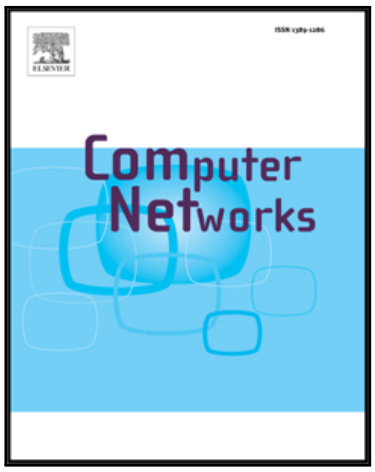


Survey, Comparison and Research Challenges of IoT Application Protocols for Smart Farming

Dimitrios Glaroudis , Athanasios Iossifides , Periklis Chatzimisios

PII: S1389-1286(19)30694-2
DOI: https://doi.org/10.1016/j.comnet.2019.107037
Reference: COMPNW 107037

To appear in: *Computer Networks*

Received date: 30 May 2019
Revised date: 27 September 2019
Accepted date: 26 November 2019

Please cite this article as: Dimitrios Glaroudis , Athanasios Iossifides , Periklis Chatzimisios , Survey, Comparison and Research Challenges of IoT Application Protocols for Smart Farming, *Computer Networks* (2019), doi: https://doi.org/10.1016/j.comnet.2019.107037

Survey, Comparison and Research Challenges of IoT Application Protocols for Smart Farming

Dimitrios Glaroudis, Athanasios Iossifides, Periklis Chatzimisios

International Hellenic University, Department of Information and Electronic Engineering, P.O. Box 141, GR-57400, Sindos, Thessaloniki, Greece

Abstract

Smart farming era has already begun and its societal and environmental implications are expected to be huge. In this context, the Internet of Things (IoT) technologies have become the major path forward towards novel farming practices. The unprecedented capability of data collection and management offered by IoT is based on several factors of the underlying communication network architecture and technology, one of the most important being the application level protocol that is used among IoT nodes, gateways, and application servers. This work offers an up-to-date survey of research efforts on the IoT application layer protocols, focusing on their basic characteristics, their performance as well as their recent use in agricultural applications. Furthermore, it provides a comparison among them, in terms of well-accepted key performance indicators and comments on their suitability in the framework of smart farming as well as the corresponding challenges that have to be faced towards their efficient implementation.

Keywords: Smart farming, Internet of Things, IoT application protocols, IoT messaging protocols, Publish-subscribe

1. Introduction

Emerging Internet of Things (IoT) technologies offer a great potential for novel solutions and smarter application development that can improve all aspects of the agricultural sector. Data collection has been made easier during the last decade due to recent advances of communication networks and protocols, mainly on the lower layers, i.e., the physical, link and network layers. However, in addition to these, the upper layer protocols are of major importance for efficient data collection and sharing. In the context of IoT, application layer protocols mainly refer to the lower part of the TCP/IP stack model's application layer that corresponds to the OSI session layer and they are also commonly referred as messaging protocols. The two main categories are the protocols that follow the request/response model, and those that follow the publish/subscribe model. They can either be used in parts only of the IoT communication architecture or in the whole of it, meaning that while an IoT application protocol can be used for communication between the IoT devices and an IoT gateway, other protocols may be used between the gateway and the cloud or the cloud and the end user. Similarly to all IoT application frameworks, smart farming applications are based on IoT application layer protocols for data transfer. Though, the nature and diversity of IoT agricultural applications which span a wide range of requirements both in the type of data and the environments that the IoT devices are installed raise major challenges related to the volume, variety, veracity and velocity of the data. In this context, this work is building upon recent research efforts and related surveys [1-14] to include all the advancements and new evaluation results of testing IoT application protocols during the past three years, so that to provide a fresh and consistent basis that will allow comparison of the major ones among them: the Message Queue Telemetry Transport (MQTT) protocol, the Constrained Application Protocol (CoAP), the Extensible Messaging and Presence Protocol (XMPP), the Advanced Message Queuing Protocol (AMQP), the Data Distribution Service (DDS) protocol, the Representational State Transfer Hypertext Transfer Protocol (REST HTTP) and the WebSocket protocol. This comparison is based on key performance indicators [3] and is combined with the latest research efforts in IoT applications in agriculture, to derive the lessons learned as well as the

issues and challenges that are still important and need to be considered for real-life agricultural applications.

While IoT in agriculture has already been in the center of smart farming research efforts for several years, the majority of works focus on the benefits and challenges of IoT in agriculture [15-16] or analyze IoT architectures, technologies and practices for smart agriculture ([7], [17]), focusing on hardware, platforms, sensors and wireless communication protocols [18-19]. There are no studies focusing explicitly on the evaluation of IoT application layer protocols in agricultural applications. Building upon recent surveys on IoT application protocols and comparative evaluations, e.g., [1-3] and [20] as well as recent IoT enabled agricultural applications' development, the contribution of this work is twofold: (i) to provide a general, up to date, survey of IoT application protocols, (ii) to focus on the use, requirements, evaluation and research challenges of IoT application protocols in smart farming based on suitable key performance indicators, i.e., latency, energy and bandwidth requirements, throughput, reliability, security as well as developers' preferences.

The next Section provides a description of the protocols under consideration and their main characteristics summarizing recent literature and the corresponding, per protocol, standards. Based on this description, a review of the recent literature and the work in [3], Section 3 provides a comparison of the protocols in terms of latency, bandwidth and energy requirements as well as throughput and reliability. Section 4 reviews the latest IoT applications in farming and Section 5 provides an evaluation and discussion on the use IoT application protocols in agriculture while Section 6 concludes our work.

2. Basic IoT application protocols

2.1. Message Queue Telemetry Transport (MQTT)

MQTT is a messaging protocol, originally introduced in 1999 by Andy Stanford-Clark (IBM) and Arlen Ninpper (Arcom) and later, its v3.1 version was standardized by OASIS [21-22]. It is based on the publish/subscribe model and is therefore suitable for devices with resource constraints and non-ideal networking conditions. Its simplicity and very small header size compared to other protocols make it one of the most important choices for IoT applications [1-3]. MQTT typically uses the TCP protocol at the transport level and its architecture includes two communication nodes: the clients and the servers/brokers (Figure 1). The clients can operate either as publishers or as subscribers or both to send and receive data. Messages are sent and received through the broker who acts as a distributor using topics. Every subscriber must subscribe to a topic to receive subsequent messages that the publisher has posted to this topic. The key messages being exchanged are: CONNECT/CONNACK to connect a client with the broker; SUBSCRIBE/SUBACK and UNSUBSCRIBE/UNSUBACK to subscribe/unsubscribe a client to/from a topic; PUBLISH/[PUBACK, PUBREC, PUBREL, PUBCOMP] for sending a message from a publisher to the broker or from the broker to a client when the last has been subscribed to the topic together with the associated confirmations, according to the provided Quality of Service (QoS).

MQTT defines three QoS levels: QoS 0, 1 and 2. QoS 0 (at most once) is the lowest level at which the message is not confirmed or stored by the receiver nor it is sent again by the transmitter ("fire and forget"). In QoS 1 (at least once) the message will arrive one or more times at the receiver which responds with a confirmation message. Finally, in QoS 2 (exactly once) each message is received once and only once by the receiver with confirmation. It is the safest but also the slowest level with a total of four messages being exchanged between the client and the broker. Obviously, reliability increases with the QoS level however the bandwidth and the energy consumed increase as well.

In their general format (Figure 2), MQTT messages consist of two bytes which are always present at the header. The Message Type specifies the type of message being exchanged, the DUP flag (duplicate) informs that the message has already been sent and therefore the receiver may have already received it; the QoS level field specifies the services' quality level (QoS) and the Retain field informs the broker to keep the last PUBLISH message and send it when a new subscriber enters the topic. The last field shows the length of the remaining message.

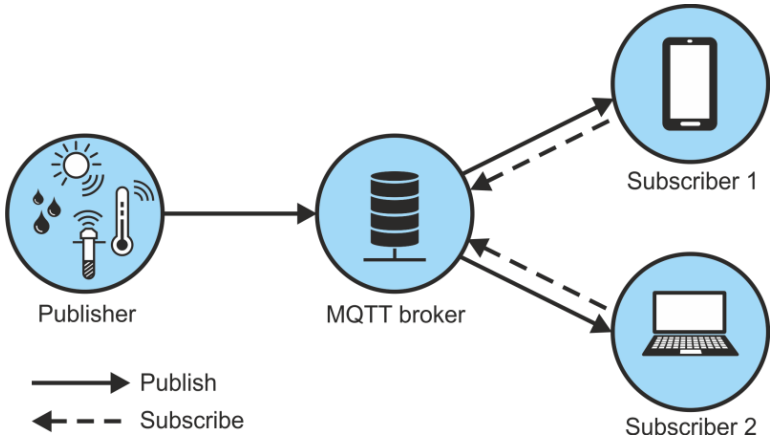


Fig 1. MQTT protocol architecture [1-2], [23].

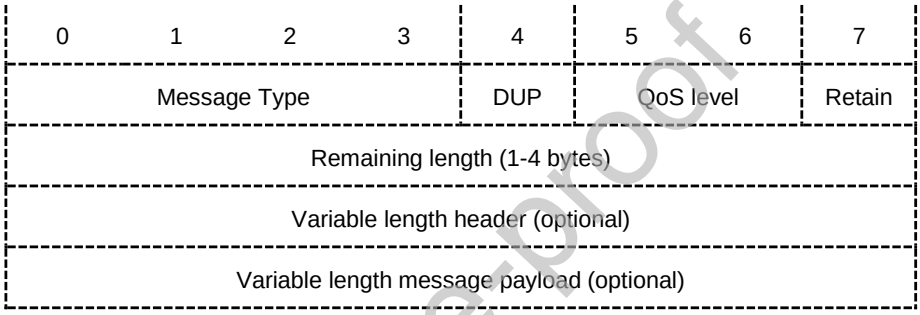


Fig 2. General format of MQTT messages [1-2], [21].

MQTT includes two more QoS features defined in the control package (CONNECT) from the client to the broker: the *Persistent session* and *Keep alive* features. When a client is disconnected (either voluntarily or by mistake) from the broker, normally the client must rerun SUBSCRIBE. Setting the cleanSession parameter in the CONNECT control packet to False creates a persistent session in which all subscriptions to topics, all new QoS 1 and 2 messages that the client has lost when offline and all QoS 2 messages which have not been confirmed by the client, are kept on the broker. With the "Keep alive" feature, the MQTT connection is maintained without exchanging messages for an interval (in seconds) reported by the client in the CONNECT control packet. If no information messages are exchanged during this interval, two PING type messages (at client's responsibility) are exchanged to maintain the connection live. In addition, MQTT includes a functionality of retaining the last message of a topic on the broker with the use of the RETAIN flag in the PUBLISH message. When a client subscribes to the topic, receives this message even though this has been earlier published.

