS] OASIS: Advancing open standards for the information society, retrieved 2017-02-16
https://www.oasis-open.org

[OMA-LWM2M] OMA SpecWorks: Lightweight M2M (LWM2M), retrieved 2022-06-04
https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/

[OMG-IDL] Object Management Group, DDS Foundation: Interface Definition Language, retrieved 2022-06-04
https://www.omg.org/spec/IDL/

[OMG-DDS] Object Management Group, DDS Foundation: DDS Portal—Data Distribution Services, retrieved 2022-06-04
https://www.dds-foundation.org/

[OMG-DDS-DCPS] Object Management Group, Data Distribution Service, version 1.4, 2015 March, retrieved 2022-06-04
https://www.omg.org/spec/DDS/

[OMG-DDS-GUIDE] Object Management Group, DDS Guidebook, retrieved 2022-02-02
https://www.dds-foundation.org/dds-resources/

[OMG-DDSI-RTPS] Object Management Group: DDS Interoperability Wire Protocol (DDSI-RTPS), version 2.5, 2021 June, retrieved 2022-06-04
http://www.omg.org/spec/DDSI-RTPS/

[OMG-DDS-OPCUA] Object Management Group: OPC-UA/DDS Gateway, version 1.0, 2020 January, retrieved 2022-06-04
https://www.omg.org/spec/DDS-OPCUA/

[OMG-DDS-RPC] Object Management Group: RPC over DDS, version 1.0, 2017 April, retrieved 2022-06-04
http://www.omg.org/spec/DDS-RPC/

[OMG-DDS-SECURITY]Object Management Group: DDS Security, 2018 July, retrieved 2022-06-04
https://www.omg.org/spec/DDS-SECURITY/

[OMG-DDS-SPECS] Object Management Group: What's in the DDS Standard? Open International Data-Centric Connectivity Standard, retrieved 2022-06-04
https://www.dds-foundation.org/omg-dds-standard/

[OMG-DDS-TSN] Object Management Group: DDS-Time Sensitive Networks (DDS-TSN), retrieved 2022-06-04
https://www.omgwiki.org/ddsf/doku.php?id=ddsf:public:guidebook:06_append:01_family_of_standards:05_wip:ddstsn

[OMG-DDS-VERTICALS] Object Management Group: DDS in Other Standards, retrieved 2022-06-04
https://www.dds-foundation.org/dds-in-other-standards/

[OMG-DDS-WEB] Object Management Group: Web-Enabled DDS, version 1.0, 2016 February, retrieved 2022-06-04
http://www.omg.org/spec/DDS-WEB/

[OMG-DDS-XRCE] Object Management Group: DDS For Extremely Resource Constrained Environments, version 1.0, 2020 February, retrieved 2022-06-04
https://www.omg.org/spec/DDS-XRCE/

[OMG-DDS-XTYPES] Object Management Group, Extensible and Dynamic Topic Types for DDS, version 1.3, 2020 February, retrieved 2022-06-04
https://www.omg.org/spec/DDS-XTypes/

[ONEM2M] OneM2M: Standards for M2M and the Internet of Things, retrieved 2022-06-04
http://www.oneM2M.org

[ONEM2M-27] OneM2M: TR-0027, DDS usage in oneM2M system, Draft Technical Reports, version 0.1.0, 2016-08-07, retrieved 2022-06-04
https://www.onem2m.org/component/rsfiles/download-file/files?path=Draft_TR%255CTR-0027-DDS_usage_in_oneM2M-V0_2_0.doc&Itemid=238

[ONEM2M-PS] OneM2M: Published Specifications, retrieved 2017-02-16
https://www.onem2m.org/technical/published-specifications

[OPC-CS] OPC Foundation: Case Studies, retrieved 2022-06-04
https://opcfoundation.org/resources/case-studies/

[OPC-DDS] OPC Foundation: OPC Foundation and Object Management Group (OMG) Announce Collaborative Strategy for the OPC UA and DDS Connectivity Standards, 2016-04-06, retrieved 2022-06-04
https://opcfoundation.org/wp-content/uploads/2016/04/OPCF-OPCUA-OMG-DDS-Positionspaper-final-v1.pdf
from *https://opcfoundation.org/news/press-releases/opc-foundation-and-object-management-group-omg-announce-collaborative-strategy-for-the-opc-ua-and-dds-connectivity-standards/*