In terms of security, MQTT messages may contain a username and a password in the variable part of the header for identification purposes. However, these values are not encrypted. In general, the Transport Layer Security (TLS) protocol [24] can be used for secure MQTT connections. An extension called Secure MQTT (SMQTT) has recently been proposed to deal with security issues [25].

Even though MQTT is designed to be lightweight, it has two drawbacks for very constrained devices. Every MQTT client must support TCP and will typically hold a connection open to the broker at all times. MQTT topic names are often long strings which make them impractical for some lower layer protocols. Both of these shortcomings are addressed by the MQTT-SN protocol, a light version of MQTT that has been recently proposed, specifically designed for sensor networks to allow less resources and power consumption [26]. It uses UDP instead of TCP and provides additional settings such as smaller payload size, broker support for indexing topic names (number entry instead of UTF-8 strings), etc.

Finally, a new version of the protocol, MQTT 5 [27], has been recently proposed for standardization by the OASIS consortium with various feature improvements and adjustments. The highlights of the new version include: better error reporting (a reason code has been added to responses to publications PUBACK/PUBREC and return codes are now present on all acknowledgements); shared subscriptions (if the message rate on a subscription is high, shared subscriptions can be used to balance the load of messages across a number of receiving clients); message properties in the form of metadata in the header of a message used to implement the other features in this list but also to allow user defined properties; message and session expiry (an option to discard messages if they cannot be delivered or a client does not connect within a user-defined period of time); limits like the maximum packet size and number of (QoS>0) messages which can be transmitted to inform the client what it is allowed to do, etc. The above enhancements improve in a great extent the capabilities of the protocol however their impact on the performance has not been yet evaluated.

2.2. Constrained Application Protocol (CoAP)

The CoAP protocol was created by the Internet Engineering Task Force CoRE (Contrained RESTful Environments) [28-29], targeting to be used by devices with limited computing capabilities [1-3]. It is a Web transfer protocol based on the Representational State Transfer (REST) architecture [30], similarly to HTTP (Figure 3). However, unlike HTTP, CoAP is designed to supports request/response functions in constrained environments. It is considered to be a "light" protocol in the sense that the overhead of the protocol in the headers and methods used is clearly less than many other application level protocols. CoAP packets are much smaller than HTTP TCP flows and bit fields and mappings from strings to integers are used extensively to save space. Packets are simple to generate and can be parsed in place without consuming extra memory in constrained devices. The total overhead is further reduced by using the UDP transport protocol instead of the TCP transport protocol. CoAP follows a client/server model. Clients make requests to servers, servers send back responses. Communication takes place through connectionless datagrams. Retries and reordering are implemented in the application stack. Removing the need for TCP allows full IP networking in small microcontrollers. When a CoAP client sends one or more requests to the server, the corresponding response is not sent to a pre-existing connection, but asynchronously using CoAP messages. On the other hand, the use of UDP reduces the reliability of the protocol, which however may be improved with appropriate protocol adjustments. IETF has recently published [31] an additional possibility of using CoAP over TCP. However, this version is still in its early stages.

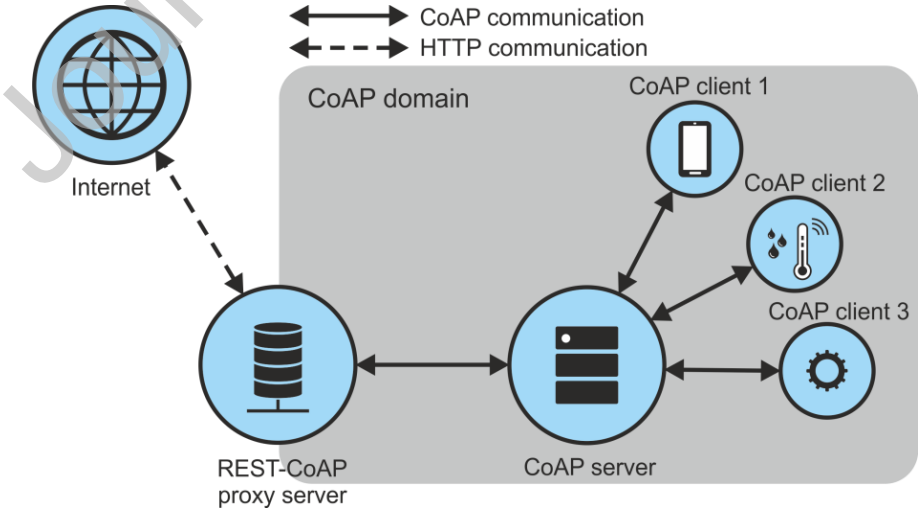


Fig 3. CoAP protocol architecture [1], [3], [29].

CoAP messages start with a 4-byte header (Figure 4) that includes the CoAP Version, the Type of message, the Token length, the request/response Code (analogous to HTTP status codes) and the Message ID that is used to identify duplicate messages and map messages with their acknowledgments. Then there is a Token with a length of 0 to 8 bytes which is used to map requests with responses. The typical CoAP message length is between 10 and 20 bytes.

The CoAP protocol is divided into two sub-layers; the lower one, called *message sublayer*, is responsible to manage the UDP layer, while the upper one, called the *request/response sublayer*, manages the communication.

The message sub-layer consists of four message types: CON (confirmed), NON (non-confirmable), ACK (acknowledgment) and RST (reset) and consists of two communication models: *Reliable Message Transport* and *Unreliable Message Transport*. In the first model the client transmits a CON message until a returning ACK message is received by the server with an ID identical to the original message. If the server cannot process the incoming message it returns RST instead of ACK. In the second model, the client sends a NON message and the server responds in the case of a problem with an RST message.

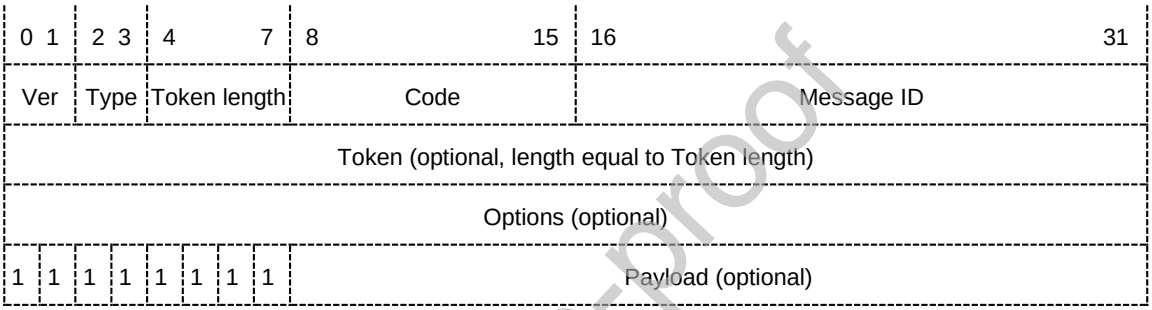


Fig 4. CoAP message format [1], [3], [29].

The request/response sub-layer implements the RESTful methodology that allows the use of methods similar to HTTP for sending messages to a server or retrieving messages from it, i.e., GET, PUT, POST or DELETE methods. Requests and responses are aligned with the use of a common token. Communication may take place using three approaches, that is, *piggy-backed*, *separate response*, and *non-confirmable request/response*. In piggy-backed option the client sends a message and the server answers with an ACK message, which includes the client response message using the original token sent by the client. In the event of an error, the ACK message includes an error code. In the separate response case the server receives a CON message and if it is unable to respond immediately, it sends an ACK message to the client. When it is ready to respond, the server sends a CON message to the client, which then responds with an ACK message. Finally in the non-confirmable request/response case, the client sends a NON message to the server and does not wait for an ACK while the server responds with another NON message to the client (with the same token).

The CoAP protocol has an optional feature that improves the request/response model by allowing clients to receive updates of a specific quantity (e.g., temperature) from the server using the option "observe" in the GET method used upon request [32]. In this case the server puts the client in the list of observers of the specified quantity and the client receives updates when this quantity changes. In this way communication is particularly close to the publish/subscribe logic. In addition, CoAP includes a standard mechanism for resource discovery. Servers provide a list of their resources (along with metadata about them) that allow a client to discover what resources are provided and what media types they are [3][32].

The CoAP protocol does not provide some internal security but may use the Datagram Transport Layer Security (DTLS) protocol at the transport level ([4], [33]) that provides the same assurances as TLS but for transfer of data over UDP. Four security cases are defined: *NoSec*, in which the DTLS is disabled and direct UDP is used without security; *PreSharedKey*, in which the device is pre-programmed with symmetrical keys where, each device has a key list and each key is used to communicate with a particular node or group of nodes; *RawPublicKey* where, the device uses a pair of asymmetric keys, that is, it has an

identity calculated from the public key and a list of other nodes identities with which it can communicate; *Certificates*, in which the device has a pair of asymmetric keys and a X.509 certificate. Although DTLS was not oriented to provide security features over constrained environments, its recently updated versions are focused on optimization for lightweight devices and some of them include IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) header compression mechanisms to compress DTLS header. Still, the optimization of DTLS for IoT remains an open issue [3]. It must be noted that CoAP does not include a key distribution and management process.

2.3. Extensible Messaging and Presence Protocol (XMPP)

The XMPP protocol started as Jabber by Jeremie Miller in 1999, offering initially an Instant Messaging (IM) service. Based on the Jabber protocol, IETF published the RFC 3920 and RFC 3921 (2004) which were then replaced, in 2011, by RFC 6120 [34], RFC 6121 [35] and RFC 6122 [36], the latter being replaced by RFC 7622 [37] in 2015. Additionally, the XMPP Standards Foundation (XSF) creates extensions of the protocol for various applications (XMPP Extensions Protocols- XEP); there are 402 XEPs at the moment.

XMPP can be used for messaging, chat, voice and video calls, etc., allowing all of these applications to provide authentication, access control, and encryption services [1-3]. Its basic architecture is described in Figure 5. It is a protocol based on text messages that uses XML (Extensible Mark-up Language) through which it can implement both request/response and publish/subscribe methods by using appropriate extensions [38]. With these extensions, XMPP entities can create topics and publish information. An event notification is then sent to all entities subscribed to that topic.

Clients and servers communicate using semantic structured data units (*stanzas*). The common attributes of stanzas are *from*, *to*, *type*, and *id*. Attribute *from* indicates the sender ID (Jabber IF – JID). When there is no content in *from*, the sender is considered to be the server, unless it is a server-to-server communication, in which case an error occurs. Attribute *to* indicates the recipient's JID. When there is no content, the server is considered to be the recipient. Attribute *Type* indicates the type of stanza and, finally, the *id* attribute is used in info/query stanzas to identify questions and/or answers.

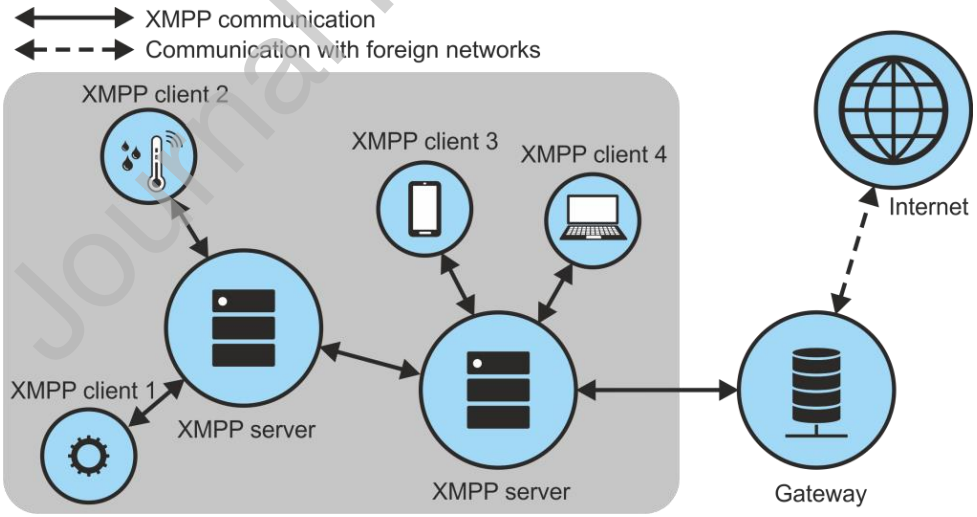


Fig 5. General architecture of XMPP protocol [1-2], [34].

Three types of stanzas are defined: *presence*, *message* and *iq* (the general form of their structure together with a simple example pre case is shown in Figure 6). Stanzas of type *presence* control and report the availability of an entity and perform the registration of an entity in the roster of another entity. The server forwards the information to all the entries of an entity's roster, i.e., the entities that are registered to this entity. *Message* type stanza is used to send data from entity to entity by specifying the title and the contents of the message. Message stanzas do not receive confirmation from the receiving entity being

either a client or a server. The stanza of type *iq* (*info/query*) creates couples of senders and receivers. It is used to receive information from the server or to send some settings. The mechanism is similar to the HTTP GET and POST methods. Each *iq* stanza has an ID that connects it to the answering stanza.

```
<stream>
        <presence   from='xxx@somewhere.gr'
                    to='yyy@elsewhere.gr'
                    type='subscribe'>
        </presence>
        <message    from='xxx@somewhere.gr'
                    to='yyy@elsewhere.gr'
                    type='chat'>
              <body>Hello there.</body>
              <thread>4fg43p09867ggsf4345</thread>
        </message>
        <iq     from='xxx@somewhere.gr
              type='get'
              id='question1'
              <query xmlns='jabber:iq:roster'/>
        </iq>
</stream>
```

Fig 6. Presence, message and iq **s**tanzas' examples in XMPP protocol [1-2], [39].

One of the most important features of XMPP is its security. Unlike other protocols such as MQTT and CoAP which are based on TLS and DTLS encryption that are not embedded in them, XMPP has a built in TLS mechanism to provide reliability in terms of confidentiality and data integrity. In addition, XMPP uses a special SASL (Simple Authentication and Security Layer) profile to identify entities.

The main disadvantage of XMPP is the use of XML language which leads to long messages that consume large bandwidth. A second issue is the lack of a QoS control process. In recent years, however, efforts have been made to adapt XMPP to IoT applications, such as in [40], where a lightweight version of XMPP is proposed with a publish/subscribe mechanism designed for IoT devices with limited resources.

2.4. Advanced Message Queuing Protocol (AMQP)

The AMQP protocol is designed to provide interoperability capabilities between different applications and systems, allowing the exchange of messages between different platforms, implemented in different languages. The AMQP protocol has been implemented with two different versions, namely the AMQP version 0.9.1 [41] and the AMQP 1.0 version [42] standardized by the OASIS consortium. AMQP 0.9.1 version follows the publish/subscribe model and it is realized with two basic entities in a broker: *exchanges* and *message queues*, as illustrated in Figure 7. Exchanges refer to that part of the broker used to route messages received from publishers to the appropriate queues, following predetermined rules and conditions. Message queues are the queues on which messages are routed and remain there until they are read by the corresponding subscriber. The newer version of AMQP protocol is not exclusively linked to the publish/subscribe mechanism. The protocol can directly link peer-to-peer entities without the need for broker mediation. In this way, the protocol becomes more flexible and can support different communication schemes, e.g., client-to-client, client-to-broker, and broker-to-broker.

The protocol requires a TCP connection and provides three QoS levels, in line with the MQTT protocol. QoS 0 does not require confirmation from the receiver to the transmitter. QoS 1 requires

confirmation of receipt of the message. The receiver sends an ACK, and if the publisher does not receive an ACK, it then sends the message again after a certain amount of time. QoS 2 guarantees that the message is delivered once and only once without repetitions. In addition, at the transport level, the communication is organized into frames (Figure 8). The first 4 bytes indicate the frame size while the DOFF (Data Offset) field shows the location of the main part within the frame. The Type fields indicate the type, format, and purpose of the frame.

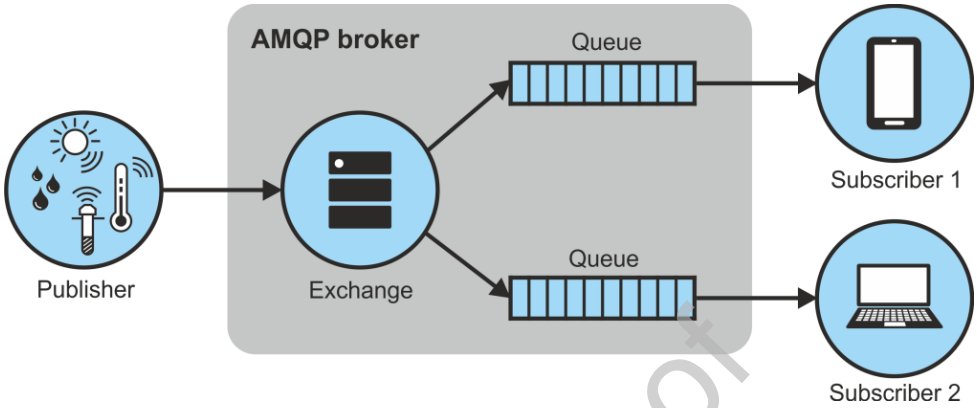


Fig 7. Publish/subscribe mechanism in AMQP protocol [1-2], [41].

Above the transport level, the protocol defines a message layer that includes two types of messages: *bare messages* and *annotated messages*, as shown in Figure 9. Bare messages are those sent by the sender and include the main part of the message (i.e., application data), the system properties (e.g., message ID, To, Subject, Reply to) and the properties of the application that implements the protocol. Annotated messages are the messages sent to the recipients after adding additional information and header to the original bare message.

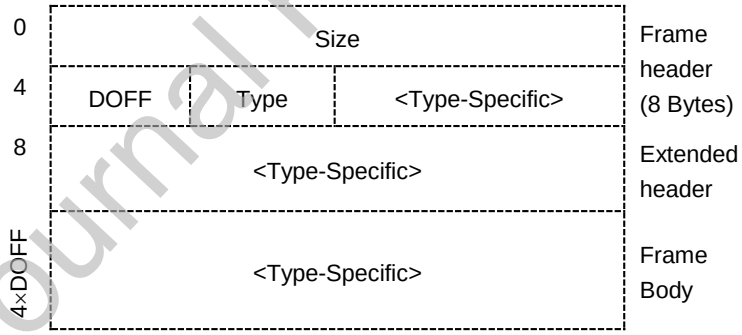


Fig 8. AMQP protocol frame structure [1-2], [41].

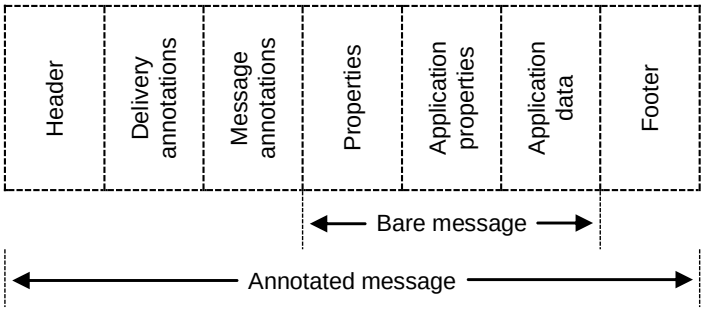


Fig 9. AMQP message format [1-2], [41].

AMQP integrates communications security and identification with the use of SASL and TLS. Overall, AMQP offers many capabilities, but it is a heavy protocol in terms of the network resources and the computational power of the nodes it requires. Because of this, its use within the Internet of Things framework is mainly confined between servers or strong autonomous nodes.

2.5. Data Distribution Service (DDS)

DDS is a real-time publish/subscribe protocol for M2M type communications developed by the Object Management Group (OMG) [43]. Unlike other publish/subscribe protocols, DDS is decentralized and based on peer-to-peer communication without the intermediary of a broker but using multicasting [1-3]. Thus, publishers and subscribers exchange data directly between them. A publisher can publish data even if there is no subscriber interested, and the use of data is basically anonymous, since publishers do not know who uses their data. One of the most important features of DDS is its scalability through the embedded discovery protocol that allows the subscribers to find out which publishers are present and determine the information they need and with what QoS. Another important feature of DDS is that it is data-centric. It is the type and content of the data that defines the communication as opposed to the message-centric protocols where the mechanisms and functions for data transfer are in the center. Grouping of data is based on their characteristics, e.g., the refresh rate [3].

The DDS architecture (Figure 10) defines two layers: the Data-Centric Publish-Subscribe (DCPS) layer, responsible for transferring information to subscribers and the Data-Local Reconstruction Layer (DLRL), an optional layer that facilitates the sharing of data between distributed objects. The basic entities of DDS are: the *Domain*, a virtual entity that allows seamless communication between nodes that share the same interests); the *Domain Participant* that associates publishers, subscribers and topics within a domain and it is the basic entity creation unit within a domain. The core entities include: the *Publisher* i.e., the sender of the data; the *Data Writer*, used by the publisher to send the data; the *Subscriber* that receives the data from the publisher and promotes it to the application; the *Data Reader*, controlled by the subscriber to read the data; and the *Topic* that is defined by the type and name of the data and connects data writers and data readers.

DDS uses UDP by default but can also support TCP. One of its benefits is the wide range of 23 available QoS policies that it offers. These policies manage the features of DDS, such as the discovery of distributed remote entities, the availability and transfer of data, the use of resources, etc. [44].