[OPC-MEM] OPC Foundation: Members, retrieved 2022-06-04
https://opcfoundation.org/members

[OPC-UA] OPC Foundation: OPC Unified Architecture, retrieved 2022-06-04
https://opcfoundation.org/about/opc-technologies/opc-ua/

[Tolk-2007] Tolk, Andreas and Diallo Y. Saikou, and Turnitsa, D. Charles: Applying the Levels of Conceptual Interoperability Model in Support of Integrability, Interoperability, and Composability for System-of-Systems Engineering, Journal of Systems, Cybernetics and Informatics, 2007, retrieved 2022-06-04
Download at *http://www.iiisci.org/journal/cv$/sci/pdfs/p468106.pdf*

[W3C] World Wide Web Consortium (W3C)
https://www.w3.org/

[W3C-WSA] World Wide Web Consortium (W3C): Web Services Architecture, W3C Working Group Note 11 February 2004, retrieved 2022-06-04
https://www.w3.org/TR/ws-arch/

[WKPD-CI] Wikipedia: Conceptual Interoperability, retrieved 2022-06-04
https://en.wikipedia.org/wiki/Interoperability

[WKPD-IPS] Wikipedia: Internet Protocol Suite, retrieved 2022-06-04
https://en.wikipedia.org/wiki/Internet_protocol_suite

[WKPD-OSI] Wikipedia: OSI-Model, retrieved 2022-06-04
https://en.wikipedia.org/wiki/OSI_model

[WKPD-REST] Wikipedia: Representational state transfer (REST), retrieved 2022-06-04
https://en.wikipedia.org/wiki/Representational_state_transfer

[WKPD-WS] Wikipedia: Web Service, retrieved 2022-06-04
https://en.wikipedia.org/wiki/Web_service

AUTHORS & LEGAL NOTICE

This document is a work product of the Industry IoT Consortium Connectivity Task Group, chaired by Dr. Rajive Joshi (RTI) and Christer Holmberg (Ericsson).

Authors: The following persons contributed substantial written content to this document: Rajive Joshi (Lead, RTI), Paul Didier (Cisco), Christer Holmberg (Ericsson), Jaime Jimenez (Ericsson), Timothy Carey (Nokia).

Contributors: The following persons contributed valuable ideas and feedback that significantly improved the content and quality of this document: Bob Gessel (Ericsson), Tony Hodgson (Synapse Wireless), Matthew Gilmore (Itron), Edward Eckert (Itron), Aron Semle (Kepware), Jeff Lund (Belden), Cliff Faurer (Enterprise Web), Christoph Gericke (Harting), Stefan Schönegger (BR Automation), Tom Rutt (Fujitsu), Jiyhe Lee (Samsung), Farooq Bari (AT&T), Gerardo Pardo-Castellote (RTI), Stan Schneider (RTI), Reinier Torenbeek (RTI), Brett Murphy (RTI), Norman Finn (Cisco), Kevin White (Distrix), Mark Crawford (SAP), Shi-Wan Lin (Thingswise), Aravind Parandhaman (NEC), Jeff Harding (ABB), Eric Harper (ABB), Brad Miller (GE), Marcellus Buchheit (Wibu-Systems), Jan Höller (Ericsson).

Editors: Rajive Joshi (Lead, RTI), Paul Didier (Cisco).

Technical Editor: Stephen Mellor (IIC staff) oversaw the process of organizing the contributions of the above Authors and Contributors into an integrated document.

We are also grateful to everyone who made comments on the draft version available to the Connectivity Task Group and thank in advance anyone who provides further constructive comments on the current version.

We acknowledge the work by the members of the Architecture Task Group for developing the Industrial Internet Reference Architecture and the Vocabulary Team in the Technology Working Group for maintaining the Vocabulary, both those are a companion document to this one.

Finally, we are grateful for the ongoing support of IIC Staff, whose diligence with supporting the tools and the mechanics of the document writing have made this process easier.