With respect to security, DDS provides several options. TLS or DTLS can be used depending on whether TCP or UDP is used. Due to the fact that these protocols introduce a heavy overhead, OMG has set up an architecture that is suitable for IoT applications. However, the issue of security is still open for DDS [45].

Overall, DDS can support both powerful and low-capacity simple devices and looks like a promising solution for IoT applications. However, as outlined in [3], it has not yet been adequately tested and open-ended implementations [46] can help towards this direction.

2.6. Representational State Transfer Hypertext Transfer Protocol (REST HTTP)

HTTP is the basic client/server protocol used on the Web for applications' development. Its most widely used version is HTPP/1.1 and is based on client/server communication that follows the request/response model. Recently HTPP has been associated with the REST architecture to facilitate interaction between different entities over web based services. The combination of HTTP and REST enables devices to make their status readily available in terms of the standardized CRUD (create, read, update, delete) functions [3]. The CRUD functions are mapped to the POST, GET, PUT and DELETE methods of HTTP, respectively. In this way, it is possible to develop a REST model for different IoT devices [47]. JSON is typically used to represent the data.

HTTP uses the TCP protocol at the transport layer. While this assures the transport of large volumes of data reliably, it is not optimized for resource constrained environments. One of the major issues is that in the context of IoT, nodes with limited resources send small amounts of data sporadically and installing a

TCP connection every time results in high delay and significant overhead. HTPP does not set any QoS levels. As far as security is concerned, HTPP uses TLS [25] to allow secure and encrypted communications, thus leading to its secure version known as HTTPS.

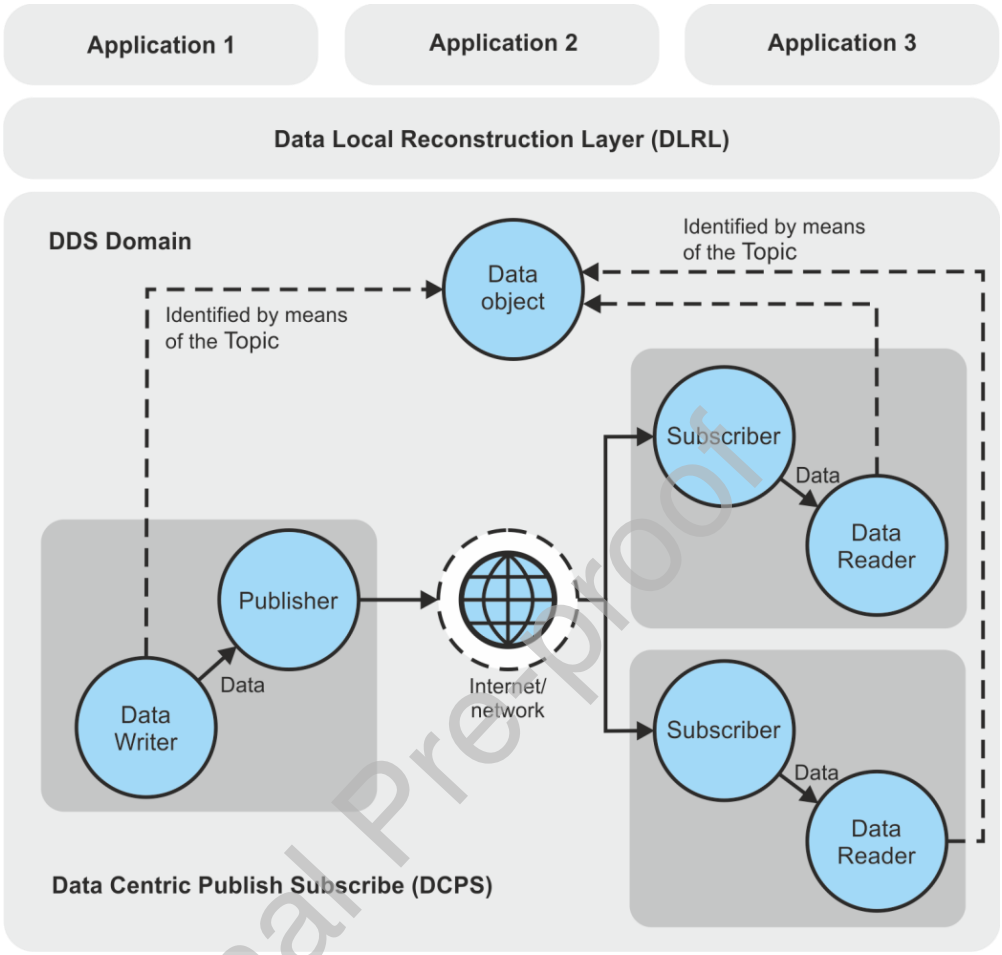


Fig 10. DDS conceptual architecture [1-2], [43].

Overall, the REST HTPP protocol is not yet sufficient for IoT devices communicating with the cloud because of its complexity, larger headers and the higher power consumption it requires. However, it is quite often used in experimental or other IoT implementations that do not have significant requirements and constraints because of its widespread use in web applications. The new version of the protocol, HTTP/2.0 [48] introduces improvements, some of which are very relevant to the IoT framework. It actually introduces a more efficient use of network resources and shorter total latency by using compressed headers and by exchanging multiple parallel messages over the same connection. Furthermore, it introduces the server push mode, in which the server can forward data to clients without waiting for requests. These features seem to be useful in IoT applications and remain to be realized and tested for their performance.

2.7. WebSocket

The WebSocket protocol was developed by an initiative within the HTML 5 framework to facilitate communications over TCP. It enables two-way communication between a client running untrusted code in a controlled environment with a remote host that has opted-in to communications from that code. It was designed to supersede existing bidirectional communication technologies that use HTTP and were implemented as trade-offs between efficiency and reliability so to benefit from existing infrastructure

(proxies, filtering, authentication), since HTTP was not initially meant to be used for bidirectional communication [49].

WebSocket does not follow either the response/request or the publish/subscribe model. The client initiates a handshake process with the server to install a WebSocket session with a message similar to the HTTP protocol. The protocol has two parts: a handshake process and the data transfer. Once the client and server have exchanged handshake, and if the handshake was successful, they can transfer messages (data) back and forth in a two-way communication where each side can send data independently (from the other) at its will. Messages are composed of one or more frames. Each frame has an associated type and frames belonging to the same message contain the same type of data. The WebSocket messages do not necessarily correspond to a particular network layer framing, as fragmented messages may be coalesced or split by an intermediary. However, after the handshake process, the HTTP headers are removed for the rest of the session and the client exchanges messages with the server in an asynchronous two-way (full-duplex) communication with an overhead of only 2 bytes (when masking is not used). The session terminates either from the server side or from the client when it is no longer required. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections.

Data framing (data is transmitted using a sequence of frames) is used to avoid confusing network intermediaries (such as intercepting proxies) and for security reasons. A client masks all frames that sends to the server (that masking is done whether or not the WebSocket Protocol is running over TLS) and the server closes the connection upon receiving a frame that is not masked and may send a “close” frame with a status code denoting protocol error. The server masks no frames that it sends to the client. A client closes a connection if it detects a masked frame.

The WebSocket Protocol is an independent TCP-based protocol. WebSocket connections are fully asynchronous, unlike HTTP/1.1 (synchronous) and HTTP/2 (asynchronous, but the server can only initiate streams in response to requests). With WebSocket, the client and the server can both send frames at any time without any restriction. It is closer to TCP than any of the HTTP protocols. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. By default, the WebSocket Protocol uses port 80 for regular WebSocket connections and port 443 for WebSocket connections tunneled over TLS. It is estimated that WebSocket subdivides the latency by a factor of three compared to HTTP. Although not designed for devices with strong constraints, it offers real-time communication, minimizes the overhead and may be a solution for IoT applications by using the WAMP sub-protocol (WebSocket Application Messaging Protocol) [50], which includes a publish/subscribe process.

The WebSocket protocol uses the origin model used by web browsers to restrict which web pages can contact a WebSocket server (when a web page uses the protocol). If a dedicated client uses the WebSocket protocol directly, the client can provide any arbitrary origin and the origin model is not useful. The WebSocket protocol protects against malicious JavaScript running inside a trusted application such as a web browser. While it is intended to be used by scripts in web pages, it can also be used directly by hosts which act on their own behalf and can therefore send fake header fields, misleading the server. In such cases servers should not assume that they are talking directly to scripts from known origins and extra validation features from server side must be considered. Overall, it is suggested to implement WebSocket with the support of TLS.

3. Comparison of IoTApplication Protocols

Several works have been published with comparative performance results of the IoT application protocols during the last years. These works usually compare two, three or more protocols via simulations, testbed implementations, close to real world conditions or a combination of the abovementioned methods in environments and applications with different requirements. In most cases, networking protocols and hardware/software implementations are significantly different. This Section records the most important studies that have been carried out over the past three years. The results are

categorized following the work of [3], based on key performance indicators such as latency, required bandwidth and throughput, energy consumption as well as the developers' choice. As a first step, Table 1 summarizes the key features of the IoT application protocols under consideration based on the previous Section's description of the protocols and the corresponding literature.

3.1. Latency

The latency of data transfer from source to destination (e.g., server to client) is one of the most important parameters for assessing the performance of a network and its protocols within the IoT framework and in specific cases can be critical depending on the application and the type of data transferred. The recent literature is rich and includes many relative comparative studies (see also [3]). Iglesias-Urkia et al. in [51], examined the latency of MQTT and CoAP protocols in a simple, uncongested scenario over LAN. In all cases and regardless of the QoS level, CoAP shows lower latency (no great difference between reliable and unreliable CoAP message transport). The latency, as expected, increases in MQTT protocol when changing the QoS option (from QoS 0 to QoS 1 and QoS 2). A comparative study of the two protocols is also given in [52] using an emulator. The authors considered cases of low and high traffic (in terms of number of nodes and messages originating from each node) with the Packet Loss Rate (PLR) of the network being the parameter under consideration. The latency of CoAP was found to be lower in all cases, with the difference being greater in low traffic and low PLR. Similar results were obtained when using a simulator for BLE (Bluetooth Low Energy) and IEEE 802.11ah protocols [53]. CoAP is always better than MQTT, regardless the packet size. The differences are clearly greater in the case of IEEE 802.11ah.

Table 1. Key features of the IoT application protocols under consideration

	MQTT	CoAP	XMPP	AMQP	DDS	REST HTTP	WebSocket
Transport protocol	TCP	UDP	TCP	TCP	TCP/UDP	TCP	TCP
Pub/Sub	X	X	X	X			
Req/Res		X	X	X	X	X	
QoS	QoS 0 (at most once) QoS 1 (at least once) QoS 2 (exactly once)	NON CON	No	At most once At least once Exactly once	Nearly 23 levels of QoS	No	No
Header size (at least)	2	4		8			6-14
Security	TLS/SSL optional	DTLS optional	TLS/SSL embedded	TLS/SSL embedded	TLS/DTLS/ DDS sec., optional	SSL embedded in HTTPS	TLS/SSL optional
Encoding format	Binary	Binary	XML	Binary	Binary	Text	Binary
RESTful support	No	Yes	No	No	No	Yes	Yes
Standard	OASIS	IETF	IETF	OASIS	OMG	IETF	IETF

An interesting comparison that includes CoAP, MQTT and WebSocket was presented in [54]. The authors measured the RTT (Round Trip Time), that is, the total time from the origination of a packet from an IoT device to the server (broker) until the reception of the server's response to the device. They considered cases where the IoT device and the server are connected to the same LAN, through an ISP and

via a mobile network. The results indicate that in cases of LAN and ISP interconnections, the delay is comparable between the three protocols when MQTT with QoS 0 is used. MQTT with QoS 1 presents much higher delay which increases with the size of the packet. In the case of the mobile network, the MQTT with QoS 0 presents the best performance, while the CoAP, WebSocket and MQTT with QoS 1 protocols follow in descending order. CoAP, MQTT, and XMPP protocols are compared over a LAN in [55]. MQTT QoS 0 appears slightly better than CoAP, while XMPP shows latencies of two orders of magnitude higher.

The delays of MQTT (QoS 1 and 2) and AMQP (QoS 1) from the publisher to the subscriber (linked via the Azure cloud platform in a factory environment) are compared in [56]. For small packet lengths the results are comparable, with MQTT QoS 1 showing almost half the latency of AMQP. For large packet lengths, however, MQTT is significantly superior to AMQP, regardless the QoS type examined. In [57], MQTT, AMQP and XMPP protocols were compared by emulating data transfers from simple sensors, sophisticated sensors as well as multimedia sensors that send data via virtual machines to the cloud; various motion scenarios that include multiple publishers and subscribers were examined. In all cases, MQTT and AMPQ perform clearly better than XMPP (with MQTT being slightly better), except for the case of multimedia sensors where the results are comparable. The same protocols are also compared in [58], where the authors used a testbed over a LAN network. The latencies of MQTT and AMQP were comparable and were found to increase with the packet size and PLR. XMPP had always a higher latency that did not depend on PLR.

A comparison of CoAP and HTTP protocols is provided in [59] and [60]. In both cases CoAP showed better results. In [61], the MQTT, CoAP and DDS protocols were compared in a medical application scenario with a network emulator. DDS delivered better results than MQTT (both using TCP), while CoAP (with UDP) performed better than the others. In addition, MQTT, CoAP, XMPP, and DDS were examined in [62] with respect to the transfer delay of sensor messages. MQTT provided the best results, followed by AMQP, XMPP and DDS (for DDS the results are not safe). Johnsen in [63] compared MQTT, MQTT-SN, AMQP and XMPP protocols using Raspberry Pi over a LAN by sending 100 messages between the publisher and the subscriber. The total message transfer time was measured to be in the following order (from lowest to highest): MQTT, MQTT-SN, XMPP and AMQP. Finally, the work of [20] compared the MQTT, CoAP, HTTP, and AMQP protocols based on papers published up to 2015 (not included in the references herein). The performance of protocols as presented by the lowest to the highest latency was CoAP, MQTT, AMQP, and HTTP.

Summarizing the above review, it is evident that in almost all cases (although dissimilar with respect to the configuration of the IoT system), the CoAP protocol (mainly because it is based on UDP) shows the lowest latency, with MQTT coming next (with a few cases resulting better when MQTT QoS 0 was used). It should be noted, however, that UDP (embedded in CoAP) gives lower reliability than TCP (used in MQTT). The AMQP protocol on the other hand is comparable to or worse than MQTT, while XMPP typically results in much higher latency. For DDS the results were not enough and reliable safe while HTTP was in all cases checked in the last place (no comparison with XMPP was recorded).

3.2. Required Bandwidth and Throughput

The bandwidth required for communication is an important parameter of any telecommunication system, especially in case of IoT applications in which IoT devices can be too many in number (depending on the application) while the available spectrum resources for the application's development are usually limited. It is natural for the required bandwidth to be larger for those protocols that present a higher overhead (in terms of the number of total bytes sent in order to transmit a particular message) and/or require a greater number of packet exchanges due to their structure and the transport layer they use. Throughput on the other hand, in terms of data (e.g., bytes, packets, messages) that are successfully transferred per unit of time, is related to the efficient use of the available spectrum and the system's reliability.

In this context, Pohl et al. [58], examine the number of bytes required to send a specific number of messages in a testbed over a LAN, using MQTT, AMQP, and XMPP protocols. As expected from the preceding description of the protocols, MQTT was found to need fewer bytes, with AMQP following and XMPP having the highest requirements (about twice the number of bytes in relation to MQTT). In [60], HTTP and CoAP were compared in terms of the required bandwidth. HTTP always required multiple times the bandwidth of CoAP, regardless of the PLR of the link. In the same context, MQTT, CoAP and DDS protocols were compared in [61], in terms of the bandwidth they consume. CoAP and MQTT were found to consume less than half the bandwidth that DDS requires. CoAP appears slightly better than MQTT in almost all cases except when the PLR or the network latency is high. In [20], where MQTT, CoAP, HTTP and AMQP are compared based on previously published research results, it becomes clear that CoAP has the smallest bandwidth requirements, followed by MQTT, AMQP and HTTP. Note that the difference between CoAP and MQTT increases significantly when QoS 2 is used in MQTT. In [64], authors compared MQTT (all QoS types), CoAP and REST HTTP in terms of bandwidth for an IoT device to cloud communication. The results indicated that bandwidth consumption depends on the size of the payload. CoAP consumed the lowest bandwidth for small payloads, followed by MQTT and REST HTTP, while for large payloads the REST HTTP outperformed the rest of the protocols.

As far as it concerns throughput, MQTT and CoAP are compared in [52] with MQTT clearly showing a better behavior in all cases of telecommunication traffic and PLR. Correspondingly, message loss rate measurements in [61], showed that MQTT and DDS clearly show very good behavior without noticeable loss of messages, as opposed to CoAP, which, due to UDP, results in increased message losses when network's PLR rises. Comparison of MQTT, AMQP and XMPP in [57], showed that MQTT greatly outperforms AMQP in cases of a large number of small messages or a large number of medium messages, while AMQP shows slightly better performance in case of large multimedia messages. XMPP was worse in all cases. Similar outcomes are presented in the work [58]. Finally, when reliability was examined in the review work of Naik [20], the resulted ranking from best to worse performance was MQTT, AMQP, CoAP and HTTP.

Overall, in the overwhelming percentage of cases (though different among them), CoAP, due to the use of UDP and the lowest overhead, presents lower bandwidth requirements, in most cases, than MQTT, which in turn clearly outperforms AMQP and XMPP. However, the use of UDP in CoAP leads to worse throughput than MQTT, which also outperforms AMQP (with the exception of large multimedia messages transfer). XMPP showed the worst performance.

3.3. Energy/Power Consumption

Energy consumption is an extremely important parameter in the context of IoT, as the majority of IoT devices do not use a central power supply but are mainly based on batteries. Thus energy and power efficiency when using IoT application protocols, among other factors, are important. In [53], the sensor battery life is examined in an IoT network that uses either BLE or IEEE 802.11ah. The comparative results for MQTT and CoAP showed that CoAP was better in all cases, in an excellence rate of 10% to 60%, with the gain decreasing when the frequency of messages decreased and the size of the messages grew. However, it was clear that the networking protocol was of paramount importance in battery life while the contribution of the application protocol was overall, almost insignificant. The estimated battery life was 20 times shorter when using IEEE 802.11ah. Joshi et al. [65] evaluated the MQTT, CoAP and HTTP protocols and demonstrated that MQTT results in slightly lower energy consumption than CoAP, with HTPP being far worse than both. Thota et al. [66] state that in simple IoT scenarios, MQTT was more suitable for IoT messaging with no power constraints, while CoAP proved to be more efficient with respect to power management capabilities. On the other hand, AMQP and MQTT performance was compared in [67] under a mobile or unstable wireless network; MQTT was found to be more energy efficient. The performance ranking of protocols in terms of power consumption was also given in the review work of Naik [20]. The results, in an increasing order (lower to higher consumption) were found to be CoAP, MQTT, AMQP, and HTTP.

3.4. Preferences in Recent IoT Applications' Development

In addition to the comparison of the features and performance of application layer protocols provided in the previous sections, it is worth noting the preferences in the adoption of protocols by IoT application developers. The Eclipse IoT Working Group, the IEEE IoT Initiative, the Open Mobile Alliance (or the IoT Council) and the AGILE-IoT H2020 project have conducted relevant surveys in recent years, i.e., 2017 and 2018 [68-69]. Their results showed that while in 2017 the majority of developers preferred HTPP (60.1%) and MQTT (54.7%) followed in a significant distance by CoAP (26.7%) proprietary or in-house solutions, HTTP 2.0, AMQP and XMPP, the situation has changed during 2018. In specific, MQTT was of high preference (over 60%), followed by HTPP (lower that 55%) WebSockets (around 35%) and HTTP 2.0, CoAP, and AMQP following. XMPP was significantly lower in 2018, while preference of DDS was low in both years.

4. On the use of IoT Application Protocols in Smart Farming

IoT applications in agriculture have been in the forefront for several years. This Section reviews the most recent attempts (over the last two years) and the application protocols used or proposed. In this context, a categorization is made to discriminate between simple, small-scale applications and complete prototypes that have been implemented in medium or large-scale applications. Then, a discussion on the evaluation of IoT application layer protocols in smart farming implementations is presented, based on the key parameters that were described in the previous sections.

4.1. Review of IoT Implementations in Smart Farming

4.1.1. Simple, Small Scale Applications

Simple experimental applications using sensors and actuators are mainly classified in this category. An application of soil moisture measurement using MQTT is presented in work [70], while in [71] a relay is used to control a water pump, also using the MQTT protocol. Slightly more complex systems using more sensors (e.g., measuring luminance, temperature, atmospheric pressure, humidity) and actuators for watering control are presented in [72-75], also using the MQTT protocol in all cases. A simple system for collecting measurements using a gateway (an edge computing device) is presented in [76] where, sensor communication with the gateway uses the MQTT protocol while transfer of measurements from the gateway to the cloud takes place with REST HTTP. A measurement collection system using LoRaWAN and TheThingsNetwork (TTN) platform [77] uses the REST API to transfer data from the TTN platform to the data management application and WebSocket to transfer them to the web. Finally, a simple measurement collection and actuators management application is described in [78], based on REST HTTP and WebSocket.

4.1.2. Full Prototypes for Medium and Large-Scale Applications

This category includes integrated prototypes developed for medium-scale applications up to integrated large-scale intelligent farming systems. Gomez et al. [79] presented a small crop monitoring prototype using MQTT. An original (also small-scale) hydroponic system using MQTT to transfer temperature, humidity, PH, and electrical conductivity measurements as well as to control actuators was presented in [80]. The MQTT protocol was also the basis of the prototypes developed in [81-83], the latest of which is a plant-monitoring system using a variety of sensors, including a camera. An application of temperature and humidity measurement of agricultural products during desiccation that includes the MQTT protocol as well was presented in [84]. A smart watering system using multiple sensors and machine learning methods in a mid-scale application was presented in [85]. The REST protocol was used to collect metrics and HTTP to manage actuators.

Moving to somewhat more complex applications and prototypes, Trilles et al. [86] presented an integrated MQTT-based application, using multiple sensors to monitor vines. An original system for precision farming with many sensors and actuators was described in [87], where the MQTT and CoAP protocols were the basic choices.

An integrated prototype system for precision farming which manages a variety of sensors and actuators was also presented in [88-89]; it was proposed to use MQTT and REST HTTP in parallel. A smaller scale application using MQTT and REST HTTP/NGSI was presented in [90]. Full prototypes for monitoring farm facilities and other environmental parameters using REST HTTP were described in [91-92]. In [93] a case study of a smart irrigation system developed using the MQTT protocol was described with the measured data (temperature and soil moisture) collected and managed by AWS IoT cloud. Finally, a large scale integrated application with a large number of sensors and actuators was described in [93], where the CoAP protocol is used to collect sensor readings into a gateway which then communicates with the management platform using REST HTTP.

In summary, and similarly to the previous cases, it is evident that the MQTT protocol is the most used and preferred for simple and complex IoT implementations. Following MQTT, the next most preferable choice is the REST HTTP, while the CoAP protocol is used in a few cases. It is worth noting that in some larger scale implementations, MQTT and CoAP are combined with REST HTTP.

4.2. Evaluation of IoT Application Layer Protocols in Smart Farming Implementations

Summarizing and elaborating the preceding description and comparison of the IoT application layer protocols, this Section presents the most important aspects identified and discusses issues and challenges that have to be considered in the context of agricultural applications.

The CoAP protocol (mainly due to the fact that is based on UDP) has been shown to achieve the lowest latency while simultaneously raises bandwidth and energy requirements, in almost all research works under consideration. MQTT comes next (being superior to CoAP in a few cases where QoS 0 was used) followed by AMQP, XMPP and REST HTTP with significantly lower performance in these aspects. However, when it comes to reliability, that is, the capability to deliver the information effectively, the situation is different. Effective throughput and reliability are of major importance for smart farming since unreliable data can lead to inaccurate decisions and inappropriate automated procedures which may result in significant or even intolerable cost. This is further complicated by the fact that a huge variety of data, ranging from low periodicity simple measurements to real-time multimedia, that may, in addition, present huge variability in velocity, need to be properly supported in precision farming. Further on, the diverse, and in several cases, unpredictable (beforehand) propagation conditions which may range from almost free space to severe attenuation and fading (e.g., passing through a heavy crop canopy), is clearly a critical factor that affects data veracity and accuracy. Even if the underlying physical/MAC/network layer communication technology is high (leading to increased cost), it might not be adequate. In this context, MQTT (which uses TCP, in contrast to CoAP UDP) seems to be at the moment the best option, outperforming the rest of the protocols considered as well, in almost all cases surveyed.

On the other hand, when it comes to real-life implementations, the challenge is still there. Scalability, in terms of adding newer devices over the existing infrastructure without affecting the functionality, the performance and QoS of an established framework, is a key aspect in agricultural applications. Many sensor and router nodes need to be diffused in an IoT agricultural system for measurements collection and reliable information transfer. Additionally, the implementation of a network in an IoT agriculture system should not be physically restricted; in most cases, farming applications refer to wide geographical areas, meaning that the architecture should be widely deployed and should be scalar. In general, a large-scale deployment or the addition/duplication of several smaller networks could minimize the operational hazards over a huge agricultural area. In the context of IoT application protocols, the above mentioned scalability considerations affect bandwidth utilization, throughput, reliability and QoS. Therefore, the IoT application protocols in contemporary or future agricultural IoT systems should be tested in real world

scenarios that include a very large number of interconnected hardware or software equipment, for sufficient time. While some recent research works attempted to emulate such scenarios, the performance of a fully developed IoT network with a huge number of nodes under different environmental conditions that create a big volume of diverse data has not been yet tested and evaluated. This remains a challenging task to achieve.

Another key aspect when choosing the application layer protocol when developing an IoT-driven smart farming system is what the favorite, best choice of the manufactures'–developers is. The MQTT and CoAP protocols are very simple and "light" in computational requirements and can be applied relatively easily to nodes with strong constraints on computational power and power autonomy. On the other hand, our survey has shown that the MQTT and HTTP protocols are at the forefront of the researchers and manufacturers choice, both in experimental and commercial level, for all applications, including agricultural applications. This is mainly due to the maturity of the protocols and their support by large companies and communities. Furthermore, the developers' choice is directly associated with the cost of implementing or integrating application protocols for better bandwidth or throughput rates between IoT devices. The total hardware and software costs in smart farming products must be kept low so as to be globally available to rich and poor country markets and it is a significant challenge to further reduce software and hardware costs. Still, the developers' choice can be influenced by the availability of an IoT application protocol, meaning the easiness of development or the existence of suitable hardware and software that can be used to deliver the desired IoT services. MQTT and HTTP application protocols have many different versions available and they may be easily integrated in off-the-shelf hardware and middleware. Software modules for protocols like CoAP on the other hand, do not yet have adequate plurality and may need more sophisticated programming and hardware settings in order to work. Hence the easy association of application protocol software modules with IoT devices or more preferably the integration and multi-functionality of many, different protocol versions in IoT hardware can contribute towards a much more effective performance of IoT in smart farming. Finally, since each protocol's performance is directly associated with the underlying software design and its integration to the IoT hardware equipment, it is extremely important to keep their services updated and widely accepted by as many as possible IoT devices so as to meet the requirements of the numerous varying agricultural applications. An additional factor, in the same framework, is the interoperability issue. In farming sector there is a huge number of heterogeneous devices used to collect, transfer and process data and managing the data traffic among these devices is a difficult and challenging task. The challenge here is not the lack of standards (e.g. for semantics and data modeling, agri-machinery, weather data, supply chain, e-commerce retail stores etc.) or technologies or protocols, but their great number and the proper selection among them.

Data security is another very challenging issue. By using various communication devices and protocols, data are vulnerable to security or intrusion threats or breaches hence maximum emphasis should be given when designing or applying IoT protocols. In their review about security implications in smart agriculture Jahn et al. [95] state that "the structure and operation of modern highly networked food systems fundamentally depends on networked information systems, some of which may not be secured from cyber-attacks and are highly vulnerable to hybrid warfare tactics". In terms of security, MQTT and WebSocket may use the TLS protocol for securing connectivity, while the CoAP protocol may use the DTLS protocol at the transport level. None of them, however, has the corresponding security protocol embedded. XMPP has a built in TLS mechanism to provide reliability in terms of confidentiality and data integrity and uses, in addition, a special SASL profile to identify entities. AMQP integrates communications security and identification with the use of SASL and TLS, and DDS provides the TLS or DTLS security options, depending on whether TCP or UDP is used, however, the issue of security is still open for DDS. The secure extended version of MQTT, called Secure MQTT (SMQTT) has recently been proposed to deal with security issues and it seems to be a promising choice for smart farming applications. Although all the IoT application layer protocols include security solutions (either embedded or not), these must be further enhanced so as maximum security should be ensured in all cases while their overall performance (in terms of latency, bandwidth and throughput efficiency) has to be evaluated.

5. Discussion

On the basis of the above, the safest, at the moment, option seems to be the MQTT protocol, either when it is applied in an end-to-end network architecture, or when a gateway-server architecture is used to collect the measurements. This outcome is reinforced by the comparison of IoT application protocols analysed in Section 3, where the most important research works with comparative performance results during the past three years have been studied. In this context, Table 2 presents a ranking of the IoT protocols from most promising to least promising based on the preceding survey and analysis of the key performance indicators, i.e., latency, required bandwidth, throughput, reliability, energy consumption as well as the developers' choice that reveals the maturity and ease of adoption of the protocols for off-the-shelf applications. Researchers' preferences for agriculture applications and prototypes development are mentioned as well. From the table, as well as from the abovementioned analysis, it can be derived that in most cases MQTT outperforms the other IoT protocols. MQTT achieves the first or (fewer times) the second best ranking for the majority of the examined performance indicators. The use of CoAP protocol can be considered as the next best option since it attains very promising performance in most network performance indicators.

Ideally, the same protocol could be used in all parts of the communication architecture, i.e., from the IoT devices to the gateway, the gateway to the cloud (IoT platform), the cloud to the end-user device as well as the reverse direction when considering actuators' management (see also [3] for IoT device-to-fog-to-cloud architecture considerations). In this way, no protocol-to-protocol translator would be required. However, this framework is not mandatory and different protocols may be used in different parts of the communication network architecture. This is a valid option depending on the additional requirements of the IoT platform to be used/developed, the hardware and software requirements of the gateways (if any), etc. In any case, in agricultural applications that have already been deployed using HTTP, the use of MQTT for IoT device-to-gateway communications would be beneficial.

Table 2. IoT application protocols comparison based on key performance indicators

Key performance indicator	Most promising protocol				Least promising protocol
Latency					
Over a LAN	CoAP	MQTT QoS 0	AMQP	HTTP/REST	XMPP
Over a mobile network	MQTT QoS 0	CoAP	WebSocket	MQTT QoS 1	
Bandwidth consumption	CoAP	MQTT	AMQP and XMPP	DDS	HTTP/REST
Throughput	MQTT	DDS	CoAP	AMQP	XMPP
Reliability	MQTT	AMQP	CoAP	HTTP/REST	
Energy consumption	CoAP	MQTT	AMQP	HTTP/REST	
Developers' preference in recent IoT applications	MQTT	HTTP/REST	WebSocket	HTTP 2.0	CoAP, AMQP, XMPP, DDS
Researchers' preference in IoT agriculture applications	MQTT	HTTP/REST	CoAP		

Overall, while MQTT seems to be the most mature choice for the moment, there are a lot of issues and challenges that need to be considered in order to come to a safe choice, as summarized above. There is not a suitable-for-all solution. Newer protocols as well as evolution of the existent ones (already described in Section 2) have to be further analyzed and tested in real settings. A specific analysis on the

implementation scale (either geographical or in terms of number of devices), the physical environment, the available hardware and software as well as the underlying network technology and the specific requirements of the agricultural application to be developed, has to take place, so to conclude to an efficient and long-lasting choice.

6. Conclusion

This work provided a thorough and up-to-date survey of IoT messaging (application) protocols that are regarded as major options for IoT applications. Based on the most recent literature, seven protocols (MQTT, CoAP, XMPP, AMQP, DDS, REST-HTTP and WebSocket) were presented, analyzed and compared with respect to their performance, measured in terms of relevant key indicators, i.e., latency, energy and bandwidth requirements, throughput, reliability and security. Important issues posed by contemporary and future smart farming applications were discussed, aiming to provide a strong basis for real-life implementation choices and to drive future research efforts that will address the described challenges towards a reform of smart farming systems and applications that will effectively and efficiently cover societal needs.

Acknowledgment

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T1EDK-02296).

References

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communications Surveys & Tutorials*, Vol. 17, No. 4, 4th quarter 2015, pp. 2347-2376.

[2] P. Masek, J. Hosek, K. Zeman, M. Stusek, D. Kovac, P. Cika, J. Masek, S. Andreev, F. Kröpfl, "Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience", *International Journal of Distributed Sensor Networks*, Vol. 2016, Article ID 8160282, 2016.

[3] J. Dizdarevic, F. Caprio, A. Jukan, X. Masip-Bruin, "A Survey of Communication Protocols for Internet-of-Things and Related Challenges of Fog and Cloud Computing Integration", *ACM Computing Surveys*, Apr. 2018.

[4] D. Dragomir, L. Gheorghe, S. Costea and A. Radovici, "A Survey on Secure Communication Protocols for IoT Systems", *International Workshop on Secure Internet of Things*, Heraklion, Greece, 26-30 Sep. 2016.

[5] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things", *Transactions on IoT and Cloud Computing*, 2015.

[6] P. Sethi and S.R. Sarangi, "Internet of Things: Architectures, Protocols, and Applications", *Hindawi Journal of Electrical and Computer Engineering*, Jan. 2017, pp. 1-25.

[7] Ö. Köksal and B. Tekinerdogan, "Architecture design approach for IoT‐based farm management information systems" *Springer Precision Agriculture*, Dec. 2018.

[8] C. Sharma and D.N.K. Gondhi, "Communication Protocol Stack for Constrained IoT Systems", *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, 2018.

[9] A.D. Pathaka and J.V. Tembhurnea, "Internet of Things: A Survey on IoT Protocols", *3rd International Conference on Internet of Things and Connected Technologies (ICIoTCT)*, 2018, pp. 483-487.

[10] A. Colakovic and M. Hadžialic, “Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues”, *Elsevier Computer Networks*, 144, 2018, pp. 17-39.

[11] Y. Fathy, P. Barnaghi, and R. Tafazolli, “Large-Scale Indexing, Discovery, and Ranking for the Internet of Things (IoT)”, . ACM Computing Surveys, Vol. 51, No. 2, pp. 29:1–53.

[12] D.G. Kogias, E.T. Michailidis, G. Tuna and V.C. Gungor, “Realizing theWireless Technology in Internet of Things (IoT)”, *Emerging Wireless Communication and Network Technologies*, Springer, 2018, pp. 173-192.

[13] J. Hoffmann, D.Kuschnerusy, T. Jonesz, M. Hubner, “Towards a Safety and Energy Aware protocol for Wireless Communication”, *13th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*, 2018.

[14] A. Rayes, S. Salam. *Internet of Things From Hype to Reality*. Springer Nature Switzerland AG, 2019.

[15] O.Elijah, T.A. Rahman, I. Orikumhi, C. Y. Leow, M.N. Hindia, "An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges", *IEEE Internet of Things Journal*, 2018

[16] Abhishek Khanna, Sanmeet Kaur,” Evolution of Internet of Things (IoT) and its significant impact in the field of Precision Agriculture”, *Computers and Electronics in Agriculture*, v.157, pp. 218-231, 2019

[17] P. P. Ray, “Internet of things for smart agriculture: Technologies, practices and future direction”, Journal of Ambient Intelligence and Smart Environments, v. 9, pp. 395–420, 2017

[18] An.Tzounis, N. Katsoulas, T. Bartzanas, C. Kittas, “Internet of Things in agriculture, recent advances and future challenges”, Biosystems engineering 2017, v.164 pp. 31-48, 2017

[19] M. J. Haider, Rosdiadee Nordin, Sadik Kamel Gharghan , M.J. Aqeel and Mahamod Ismail, “Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review”, Sensors (Basel), 17(8): 1781, doi: 10.3390/s17081781, 2017.

[20] N. Naik, “Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP”, *IEEE International Systems Engineering Symposium (ISSE)*, 2017.

[21] D. Locke, “MQ Telemetry Transport (MQTT) v 3.1 protocol specification”, *IBM Developer Works Technical Library*, http://goo.gl/3tLZVj.

[22] A. Banks and R. Gupta (Eds). *MQTT Version 3.1.1*. OASIS Standard, Oct. 2014. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[23] https://www.hivemq.com/mqtt-essentials/

[24] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246, 2008. http://www.rfc-editor.org/rfc/rfc5246.txt

[25] M. Singh, M. A. Rajan, V. L. Shivraj, and P. Balamuralidhar, “Secure MQTT for Internet of Things (IoT),” *5th International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 746–751, 2015.

[26] A. Stanford-Clark and H.L. Truong, *MQTT For Sensor Networks (MQTT-SN): Protocol Specification Version 1.2*, 2013, http://goo.gl/eDqIRQ.

[27] A. Banks, E. Briggs, K. Borgendale, and R. Gupta (Eds), *MQTT Version 5.0*. 31 October 2018. http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[28] Z. Shelby, K. Hartke, and C. Bormann. 2014. *The Constrained Application Protocol (CoAP)*. RFC 7252. http://www.rfc-editor.org/rfc/rfc7252.txt

[29] C. Bormann, A. P. Castellani, and Z. Shelby, “CoAP: An application protocol for billions of tiny Internet nodes,” *IEEE Internet Computing*, Vol. 16, no. 2, pp. 62–67, Mar./Apr. 2012.

[30] R.T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, PhD dissertation, University of California Irvine, 2000.

[31] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor. *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*. RFC 8323, 2018.

[32] N. Correia, D. Sacramento, and G. SchÃijtz. 2016, “Dynamic Aggregation and Scheduling in CoAP/Observe-Based Wireless Sensor Networks”, *IEEE Internet of Things Journal*, Vol. 3, no. 6, Dec 2016, pp. 923–936.

[33] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security*. RFC 4347, Oct. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc4347.txt

[34] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. RFC 6120, 2011.

[35] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. RFC 6121, 2011.

[36] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Address Format*. RFC 6122, 2011.

[37] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Address Format*. RFC 7622, 2011.

[38] P. Millard, P. Saint-Andre, R. Meijer. *XEP-0060: Publish-Subscribe, Draft Standard, ver. 1.15.6*, last updated 22-11-2018. https://xmpp.org/extensions/xep-0060.html.

[39] M. T. Jones, *Meet the Extensible Messaging and Presence Protocol (XMPP)*. DeveloperWorks, 2009.

[40] H. Wang, D. Xiong, P. Wang, and Y. Liu, "A Lightweight XMPP Publish/Subscribe Scheme for Resource-Constrained IoT Devices", *IEEE Access*, Vol. 5, 2017, pp. 16393–16405.

[41] *AMQP Advanced Message Queueing Protocol, Ver. 0-9-1*, November 2008. A General Purpose Messaging Standard. http://www.amqp.org/specification/0-9-1/amqp-org-download.

[42] OASIS. 29 October 2012. *Advanced Message Queuing Protocol (AMQP) Version 1.0*. OASIS Standard. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html

[43] *Data distribution services specification, V1.2*, Object Manage. Group (OMG), Needham, MA, USA, Apr. 2, 2015. [Online]. Available: http://www.omg.org/spec/DDS/1.2/

[44] F. Inglés-Romero, A. Romero-Garcés, C. Vicente-Chicote, and J. Martínez. "A Model-Driven Approach to Enable Adaptive QoS in DDS-Based Middleware", *IEEE Transactions on Emerging Topics in Computational Intelligence*, June 2017, pp. 176–187.

[45] Inc Twin Oaks Computing. *What can DDS do for You?* Retrieved January 3, 2018 from https://www.omg.org/hot-topics/documents/dds/CoreDX_DDS_Why_Use_DDS.pdf/

[46] Object Management Group (OMG). *OpenDDS Developer's Guide*, 2017.

[47] Z.B. Babovic, J. Protic and V. Milutinovic, "Web Performance Evaluation for Internet of Things Applications", *IEEE Access*, Vol. 4, 2016, pp. 6974-6992.

[48] M. Belshe, R. Peon, and M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540, 2015. http://www.rfc-editor.org/rfc/rfc7540.txt http://www.rfc-editor.org/rfc/rfc7540.txt.

[49] I. Fette and A. Melnikov. *The WebSocket Protocol*. RFC 6455, 2011. http://www.rfc-editor.org/rfc/rfc6455.txt

[50] https://wamp-proto.org/

[51] M. Iglesias-Urkia, A. Orive, M. Barcelo, A. Moran, J. Bilbao, and A. Urbieta, "Towards a lightweight protocol for Industry 4.0: An implementation based benchmark", In IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017.

[52] F. Ouakasse and S. Rakrak, "A Comparative Study of MQTT and CoAP Application Layer Protocols via Performance Evaluation", Journal of Engineering and Applied Sciences 13 (15), 2018, pp. 6053-6061.

[53] A. Larmo, F. Del Carpio, P. Arvidson, R. Chirikov, "Comparison of CoAP and MQTT Performance Over Capillary Radios", *Global Internet of Things Summit (GIoTS)*, Bilbao, Spain, June 4-7, 2018.

[54] S. Mijovic, E. Shehu, and C. Buratti, "Comparing application layer protocols for the Internet of Things via experimentation", IEEE *2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, 2016.

[55] B.H. Çorak, F.Y. Okay, M. Güzel, Ş. Murt, S. Ozdemir, "Comparative Analysis of IoT Communication Protocols", *International Symposium on Networks, Computers and Communications (ISNCC)*, Rome, Italy, June 19-21, 2018.

[56] E. C. M. van der Linden. A latency comparison of IoT protocols in MES. MSc Thesis, Linköping University, Sweden, 2017.

[57] D. Happ, N. Karowski, T. Menzel, V.Handziski, A.Wolisz, "Meeting IoT platform requirements with open pub/sub solutions", *Spinger Ann. Telecommun*, vol. 72, 2018, pp. 41–52.

[58] M. Pohl, J. Kubela, S. Bosse, K. Turowski, "Performance Evaluation of Application Layer Protocols for the Internet-of-Things", *6th International Conference on Enterprise Systems (ES)*, Limassol, Cyprus, Oct. 1-2, 2018, pp. 180-187.

[59] M. Saleh, M. A. Abdou, and M. Aboulhassan, "Assessing the use of IP network management protocols in smart grids" *IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, 2016

[60] W. Gao, J. H. Nguyen, W. Yu, C. Lu, D. T. Ku, and W. G. Hatcher, "Towards Emulation-Based Performance Assessment of Constrained Application Protocol in Dynamic Networks", *IEEE Internet of Things Journal*, Vol. 4, No. 5, Oct. 2017, pp. 1597-1610.

[61] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network", *International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*, 2016.

[62] Z.B. Babovic, J. Protic, and V. Milutinovic, "Web Performance Evaluation for Internet of Things Applications", *IEEE Access*, Vol. 4, 2016, pp. 6974–6992.

[63] F.T. Johnsen, "Using Publish/Subscribe for Short-lived IoT Data", *Federated Conference on Computer Science and Information Systems (FedCSIS)*, Poznan, Poland, Sep. 9-12, 2018, pp. 645-649.

[64] Tandale, B. Momin, and D. P. Seetharam, "An empirical study of application layer protocols for IoT", *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 2447–2451, Aug 2017.

[65] J. Joshi, V. Rajapriya, S.R. Rahul, P. Kumar, S. Polepally, R. Samineni, and D.G.K. Tej, "Performance enhancement and IoT based monitoring for smart home", *International Conference on Information Networking (ICOIN)*, 2017, pp. 468–473.

[66] P. Thota and Y. Kim, "Implementation and Comparison of M2M Protocols for Internet of Things", *4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science Engineering (ACIT-CSII-BCD)*, 2016, pp. 43–48.

[67] J.E. Luzuriaga, M. Perez, P. Boronat, J.C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks", *12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 2015, pp. 931–936.

[68] Eclipse IoT Working Group, το IEEE IoT Initiative, IoT Council and AGILE-IoT H2020 project. *IoT Developer Survey Results*. Apr. 2017.

[69] Eclipse IoT Working Group, το IEEE IoT Initiative, Open Mobile Alliance, AGILE-IoT H2020 project. *IoT Developer Survey Results*. Apr. 2018.

[70] R.K. Kodali and A. Sahu, "An IoT Based Soil Moisture Monitoring on Losant Platform", *2nd International Conference on Contemporary Computing and Informatics (IC3I)*, Noida, India, Dec. 14-17, 2016, pp. 764-768.

[71] R.K. Kodali and B. S. Sarjerao, "A low cost smart irrigation system using MQTT protocol", *IEEE Region 10 Symposium (TENSYMP)*, July 14-16, 2016.

[72] S. Pooja ; D. V. Uday ; U. B. Nagesh ; S.G. Talekar, "Application of MQTT protocol for real time weather monitoring and precision farming", *International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, Mysuru, India, Dec. 15-16, 2017, pp. 814-819.

[73] T. Cao-hoang and C.N. Duy, "Environment Monitoring System for Agricultural Application Based on Wireless Sensor Network", *7th International Conference on Information Science and Technology*, Da Nang, Vietnam, Apr. 16-19, 2017, pp. 99-102.

[74] M.M. Raikar, P. Desai, N. Kanthi, S. Bawoor, "Blend of Cloud and Internet of Things (IoT) in agriculture sector using lightweight protocol", *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Bangalore, India, Sep. 19-22, 2018, pp. 185-190.

[75] Y. Syafarinda, F. Akhadin, Z.E. Fitri, Yogiswara, B. Widiawan, E. Rosdiana, "The Precision Agriculture Based on Wireless Sensor Network with MQTT Protocol", *1st International Conference on Food and Agriculture*, 2018.

[76] M. R. Suma and P. Madhumathy, "Acquisition and Mining of Agricultural Data Using Ubiquitous Sensors with Internet of Things", *International Conference on Computer Networks and Communication Technologies*, 2018.

[77] D. Davcev, K.Mitreski, S. Trajkovic, V. Nikolovski, N. Koteli, "IoT agriculture system based on LoRaWAN", *14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Imperia, Italy, June 13-15, 2018.

[78] R. Marcelino, L.C. Casagrande, R. Cunha, Y. Crotti, V. Gruber, "Internet of Things Applied to Precision Agriculture", *Lecture Notes in Networks and Systems*, 499–509, 2017.

[79] J. Gomez, A. Fernandez, M.Z. Sánchez, "Monitoring of Small Crops for the Measurement of Environmental Factors Through the Internet of Things (IoT)", *International Conference on Technology Trends (CITT)*, 2018, pp. 16-28.

[80] S. Ruengittinun, S. Phongsamsuan and P. Sureeratanakorn, "Applied Internet of Things for Smart Hydroponic Farming Ecosystem (HFE)", *10th International Conference on Ubi-media Computing and Workshops* (Ubi-Media), Pattaya, Thailand, Aug. 1-4, 2017.

[81] J. Bauer and N. Aschenbruck, "Measuring and Adapting MQTT in Cellular Networks for Collaborative Smart Farming", *IEEE 42nd Conference on Local Computer Networks (LCN)*, Singapore, Oct. 9-12, 2017, pp. 294-302.

[82] J. Bauer and N. Aschenbruck, "Design and Implementation of an Agricultural Monitoring System for Smart Farming", *IoT Vertical and Topical Summit on Agriculture - Tuscany (IoT Tuscany)*, Tuscany, Italy, May 8-9, 2018.

[83] H.A.M. Tran, H.Q.T. Ngo, T.P. Nguyen, H. Nguyen, "Design of Green Agriculture System Using Internet of Things and Image Processing Techniques", *4th International Conference on Green Technology and Sustainable Development (GTSD)*, Ho Chi Minh City, Vietnam, Nov. 23-24, 2018, pp. 28-32.

[84] K. Grgić, I. Špeh and Ivan Heđi, "A Web-Based IoT Solution for Monitoring Data Using MQTT Protocol", *International Conference on Smart Systems and Technologies (SST)*, Osijek, Croatia, Oct. 12-14, 2016, pp. 249-253.

[85] A. Goap, D. Sharma, A.K. Shuklab C.R.Krishna, "An IoT based smart irrigation management system using Machine learning and open source technologies", *Computers and Electronics in Agriculture*, Vol. 155, 2018, pp. 41-49.

[86] S. Trilles, A. González-Pérez and J. Huerta, "A Comprehensive IoT Node Proposal Using Open Hardware. A Smart Farming Use Case to Monitor Vineyards", *Electronics*, 7, 2018, 419.

[87] N. Ahmed, D. De, Md. I. Hussain, "Internet of Things (IoT) for Smart Precision Agriculture and Farming in Rural Areas", *IEEE Internet of Things Journal*, Nov. 2018.

[88] F.J. Ferrández-Pastor, J.M. García-Chamizo, M. Nieto-Hidalgo, J. Mora-Pascual and J. Mora-Martínez, "Developing Ubiquitous Sensor Network Platform Using Internet of Things: Application in Precision Agriculture", *Sensors*, 16, 1141, 2016.

[89] F.J. Ferrández-Pastor, J.M. García-Chamizo, M. Nieto-Hidalgo, and J. Mora-Martínez, "Precision Agriculture Design Method Using a Distributed Computing Architecture on Internet of Things Context", *Sensors* 2018, 18, 1731.

[90] M.A. Zamora-Izquierdo, J. Santa, J.A. Martinez, V. Martinez, A.F. Skarmeta, "Smart Farming IoT Platform Based on Edge and Cloud Computing", *Elsevier Biosystems Engineering*, Special Issue Intelligent Systems for Environmental Applications, 177, 2019, pp. 4-17.

[91] M. Bajceta, P. Sekulic, B. Krstajic, S. Djukanovic, T. Popovic, "A private IoT cloud platform for precision agriculture and ecological monitoring", *International Conference on Electrical, Electronic and Computing Engineering*, 2016.

[92] T. Popovic, N. Latinovic, A. Pešicc, Zarko Zecevic, B. Krstajic, S. Djukanovic, "Architecting an IoT-enabled platform for precision agriculture and ecological monitoring: A case study", *Elsevier Computers and Electronics in Agriculture*, 140, 2017, pp. 255–265.

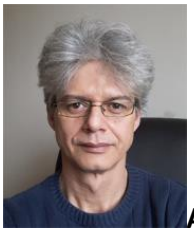
[93] M.M. Raikar, P. Desai, N. Kanthi, and S. Bawoor, "Blend of Cloud and Internet of Things (IoT) in agriculture sector using lightweight protocol", *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2018.

[94] M. Lee, H. Kim, H. Yoe, "ICBM-Based Smart Farm Environment Management System", *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2018.

[95] M.M. Jahn, W.L. Oemichen, G.F. Treverton et al. *Cyber Risk and Security Implications in Smart Agriculture and Food Systems*. White Paper, Jahn Research Group, University of Wisconsin–Madison, College of Agriculture and Life Sciences, 2019.

 **Dimitrios Glaroudis**

Dr. Dimitrios Glaroudis holds an MSc in Multimedia Signal Processing and a PhD in mobile learning. He is a Teaching Fellow in Information and Electronic Engineering Department of International Hellenic University of Thessaloniki since 2004. From 2013 to 2015 he worked in the Hellenic School Network as an IT engineer and since 2006 he has participated in several European and national projects related to ICT technologies. From April 2018 he serves as a research associate in Diophantus Computer Technology Institute and Press (Patras). His main research interests focus on the area of mobile computing and IoT in education.

 **Athanasios Iossifides (Iosifidis)**

Dr. Athanasios Iossifides serves as an Associate Professor in the International Hellenic University and the Director of Advanced Electronic Systems Lab. He received the diploma in Electrical Engineering (1994) and the PhD Degree (2000) from the Department of Electrical & Computer Engineering of Aristotle University of Thessaloniki. From 1999 to 2010 he was with COSMOTE SA. first as a telecom engineer and as the manager of Network Management Section of North Greece. He is a member of the Technical Chamber of Greece and IEEE since 1994. He has served as an associate editor in Wiley Transactions on Emerging Telecommunications Technologies from 2013 to 2018 and IEEE Communications Letters from 2015 to 2018. He has also served as symposium/TPC co-chair in 4 international conferences and as a TPC member in several international conferences. His main research interests lie in the areas of wireless, mobile communications and the Internet of Things. He has co-authored over 40 papers in international journals, conferences and chapters in international books. He received the Best Paper Award in ISCC 2011 and one of the Best Editors Awards of Wiley Transactions on ETT (2017).

Periklis Chatzimisios

Dr. Periklis Chatzimisios serves as an Associate Professor and the Director of the Computing Systems, Security and Networks (CSSN) Research Lab, in the International Hellenic University. He is also a Visiting Fellow in the Faculty of Science & Technology, at Bournemouth University, UK. Dr. Chatzimisios is/has been involved in several standardization and IEEE activities serving such as the IEEE 5G Initiative, the Standards Development Board for IEEE Communication Society (ComSoc), the IEEE ComSoc Standards Program Development Board and the IEEE ComSoc Education Services Board. Dr. Chatzimisios is the editor/author of 8 books and more than 130 papers and book chapters on the topics of performance evaluation and standardization of mobile/wireless communications, Internet of Things, Vehicular Networking and Big Data.

Declaration of interests

× The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: