A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration

JASENKA DIZDAREVIĆ, Technische Universität Braunschweig, Germany
FRANCISCO CARPIO, Technische Universität Braunschweig, Germany
ADMELA JUKAN, Technische Universität Braunschweig, Germany
XAVI MASIP-BRUIN, Universitat Politècnica de Catalunya, Spain

The fast increment in the number of IoT (Internet of Things) devices is accelerating the research on new solutions to make cloud services scalable. In this context, the novel concept of fog computing as well as the combined fog-to-cloud computing paradigm is becoming essential to decentralize the cloud, while bringing the services closer to the end-system. This paper surveys on the application layer communication protocols to fulfill the IoT communication requirements, and their potential for implementation in fog- and cloud-based IoT systems. To this end, the paper first briefly presents potential protocol candidates, including request-reply and publish-subscribe protocols. After that, the paper surveys these protocols based on their main characteristics, as well as the main performance issues, including latency, energy consumption and network throughput. These findings are thereafter used to place the protocols in each segment of the system (IoT, fog, cloud), and thus opens up the discussion on their choice, interoperability and wider system integration. The survey is expected to be useful to system architects and protocol designers when choosing the communication protocols in an integrated IoT-to-fog-to-cloud system architecture.

CCS Concepts: • **General and reference → Surveys and overviews**;

Additional Key Words and Phrases: Internet of Things, fog computing, cloud computing, fog-to-cloud, communication protocols

ACM Reference Format:
Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. 2018. A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration. *ACM Comput. Surv.* 1, 1 (August 2018), 30 pages. https://doi.org/0000001.0000001

Authors' addresses: Jasenka Dizdarević, Technische Universität Braunschweig, Braunschweig, Germany, j.dizdarevic@tu-bs.de; Francisco Carpio, Technische Universität Braunschweig, Braunschweig, Germany, f.carpio@tu-bs.de; Admela Jukan, Technische Universität Braunschweig, Braunschweig, Germany, a.jukan@tu-bs.de; Xavi Masip-Bruin, Universitat Politècnica de Catalunya, Vilanova, Spain, xmasip@ac.upc.edu.

0360-0300/2018/8-ART $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

Continuous innovations in hardware, software and connection solutions in the last decade have lead to the expansion of the Internet of Things (IoT) with the number of connected devices growing by the day [1] [2]. The huge amount of data generated by these devices require to find a proper system architecture able to both process and store all the data. While cloud-based architectures are being currently used for that purpose, the new fog computing paradigm is envisioned to scale and optimize the IoT infrastructures [3]. Examples of the cloud-based IoT solutions have been proposed in [4], [5], [6] and a detailed analysis of properties for IoT cloud providers has been conducted in [7]. These studies have shown that cloud computing has the potential to satisfy many IoT requirements, such as monitoring of services, powerful processing of sensor data streams and visualization tasks. On the other hand, fog-based solutions are suited to address real-time processing, fast data response, and latency issues, thus extending the cloud capabilities closer to the edge of the network [8]. Among many factors that will determine the performance in a combined IoT, fog and cloud computing paradigm, the application layer communication, which in turn depends on the selected communication protocols, is one of the main ones.

Despite the popularity and wide spread usage of HTTP, the currently used protocols in various domains of IoT, fog and cloud domains are de-facto fragmented with many different solutions. This is due to the different requirements and areas that IoT needs to cover, combining the functionalities of sensors, actuators and computing power with security, connectivity and a myriad of other features. As a result, there is no common agreement on the reference architecture or adopted standards of communication protocols. Thus, one of the fundamental challenges for system engineers is to choose the appropriate protocol for their specific IoT system requirements, while levering the advances in fog and cloud computing. For this challenge to be addressed, some general architecture requirements need to be taken into consideration. These requirements include: devices that can range from resource constrained devices to high performance cloud systems, data generated to be processed between the cloud and the fog layer, the types of wireless connectivity that can be used, or security and privacy solutions, just to name a few. While there have been several surveys covering different aspects of the IoT architecture [1, 6, 9–13], the specific issues of communication protocols in the application layer have not been addressed yet.

This paper surveys communication protocols in the application layer (also referred to as messaging protocols and machine-to-machine, depending on the context) in IoT architectures in the context of specific challenges in fog and cloud computing integration, including MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol), XMPP (Extensible Messaging and Presence Protocol), DDS (Data Distribution Service), HTTP (Hypertext Transfer Protocol) and CoAP (Constrained Application Protocol). Recognizing the fact that one single messaging protocol will not be enough to cover the entire communication on the combined IoT-F2C architecture built by bringing together IoT, fog and cloud systems, our goal is to unveil open issues and challenges towards the end goal: their seamless interoperability, coordination and integration. To this end, the paper first presents a comparative analysis of the main characteristics of IoT communication protocols, including request-reply and publish-subscribe protocols. After that, the paper surveys each protocol in detail, and discusses their implementation in various segments of the system (IoT, fog, cloud), and thus opens up the discussion on their interoperability and wider system integration. Finally, we also review the main performance issues, including latency, energy consumption and network throughput. Compared to other related surveys, including [14–22], our focus is on communication protocols in the application layer, with the goal of both exploring their current status, as well as exploring the potential for their integration in the combined IoT, fog and cloud systems.

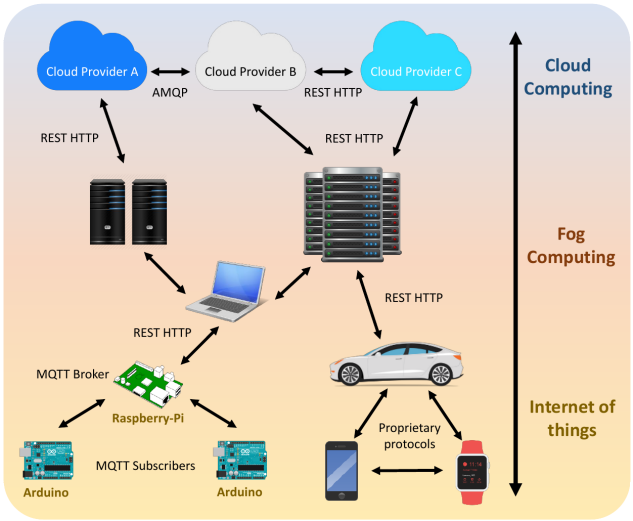


Fig. 1. IoT and Fog to Cloud systems

The rest of this paper is organized as follows. Section 2 presents the background of IoT-F2C architectures and protocols based on publish-subscribe and request-reply interaction model. Section 3 gives a detailed overview of the main features of application layer protocols. Section 4 presents a comparative performance analysis of the protocols surveyed. Section 5 presents possible implementation solutions, such as solutions based on a single communication protocol or a combination of protocols, as well as open issues and challenges. Section 6 concludes the paper.

2 BACKGROUND

This section first provides a background on communication protocols for Internet of Things (IoT) and related scenarios towards fog and cloud computing integration, which motivates the survey. We also provide a brief introduction to communication protocols as basis for more detailed descriptions in the following sections.

2.1 A Fog-to-Cloud Architecture (F2C) for IoT

Recently, notable efforts have been devoted to analyze the advantages and benefits brought by an efficient and coordinated management of IoT, cloud and fog. A few standardization initiatives and industrially led researach consortia highlight their importance, such as the OpenFog Consortium [23], Edge Computing Consortium [24] and the mF2C H2020 EU project [25]. While previous cloud-based solutions only consider two layers, the cloud and the IoT end-devices, these recently proposed combined IoT-fog-to-cloud systems introduce new functional abstractions in between. These abstractions can include a single fog computing layer, whereby the fog computing layer itself can be divided into multiple abstraction sub-layers, depending on various factors, such as resource specifications or set of policies defined to accommodate the different devices into layers. We illustrate one such abstraction with Fig.1 along with the candidate communication protocols. In this typical IoT-fog-cloud ecosystem, the IoT devices are positioned to send data to more capable servers and computing systems in the fog computing layer, such as to perform computing tasks that require low latency. In the same system, cloud computing performs tasks that require larger amounts of computing or storage resources. As we can see at the bottom of Fig.1, some IoT devices can be

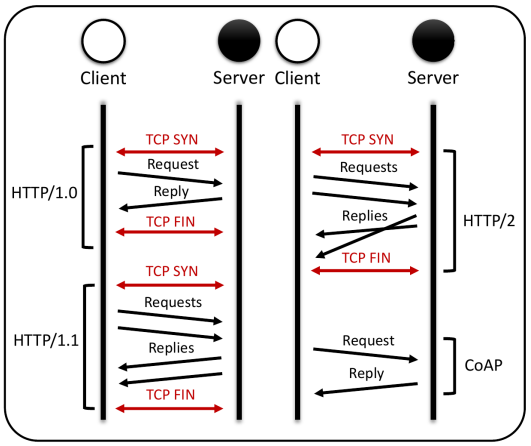


Fig. 2. Request-Reply model, for example: COAP and HTTP

implemented as low-cost processing platforms, such as Arduinos and Raspberry Pis, with MQTT protocol as a communication protocol of choice, as it is optimized to work on constrained devices. It should be noted that devices without computing capabilities are not taken into consideration in regards to communication protocols, since they communicate at the level of hardware that typically does not require interoperability features. Other smart objects, such as smart phones and smart watches, can be considered IoT devices as well in the context of communication protocols. In that case, however, proprietary communication protocols from major vendors are typically implemented. The IoT data generated is communicated with the fog abstraction layer commonly using the REST HTTP protocol, which provides flexibility and interoperability for developers to create RESTful (Representational State Transfer) web services. The latter is critical to remaining backwards compatible with the existing computing infrastructure, running on local computers, servers, or cluster of servers. The local resources are commonly referred to as fog nodes [26] and are able to filter the received data to be either consumed locally or forwarded to cloud for further computations. While the cloud is usually perceived as a unique entity, in reality, cloud services can use more than one cloud provider to meet requirements for reliability, scalability and economics. For this reason, clouds usually support different communication protocols, with AMQP and REST HTTP being among the most used ones. Since HTTP is widely accepted and compatible with the current Internet, the natural question would be whether to use HTTP in the IoT and fog layers. However, this protocol, despite its popularity, has been shown to exhibit a lot of performance issues when used in constrained nodes, as we will discuss later in the survey.

2.2 On Communication Protocols

In general, the candidate communication protocols differ in their interaction models, i.e., request-reply and publish-subscribe. The request-reply communication model is one of the most basic communication paradigms. It represents a message exchange pattern especially common in client/server architectures. It allows a client to request information from a server that receives the request message, processes it and returns a response message. This kind of information is usually managed and exchanged centrally. The two most known protocols based on the request/reply model are REST HTTP and CoAP. Fig.2 shows examples of different client/server interactions, for three HTTP versions (i.e., v.1.0, v.1.1 and v.2.0) as well as for CoAP. In HTTP 1.0, the TCP connection is closed after a single HTTP request/reply pair. In HTTP 1.1, a keep-alive-mechanism was introduced, where

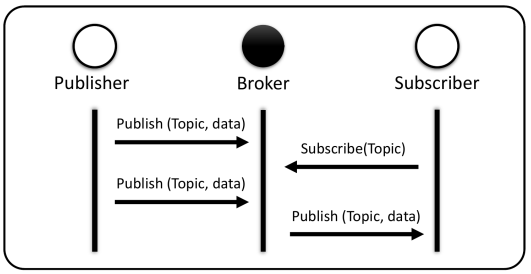


Fig. 3. Publish-Subscribe model, for example: MQTT, DDS and AMQP

a TCP connection could be reused for sending multiple requests to the server without waiting for a response (pipelining). Once the requests are all sent, the browser starts listening for responses and HTTP 1.1 specification requires that a server must send its responses to those requests in the same order that the requests were received. The new HTTP 2.0 introduces a multiplexing method by which multiple HTTP requests can be sent and responses can be received asynchronously via a single TCP connection. The fourth interaction shown is for CoAP, and unlike the others it does not depend on an underlying reliable TCP connection to exchange request/reply messages between the client and the server. The publish-subscribe model, on the other hand, emerged out of the need to provide a distributed, asynchronous, loosely coupled communication between data generators and destinations. The solution appears today in the form of numerous publish-subscribe Message-Oriented Middlewares (MoM) [27] and recently has been a subject of numerous research efforts [28–31].

In this survey, of particular interest are the protocols based on the publish-subscribe interaction model as an alternative to the traditional request-reply (client-server) model. We can see an example of this interaction model, that consists of three parties, publisher, subscriber and a broker presented in Fig. 3. Here, the client with a role of a subscriber does not have to request information from the server. Instead of the request, the subscriber interested in receiving messages will subscribe to particular events (topics) within the system. The client subscribes to the broker, the central point in this architecture, responsible for filtering all incoming messages and routing them accordingly between publishers and subscribers [32]. The third party is the publisher that serves as the information provider. When an event about a certain topic occurs, it publishes it to the broker who sends the data on the requested topic to the subscriber. For these reasons, publish-subscribe interaction model can be described as an event-based architecture [33]. This interaction model is interesting for the applications of IoT, fog and cloud computing systems due to its ability to provide scalability and simplify interconnections between different devices, by supporting dynamic, many-to-many and asynchronous communication [34].

Comparing the two basic models, i.e., request-reply and publish-subscribe, we can observe that the publish-subscribe model has many benefits: i) publishers and subscribers do not need to know about the existence of each other; ii) one subscriber can receive information from many different publishers and one publisher can send data to many different subscribers (many-to-many communication is supported); iii) publisher and subscriber do not need to be active at the same time to exchange information, because the broker (working as a sort of queuing system) can store messages for clients that are not currently connected [35]. There are many standardized messaging protocols currently implementing a publish/subscribe interaction model, most notably MQTT, AMQP and DDS. However, request-reply model also has some advantages. In cases where the capacity of the server side for processing multiple clients requests is not an issue it makes more sense

Table 1. Application layer protocols main features comparison

Protocol	Req.-Rep.	Pub.-Sub.	Standard	Transport	QoS	Security
REST HTTP	✓		IETF [41]	TCP	-	TLS/SSL
MQTT		✓	OASIS [42]	TCP	3 levels	TLS/SSL
CoAP	✓	✓	IETF [43]	UDP	Limited	DTLS
AMQP	✓	✓	OASIS [44]	TCP	3 levels	TLS/SSL
DDS		✓	OMG [45]	TCP/UDP	Extensive	TLS/DTLS/DDS sec.
XMPP	✓	✓	IETF [46]	TCP	-	TLS/SSL
HTTP/2.0	✓	✓	IETF [47]	TCP	-	TLS/SSL

to use already proven and reliable request-reply interactions. So, the choice of the model depends on the application scenario for which it will be used. Finally, some protocols support both request-reply and publish-subscribe interaction models. This includes XMPP protocol, and the new version of the HTTP - HTTP2.0, which supports the server push option, as discussed in Section 3.1. IETF has also released a draft describing a Publish-Subscribe Broker for other protocols of interest, such as CoAP [36]. In an attempt to solve the message exchange, over the time few other solutions emerged, such as the WebSockets protocol [37] or using HTTP over QUIC (Quick UDP Internet Connections) protocol. In case of WebSocket, although it is used for real-time pushing of data from a server to a web client and enable persistent connections with simultaneous bidirectional communication, it is not designed for resource constrained devices [21]. QUIC is also rather noteworthy, as a novel transport protocol creating a wave of the new research efforts [38–40]. Since QUIC has not been standardized yet, it maybe too early to predict its possible application and impact in IoT based solutions. For these reasons, and despite their novelty, WebSockets and QUIC are out of the scope in this survey.

3 COMMUNICATION PROTOCOLS OVERVIEW

This section describes the above mentioned protocols based on their main features, as summarized in Table 1. In a nutshell, Table 1 summarizes the standardization status, interaction model, quality of service options, transport protocol and security mechanisms. MQTT, AMQP, XMPP and REST HTTP, are designed to run on networks that use TCP, while CoAP uses UDP as the underlying transport. DDS primarily uses UDP as its underlying transport, but it also supports TCP. As mentioned in the previous section, MQTT, AMQP and DDS implement a publish/subscribe model, while REST HTTP and CoAP implement a request/reply interaction model. MQTT, AMQP and CoAP protocols provide very basic QoS support for delivering messages. MQTT and AMQP implement three different QoS levels, while in CoAP request and reply messages are limited to two. The QoS in REST HTTP and XMPP is provided by the underlying transport protocols. DDS, on the other hand, provides a rich set of QoS policies with over 20 different QoS options defined by the standard [15]. Most of these protocols choose TLS or DTLS protocol as security mechanisms. Readers interested more in applications of these protocols in various segments (IoT, fog, and cloud) and less so in the protocols design itself, or readers familiar with individual protocols, can skip this section or parts of it, and use the overview in Table 1 to follow up on further discussions in Section 4.

3.1 Hyper Text Transport Protocol (HTTP)

This protocol is the fundamental client-server model protocol used for the Web, and the one most compatible with existing network infrastructure, used by the web developers on a daily basis. Currently, the most widely accepted version of this protocol is HTTP/1.1. Communication between

Fig. 4. REST HTTP interaction model

a client and a server occurs via a request/response messaging, with client sending an HTTP request message and server then returning a response message, containing the resource that was requested in case the request was accepted. Recently, HTTP has been associated with REST [48], a guideline for developing web services based on a specific architectural style in order to define the interaction between different components. Because of the success of RESTful Web services, there has been a lot of effort in bringing this architecture into IoT based systems by combining HTTP and REST. The combination of HTTP protocol with REST is commendable, because the devices can make their state information easily available, due to a standardized way to create, read, update, and delete data (the so-called CRUD operations). According to this mapping, the operations for creating, updating, reading and deleting resources correspond to the HTTP POST, GET, PUT and DELETE methods, respectively. For developers, the fact that REST establishes a mapping of these CRUD operations with HTTP methods, means that they can easily build a REST model for different IoT devices [49]. The presentation of the data is not pre-defined and as such, the type is arbitrary, with the most common being JSON and XML. In most cases, IoT standardizes around JSON over HTTP. Fig. 4 illustrates an example of a REST HTTP request/reply interaction between two clients and one server . First one of the REST HTTP client wants to add a resource on a server side. For this it is necessary to specify, in the header of the POST method, the root of the resource that will be added */resources*, the HTTP version, the *Content-type*, which in this case is a JSON file that represents a specific resource, and finally the JSON object itself. The response from the server specifies whether the request was successful, by specifying the HTTP standard status codes (e.g., 201, resource created). For the second client to get this new resource, the GET method has to be specified with the specific URI (e.g. */resources/1*), which contains the root of the resource and the id of the resource itself. The server will return the JSON object representing the resource. It is worth to mention that beside the simple communication which REST HTTP offers it also has the abundant support and available frameworks making it a default way of web communication, and all servers and client side drivers support it.

Regarding the transport protocol used, HTTP uses TCP. While using TCP provides reliable delivery of large amounts of data which is an advantage in connections that do not have strict latency requirements, it creates challenges in resource constrained environments [50]. One of the main problems is that the constrained nodes most of the time send small amounts of data sporadically and setting up a TCP connection takes time and produces unnecessary overhead. For QoS, HTTP does not provide additional options, but instead it relies on TCP, which guarantees successful delivery as long as connection is not interrupted.

HTTP as a security mechanism uses the very well-known TLS [51] for enabling secure encrypted communication channel, resulting in a secure version of HTTP, also known as HTTPS. The first part of securing the client-server data exchange is a TLS handshake, implemented as an exchange of a 'client hello' and a 'server hello' messages where they have to agree upon a cipher suite, which is a combination of algorithms they will use to assure secure settings. After that, the client and server side exchange keys based on the agreed key exchange algorithm. The result is an exchange of messages encrypted with a shared secret key. The data is encrypted to prevent anyone from

listening to and understanding the content. In systems that will include resource constrained nodes, current TLS implementation (TLS version 1.2) through its handshake process adds additional traffic with each connection establishment that can deplete the computing capabilities of these devices. Efforts are being made in developing a new TLS version 1.3 that will make TLS handshake faster and lighter, as more convenient for IoT [52, 53].

While in general, HTTP presents one the most stable protocol options, there are still a few issues that have lead to the exploration of alternative protocol solutions, due to HTTP complexity, long header fields and high power consumption. Furthermore, HTTP uses the request/reply paradigm, which is not suitable for push notifications, where the server delivers notifications to the client without a client request. Moreover, the TCP protocol overhead maybe too large (three way handshake), especially in case of simple computing nodes in IoT architectures [54]. HTTP does not explicitly define QoS levels and requires additional support for it. This has led to modifications and extension of HTTP, most notably in form of HTTP/2.0 [47], that introduced a number of improvements, some of which are especially relevant in IoT context. HTTP/2.0 enables a more efficient use of network resources and a reduced latency by introducing compressed headers, using a very efficient and low memory compression format, as well as allowing multiple concurrent exchanges on the same connection [55]. These features are particularly interesting for the IoT as it means the size of packets is significantly smaller, making it a more adequate option for constrained devices. Additionally, it introduces the so-called server push, which means the server can send content to clients with no need to wait for their requests. The drawbacks of this version of the protocol in IoT based systems are not known yet, as to the best of our knowledge there are no implemented and tested solutions reported in the literature, as of today. It is however likely that one of the drawbacks will be the same as found in HTTP 1.1, i.e., the utilization of TLS protocol as the security mechanism.

3.2 Constrained Application Protocol (CoAP)

This protocol was designed by the Constrained RESTful Environments (CoRE) working group of IETF [43] for the use in constrained devices with limited processing capabilities. Similar to HTTP, one of its most defining characteristics is its use of tested and well accepted REST architecture. With this feature CoAP supports request/response paradigm just like REST HTTP, and especially so for constrained environments. CoAP is considered a lightweight protocol, so the headers, methods and status codes are all binary encoded, thus reducing the protocol overhead in comparison with many protocols. It also runs over less complex UDP transport protocol instead of TCP, further reducing the overhead. When a CoAP client sends one or multiple CoAP requests to the server and gets the response, this response is not sent over a previously established connection, but exchanged asynchronously over CoAP messages. The price paid for this reduction is reliability. It should be noted that because of the reduced reliability features, which is known when using UDP, IETF has created an additional standard document, opening up the possibility of CoAP running over TCP [56]. However, at this moment this feature is still in its early stages.

CoAP relies on a structure that is divided into two logically different layers. One of the layers, the so-called request/response layer, implements RESTful paradigm and allows for CoAP clients to use the HTTP-like methods when sending requests. In other words, clients can use GET, PUT, POST or DELETE methods to manage the URI identified resources in the network [57]. Just like in HTTP, for its requests for obtaining data from the server, for instance when obtaining the sensor values, client will use method GET with a server URL, and as a reply will receive a packet with that data. The request and responses are matched through a token; a token in the response has to be the same as the one defined in the request. It is also possible for a client to push data, for example updated sensor data, to a device by using method POST to its URL. As we can see, in this layer CoAP uses

the same methods as REST HTTP. What makes COAP different from HTTP is the second layer. Because UDP does not ensure reliable connections, CoAP relies on its second structural layer for reliability, called the message layer, designed for retransmitting lost packets. This layer defines four types of messages: CON (Confirmable), NON (non-confirmable), ACK (Acknowledgement), and RST (reset). The CON messages are used for ensuring reliable communication, and they demand an acknowledged from the receiver side with an ACK message. Precisely this feature that marks whether the messages need the acknowledgement is what enables QoS differentiation in CoAP, albeit in a limited fashion.

CoAP has an optional feature that can improve the request/response model by allowing clients to continue receiving changes on a requested resource from the server [58] by adding an *observe* option to a GET request. With this option, the server adds the client to the list of observers for the specific resource, which will allow the client to receive the notifications when resource state changes. Instead of relying on repetitive polling to check for changes in resource state, setting an *observe* flag in a CoAP client's GET request, allows an interaction much closer to a publish-subscribe paradigm with a server alerting a client when there are changes. In an attempt to get even closer to publish/subscribe paradigm, IETF has recently released the draft of Publish-Subscribe Broker that extends the capabilities of CoAP for supporting nodes with long interruptions in connectivity and/or up-time [36], with preliminary performance evaluations showing promising results [59].

As a security mechanism CoAP uses DTLS [60] on top of its UDP transport protocol. It is based on TLS protocol with necessary changes to run over an unreliable connection. The result is a secure CoAPS protocol version. Most of the modifications in comparison to TLS include features that stop connection termination in case of lost or out of order packets. As an example, there is a possibility to retransmit handshake messages. Handshaking process is very similar to the one in TLS, with the exchange of client and server 'hello' messages, but with the additional possibility for a server to send a verification query to making sure that the client was sending its 'hello' message from the authentic source address. This mechanism helps prevent Denial-of-Service attacks. Through these messages, client and server also exchange supported cipher suits and keys, and agree on the ones both sides support, which will further be used for data exchange protection during the communication.

Since DTLS was not originally designed for IoT and constrained devices, new versions optimized for the lightweight devices have emerged recently [61, 62]. Some of the DTLS optimization mechanisms with a goal of making it more lightweight include IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) header compression mechanisms to compress DTLS header [63]. Because of its limitations, optimizing DTLS for IoT is still an open issue [13, 64].

3.3 Message Queue Telemetry Transport Protocol (MQTT)

MQTT is one of the lightweight messaging protocols that follows the publish-subscribe paradigm, which makes it rather suitable for resource constrained devices and for non-ideal network connectivity conditions, such as with low bandwidth and high latency. MQTT was released by IBM, with its latest version MQTT v3.1 adopted for IoT by the OASIS [42]. Because of its simplicity, and a very small message header comparing with other messaging protocols, it is often recommended as the communication solution of choice in IoT. MQTT runs on top of the TCP transport protocol, which ensures its reliability. In comparison with other reliable protocols, such as HTTP, and thanks to its lighter header, MQTT comes with much lower power requirements, making it one of the most prominent protocol solutions in constrained environments.

There are two communication parties in MQTT architecture that usually take the roles of publishers and subscribers, clients and servers/brokers. Clients are the devices that can publish messages, subscribe to receive messages, or both. The client must know about the broker that

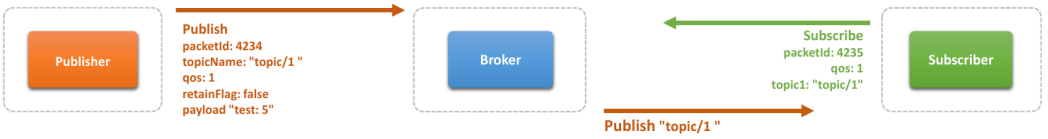


Fig. 5. MQTT interaction model

it connects to, and for its subscriber role it has to know the subject it is subscribing to. A client subscribes to a specific topic, in order to receive corresponding messages. However, other clients can also subscribe to the same topic and get the updates from the broker with the arrival of new messages. Broker serves as a central component that accepts messages published by clients and with the help of the topic and filtering delivers them to the subscribed clients. In MQTT, a publish-subscribe interaction model can be used as illustrated in Fig. 5. The communication takes place between a broker and two MQTT clients, a publisher and a subscriber. For a device to have a role of the broker, it is necessary to install MQTT broker library, for example Mosquitto broker [65], which is one of best known open source MQTT brokers. It should be noted that there are various other MQTT protocol brokers that are open for use, which differ by way of implementation of the MQTT protocol. Some of them are Emqttd [66], ActiveMQ [67], HiveMQ [68], IBM MessageSight [69], JoramMQ [70], RabbitMQ [71], and VerneMQ [72]. The clients are realized by installing MQTT client libraries. The publisher creates labeled topics into the Broker, as shown in Fig. 5. Topics in MQTT are treated as a hierarchy, with strings separated by slashes that indicate the topic level [73]. One MQTT publisher can publish messages to defined set of topics. In this case client will publish the topic: *topic/1*. This information will be published to the broker which can temporally store it in a local database. The subscriber interested in this topic sends a subscribe message to a broker, specifying the same topic.

For QoS, MQTT defines three QoS levels, QoS 0, 1, and 2 [42, 74]. The choice of the level can be defined both in the publish and the subscribe message body. QoS 0 delivers on the best effort basis, without confirmation on message reception. This is a choice in cases where some sensors gather telemetry information over a longer time period, and where the sensors values do not change significantly. It is then acceptable if sometimes the messages are missing, because the general sensor value is still known since most of the message updates have been received. The next level of guarantee is QoS 1, which assures that messages will arrive, so a message confirmation is necessary. This means that receiver must send an acknowledgment, and if it does not arrive in a defined period time, publisher will send a publish message again. The third option, QoS 2, guarantees that the message will be delivered exactly once without duplications. The amount of resources necessary to process MQTT packet increases with the higher chosen QoS level, so it is important to adjust the QoS choice to specific network conditions.

Another important feature MQTT offers is the possibility to store some messages for new subscribers by setting a 'retain' flag in published messages. If there is nobody interested in a topic on which the publisher sends the updates, broker will discard the published messages. But, in some situations, especially when the state of the followed topic does not change often, it is useful to enable for new subscribers to receive the information on that topic. In this default case new subscribers would have to wait for the state to change in order to receive a message about the topic. By setting a 'retain' flag to value: *true*, broker is informed that it should store the published message, so it could be delivered to new subscribers.

MQTT uses TCP which can be critical for constrained devices. To this end, a solution has been proposed as MQTT for Sensor Networks (MQTT-SN) version that uses UDP and supports topic

name indexing [75]. This solution does not depend on TCP, but instead uses UDP as faster, simpler, and more efficient transport option over a wireless link [76]. The other important improved feature is the reduced size of the payloads. This is done by numbering the data packets with numeric topic *id's* rather than long topic names. The biggest disadvantage is that at the moment MQTT-SN is only supported by a few platforms, and there is only one free broker implementation known, called Really Small Message Broker [2].

Since it was designed to be as lightweight, MQTT does not provide encryption, and instead, data is exchanged as plain-text, which is clearly an issue from the security standpoint. Therefore, encryption needs to be implemented as a separate feature, for instance via TLS, which on the other hand increases overhead. Authentication is implemented by many MQTT brokers, through one of the MQTTs control type message packets, called CONNECT. Brokers require from clients, that when sending the CONNECT message, they should define username/password combination before validating the connection, or refusing it in case the authentication was unsuccessful. Overall, security is an ongoing effort for MQTT [77], and probably the most important one since MQTT is one of the most widely adopted and mature communication protocol solutions. Solving the security issue would create an important and big advantage for MQTT, in comparison with other available solutions.

3.4 Data Distribution Service (DDS)

DDS is a real-time data-centric interoperability standard that uses a publish-subscribe interaction model, as defined by the Object Management Group (OMG) [45]. Unlike some other publish-subscribe protocols, DDS is decentralized and based on peer-to-peer communication, and as such does not depend on the broker component. In DDS, publishers and subscribers can communicate as peers through the data bus, enabling asynchronous data exchange based on their interests. The fact that there is no broker also decreases the probability of system failure because there is no single point of failure for the entire system, making a system more reliable. Both communication sides are decoupled from each other, and a publisher can publish data even if there are no interested subscribers. The data usage is fundamentally anonymous, since the publishers do not enquire about who consumes their data.

One of the salient features of DDS protocol is its scalability, which comes from its support for dynamic discovery. The discovery process, achieved through DDS built-in discovery protocol, allows for subscribers to find out which publishers are present, and to specify the information they are interested in with the defined desired quality of service, and for publishers to publish their data [78]. DDS ensures that proper publish and subscribe nodes will be connected and that the data exchange will be in real-time. Another important and unique characteristic in DDS is its data-centricity, unlike most protocols that are message-centric. For data-centric paradigm what matters the most is the data that clients want to access to, so the focus is on the content information itself. Thus, DDS enables an architecture where participating nodes understand the data value in a consistent manner. In DDS, the data type and content define the communication, whereas in message-centric protocols the focus lies on the operations and mechanisms for delivering that data. The data-centric approach of DDS can be used when system architects define the so-called *topics* by grouping together data items that can logically be related with a goal of ensuring better scalability and performance results.

The main entities in DDS architecture include: Domain, Domain Participant, Topic, publisher, subscriber, Data Writer and a Data Reader [79]. Publishers and Subscribers are divided into Domains, a virtual concept entity that allows the isolation of communication within nodes that have common interests. Domain Participant is the entry point for message exchange in specific domains, which associates publishers and subscribers and the domains they belong to. It is used to create publishers,

subscribers, Data Writers, Data Readers and topics within a domain. The DDS implementation middleware, with data as the main point that will define how the interactions will be conducted, defines how data is structured, changed and accessed in an abstract data space, with a goal of creating a globally shared data [80]. The way this is achieved is through a data space abstraction where all the clients can access to read or store their data, known as Global Data Space (GDS). It is in GDS where the DDS dynamic discovery feature comes into play by allowing publishers and subscribers that join the GDS to automatically discover their mutual existence as well as their interests. The exchange information unit among DDS nodes in GDS is a Topic, and is defined by a name, a data type and a set of QoS policies. Publishers and subscribers are the entities for data distribution and consumption, which publish and receive data through the GDS, but they can not do it on their own. Instead, publishers use Data Writers to send data and subscribers use Data Readers to receive data [81] with the matching between the two through topics, that is in order to communicate with each other, publishers and subscribers must use the same topic (same name, type and a compatible QoS).

DDS uses UDP by default, but it can also support TCP. Another important protocol in DDS is the Real Time Publish Subscribe (RTPS) [82] wire protocol, which represents DDS interoperability protocol that allows data sharing among different vendor implementations. One of the advantages of using DDS is a wide set of QoS policies offered (over 20 QoS as defined by the standard). When sending data, the QoS policies of each topic, Data Writers and publishers control how and when the data is sent to the middleware. On the other side, topic QoS, Data Readers and subscribers control the behavior when receiving data.These various policies manage a myriad of DDS features, such as discovery of distributed remote entities, data delivery, data availability, time, and resource utilization [83].

For a security mechanism DDS implements various solutions. Based on a transport protocol of choice, TLS can be used in case TCP is the transport protocol, or DTLS protocol in case UDP is used. Similarly for TLS also DTLS brings too much overhead in constrained environments, for which improved mechanisms have been proposed. To this end, the OMG (Object Management Group) DDS Security Specification defines an extensive Security Model and Service Plugin Interface (SPI) architecture designed for for DDS implementations suitable in IoT systems [84]. The question of security specification is currently still an open one for DDS and it is expected that new additions will be implemented in the future. One of the additions expected is a secure discovery mechanism capable of establishing Secure Transport flows between DDS-based applications that have matching security classification, as proposed in [85].

DDS is an important solution for IoT-based environments for its decentralized publish/subscribe architecture and its support for implementation in both powerful devices and constrained devices [14]. A challenge of DDS is that it has not been widely used, though this may change with emerging open source DDS implementations ready for testing, such as OpenDDS [86].

3.5 Advanced Message Queueing Protocol (AMQP)

AMQP is an open standard protocol that follows the publish-subscribe paradigm as defined by OASIS [44], designed to enable interoperability between a wide range of different applications and systems, regardless of their internal designs. Originally it was developed for business messaging with the idea of offering a non-proprietary solution that can manage a large amount of message exchanges that could happen in a short period of time in a system. This AMQP interoperability feature is significant as it allows different platforms, implemented in different languages, to exchange messages, which maybe especially useful in heterogenous systems [87].

AMQP has been implemented in two very different versions, AMQP 0.9.1 and AMQP 1.0, each with a completely different messaging paradigm. AMQP 0.9.1 implements the publish-subscribe

paradigm, which revolves around two main AMQP entities, both part of an AMQP broker: the exchanges and the message queues. The exchanges represent a part of the broker that is used to direct the messages received from publishers. The publishing of messages to an exchange entity is the first step in the process, and after that messages are routed into one or more appropriate queues. This depends on whether there are more subscribers interested in a particular message, in which case the broker can duplicate the messages and send their copies to multiple queues. A message will stay in the queue until it is received by a subscriber. This routing process, that actually links exchanges and queues, depends on the so-called bindings, which are predefined rules and conditions for message distribution. The newer version of AMQP protocol, AMQP 1.0, is on the other hand not tied to any particular messaging mechanism. While the older versions of the protocol used specifically the above mentioned publish-subscribe approach with architecture that consists of exchanges and the message queues, new AMQP implementations follow a peer-to-peer paradigm, and can be used without a broker in the middle. Broker is present only in the communication that needs to provide store-and-forward mechanism, while in other cases direct messaging is possible. This option of supporting different topologies increases the flexibility for the possible AMQP based solutions, enabling different communication patterns, such as client-to-client, client-to-broker, and broker-to-broker [88]. It should be noted that a significant amount of infrastructures still use the older AMQP version 0.9.

AMQP uses TCP for reliable transport, and in addition it provides three different levels of QoS, same as MQTT. Finally, the AMQP protocol provides complementary security mechanisms, for data protection by using TLS protocol for encryption, and for authentication by using SASL (Simple Authentication and Security Layer).

With all the features it offers, AMQP has relatively high power-, processing- and memory related requirements, making it a rather heavy protocol, which has been its biggest disadvantage in IoT-based ecosystems. This protocol is better suited in the parts of the system that is not bandwidth and latency restricted, with more processing power.

3.6 Extensible Messaging and Presence Protocol (XMPP)

XMPP is an open standard messaging protocol formalized by IETF [46], and was initially designed for instant messaging and the exchange of messages between applications. It is a text-based protocol, based on Extensible Markup Language (XML) that implements both client-server and publish-subscribe interaction [89], running over TCP. In IoT solutions it is designed to allow users to send messages in real time, in addition to managing the presence of the user. XMPP allows instant messaging applications to achieve all basic features, including authentication, end-to-end encryption and compatibility with other protocols [17].

XMPP supports client-server interaction model, but there are new extensions that enable also for generic publish-subscribe model to be used. These extensions enable XMPP entities to create topics and publish information; an event notification is then broadcasted to all entities that have subscribed to a specific node. This functionality is rather important for IoT-fog-cloud scenarios, being the foundation for a wide variety of applications that require event notifications. The clients and servers in XMPP communicate with each other using XML streams to exchange data in the form of XML stanzas (semantic structured data units) [14]. Three types of stanzas are defined: <presence/>, <message/> and <iq/> (info/query). A message stanza defines a message title and contents and it is used to send data between XMPP entities. Message stanzas do not receive an acknowledged by the receiving entity, whether it is client or server. A presence stanza shows and notifies entities of status updates, having the role of subscription in XMPP. If there is an interest in the presence of some JID (Jaber ID - a node address in XMPP), a client subscribes to their presence and every time that a node sends a presence update a client will be notified. An iq stanza pairs

Table 2. Ongoing efforts for constrained environments adaptation

Protocol	Open challenges and efforts in constrained environments
REST HTTP	TLS version 1.3; HTTP/2.0 version
MQTT	TLS version 1.3; MQTT-SN (based on UDP)
CoAP	DTLS optimization
AMQP	not recommended for constrained devices
DDS	DDS security specification
XMPP	light-weight XMPP publish-subsribe scheme

message senders and receivers. It is used to get some information from the server, for example information about the server or its registered clients, or to apply some settings to the server. Its function is similar to HTTP GET and POST methods.

One of the most important characteristics of this protocol are its security features, which makes it one of the more secure messaging protocols surveyed. Unlike the other protocols surveyed, for example MQTT and CoAP, where the TLS and DTLS encryptions are not built-in within the protocol specifications, XMPP specification already incorporates TLS mechanisms, which provides a reliable mechanism to ensure the confidentiality and data integrity. New additions to the XMPP specifications also include extensions related to security, authentication, privacy and access control. Beside TLS, XMPP implements SASL, which guarantees server validation through an XMPP-specific profile [90].

Since XMPP was initially designed for instant messaging there are some notable potential weakness. By using XML, the size of the messages makes it inconvenient in the networks with bandwidth constraints. Another downside is the absence of reliable QoS guarantees. Because XMPP runs on top of a persistent TCP connection and lacks an efficient binary encoding, it has not been practical for use over lossy, low-power wireless networks often associated with IoT technologies. However, lately, there has been a lot of effort to make XMPP better suited for IoT [91–93]. In [94], a lightweight XMPP publish/subscribe scheme was presented for resource constrained IoT devices, thus improving and optimizing the existing version of the same protocol.

Throughout this section we saw that one of the main features of all of the above mentioned protocols relevant for their potential utilization and correct placement in an integrated IoT, fog and cloud system is their applicability in resource constrained devices. Table 2 offers a summarized overview of the ongoing efforts of making these protocols more compatible for constrained environments.

4 PERFORMANCE COMPARISON

The performance analysis and comparison of communication protocols is still a lively area in the research community. In this section, we survey and analyze the studies reported in different testbed scenarios. An overview of studies focused on performance is shown in table 3 and are described in more detail in the following subsections.

4.1 Latency

When comparing different parameters for communication protocols, especially for IoT related application, latency comes as one of the priorities. In [95], authors have analyzed the behavior of two HTTP and MQTT in a fog-to-cloud IoT based architecture scenarios. The results of the experiments have shown that the measured response times for the requests were shorter for MQTT

then the ones for HTTP. In [96], a Raspberry Pi based home automation system was used along with a web server and smart phones to measure latency generated by the MQTT (Mosquitto) and HTTP (REST) based architectures. As a result, MQTT-based architecture produced lower latency. In [97], it was shown that MQTT messages had experienced lower delays than CoAP for lower packet loss and higher delays than CoAP for higher packet loss. The comparison of these two protocols has also been conducted in [98] where authors assessed latency by measuring RTT. The results have shown that the average CoAP RTT was more than 20% shorter than MQTT. Another comparison of RTT in MQTT and CoAP [99] was conducted in two scenarios, local area network and an IoT network, with average RTT being from two to three times higher in the IoT network scenario. The results showed that MQTT with QoS0 had lower RTT in comparison with CoAP, while MQTT with QoS1 had the higher RTT due to the presence of both transport and application layer ACKs. In [100], the latency of MQTT and CoAP was analyzed for different QoS levels in a network without congestion. These conditions favored CoAP because it required fewer bytes to transfer the same message with a shorter delay regardless of the QoS level. Regarding the latency, MQTT latencies were measured in the order of milliseconds, and CoAP latencies as low as hundreds of microseconds. However, it is important to notice that in the cases of less reliable networks, MQTTś underlying TCP protocol will be an important advantage and the results would be different.

The latency comparison in [88] for the two broker based protocols, AMQP and MQTT, for increasing payload size showed that when transferring relatively small payloads the latencies of the two protocols are almost the same, but when transferring huge payloads MQTT yields a lower latency.

In [101], CoAP was compared with HTTP in a machine-to-machine communication scenario with devices deployed on a top of the vehicles and equipped with the gas sensors, weather sensors), location (GPS) and a mobile network interface (GPRS). The time needed to transfer a CoAP message over mobile network was almost three times shorter then the time required when HTTP messages are used. Another comparison of these two protocols was conducted in [102] as potential communication protocols for smart grid devices over the Arduino hardware platform. The results have shown the HTTP had a longer response time. An emulation-based quantitative performance assessment of CoAP in comparison with HTTP was conducted in [103], taking into account different QoS levels of CoAP (with confirmable and non-confirmable messages). Again, HTTP showed a comparably poor delay performance.

Including more protocols in their analysis, the authors in [104] compared the performance of IoT protocols MQTT, DDS and CoAP in a medical application scenario using a network emulator. DDS outperformed MQTT in terms of experienced telemetry latency in various poor network conditions. The UDP based CoAP performed well for the applications that required low latency; however, since it is UDP based there was a significant amount of unpredictable packet loss. In [49], authors compared web performance of publish/subscribe IoT messaging protocols MQTT, AMQP, XMPP, and DDS by measuring the latency of sensor data message delivery and the message throughput rate. The results have to be taken with a reservation because they heavily depended on the message broker and JavaScript client implementations. The shortest latency was produced by the MQTT protocol, followed by AMQP, while the difference between XMPP and DDS was negligible. Authors in [16] compared MQTT, CoAP, HTTP and AMQP messaging protocols based on their average latency among other parameters. The results have shown the highest latency in HTTP, followed by AMQP and MQTT respectively, with CoAP having the lowest latency results. Finally, it should be noted that only a few papers compare a new HTTP version, HTTP2/0 with the other messaging protocols, or evaluate HTTP2/0 performances in IoT scenarios. The paper [105] compares the IoT adapted SPDY (Speedy) protocol, which was used a basis for HTTP/2 with CoAP and HTTP. The experiments showed that CoAP has the lowest download time and the least

number of bytes transferred. In [106], the data transfer time for CoAP and HTTP/2 was compared, with HTTP/2 having better results in high congestion scenarios and CoAP in lower congestion scenarios.

To summarize, even though there is no comparative study of all the protocols discussed here, we can conclude that the latency is heavily influenced by the underlying transport protocol, and the use of TCP in MQTT, AMQP, HTTP and XMPP is a major factor that causes higher latency values than in CoAP and UDP based DDS.

4.2 Bandwidth consumption and throughput

In [97] MQTT and CoAP have been analysed using the common middleware in terms of bandwidth consumption that was measured as total data transferred per message. In the cases where message size was small, and independently of the increase of packet loss rate, CoAP consumed less bandwidth than MQTT. The authors in [87] calculated protocol efficiency, as as the ratio between the number of useful information bytes and the total number of bytes exchanged at application and transport layers, and used it to compare MQTT and CoAP. The results showed higher efficiency for CoAP. An emulation-based quantitative performance assessment of CoAP in comparison with HTTP was conducted in [103] in the dynamic network environment. This scenario included a large amount of devices transferring data at the same time, which is a typical case in IoT environments. In order to achieve a higher utilization results have shown that it is better to use CoAP. Alongside latency, the authors in [104] compared bandwidth consumption for three protocols MQTT, DDS (with TCP as a transport protocol) and CoAP once as a function of a network packet loss and once as A function of network latency. CoAP generally showed a comparably lower bandwidth consumption that did not increase with increased network packet loss or increased network latency, unlike MQTT and DDS, where bandwidth consumption increased in mentioned scenarios. Also DDS consumed approximately twice the bandwidth of MQTT. In another study, [107], authors have also compared three protocols, this time MQTT (taking into consideration all three QoS level options), CoAP and REST HTTP in terms of bandwidth measurements. The scenario in question covered IoT to cloud communication and the results heavily depended on the size of the payloads that were being transferred. In the case of small payloads CoAP used the least amount of bandwidth, followed by MQTT and REST HTTP. However, when the size of payloads increased, the best performances were measured for REST HTTP.

4.3 Energy consumption

The power/energy consumption is essential in every IoT based system, and the choice of protocols affects the same. In [96] along with latency analysis, authors have compared the energy consumption between MQTT and HTTP, with the results that energy consumed by HTTP was much larger than with MQTT. In [108] authors have analysed average energy consumed by MQTT and CoAP for a constrained gateway device with experimental results showing that CoAP is more efficient in terms of energy, though both of them proved to be efficient. A similar conclusion can be found in [109] where authors have shown that in the simple scenarios MQTT was more suitable for IoT messaging and nodes with no power constraints. CoAP on the other hand has proved to have efficient power management capabilities. In [110] authors provides an evaluation of CoAP compared to HTTP which demonstrated that thanks to the smaller headed and packet size in CoAP results in it having a lower energy consumption. In [111] authors compared the capabilities of AMQP and MQTT under a mobile or unstable wireless network testbed with the conclusion that AMQP offered more aspects related to security and MQTT was more energy efficient. Similar to other performance measures presented in this survey, most of the papers compare a pair of protocols in one study, and no study has evaluated and compared the energy consumption of all candidate communication protocols.

4.4 Security

Security remains one of the most important challenges as pointed out by a large number of research papers [13, 20, 61, 112–116]. Related to security, we focus here on application layer, where it is also necessary to understand the communication challenges related to performance factors such as latency, overhead and packet loss. As mentioned in previous sections, the choice for security mechanism for surveyed protocols is usually based on TLS or DTLS protocol with protocols like HTTP, MQTT, AMQP and XMPP basing their security on TLS, CoAP on DTLS, and DDS supports both options. Both TLS and DTLS start with the handshaking process between client and server side in order to exchange supported cipher suits and keys, and agree on the ones both sides support to assure that further communication happens in a secure communication channel. The difference between the two is in small modifications that allow UDP-based DTLS to run over an unreliable connection. The slight advantage of TLS is that it is a widely used and stable security protocol, with a software client and server support and in available cryptography libraries [61]. In [117] authors presented distinguishing and plaintext recovery attacks against TLS and DTLS with experimental results demonstrating the feasibility of the attacks in realistic network environments for several different implementations of TLS and DTLS. The results reported were in favor of TLS since the attacks proved to be much more serious for DTLS, because of its tolerance of errors.

However, the biggest issue with the implementation of these protocols in IoT-F2C systems is that they were not originally designed for utilization in IoT and constrained devices. Through their handshake process, they add additional traffic with each connection establishment that drains the computing resources. In [118] the use of TLS and DTLS protocols in communication channel was analysed and compared with their corresponding insecure options, TLS communication was compared with regular TCP-based exchange and DTLS communication with regular UDP-based exchange. On average results showed an increase of 6.5% for TLS and 11% for DTLS in overhead, compared to communication without security layer. In resource rich environments that are usually located in the cloud layer, this would not be a problem, but in the IoT to fog layer communication this becomes an important limitation. For these reasons, security is an ongoing effort with a goal of optimizing TLS and DTLS by creating more lightweight versions, or finding alternative solutions.

4.5 Developer's choice

One important factor for any protocol adoption is the choice by system developers. The adoption of the protocols presented here was in fact surveyed collaboratively by Eclipse IoT Working Group, IEEE, Agile-IoT EU and the IoT Council in order to better understand how developers are building IoT solutions [119]. On the question what messaging protocol is used in IoT solutions, the results of that analysis have shown that MQTT and HTTP are the most used and adopted protocols. The reason for this is that MQTT and HTTP REST are currently comparably more mature and more stable IoT standards than other protocols. For many IoT developers, MQTT and HTTP are protocols of choice in their IoT, fog and cloud implementations.

5 CONNECTING THE IOT, FOG AND THE CLOUD

We now focus on the communication protocols and discuss their positioning within a combined IoT-fog-cloud architecture, which per se is an open direction for future research. It should be first noted that no communication protocol has been originally designed for a combined IoT-fog-cloud systems, and there is no unifying standard, and this alone is an ongoing area of research and development. On the other hand, all of the above mentioned protocols are operating in the application layer of the OSI protocol stack, and as such they could, in theory, fill the same role in various parts of the system. This section is dedicated to discussing the pros and cons of single- or multiple protocol

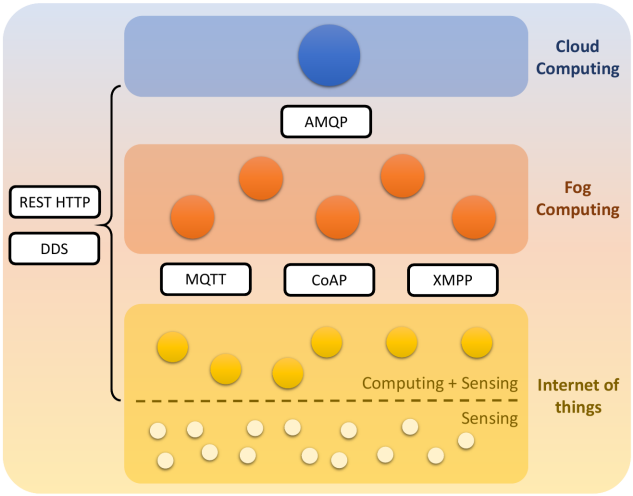


Fig. 6. Application protocols in IoT, Fog and Cloud networks

solutions in various segments, opening this area for broader dissemination and further reserach. We conclude this section with a summarized discussion of other open issues and challenges.

5.1 Protocol solutions based on a single communication protocol

Let us start the discussion with an example. We first note that some of the protocols previously described are more widely used and accepted than the others, such as MQTT and REST HTTP, and are candidates for a single-protocol solutions. The practice of single-protocols solution has many downsides, however, since based on the features considered, it is obvious that each of the protocols can optimally satisfy specific requirements. For example, a protocol that satisfies constrained environment can underperform in the domain that has strict security requirements. With this in mind, we consider two candidate single protocol based solutions in combined IoT, fog and cloud, and these are MQTT and REST HTTP. It should be mentioned that based on its characteristics for this kind of a solution DDS could also be considered, and in fact Vortex DDS Platform offers DDS based solutions both in cloud and fog based IoT systems. However, at this moment, MQTT and REST HTTP based solutions are much more widely accepted and as such are of the more interest here.

5.1.1 REST HTTP as a single protocol solution. Fig. 7 illustrates REST HTTP request/reply interaction in an IoT-to-fog application in smart farming, which we adopted from [120]. A code example of how HTTP methods support CRUD operations in this kind of applications shown in Fig. 8. In this example, animals are equipped with wearable sensors (IoT Client, C) and managed in a fog computing smart farming system (fog server, S). Here, in the header of the POST method the resource to modify is specified */farm/animals*, as well as HTTP version and *Content-type* which in this case is a JSON object that represents a farm animal to be managed by the system (Nicky, the cow). In this example the response from the fog server specifies that the request was successful, with the HTTPS status code *201, resource created.* The GET method only needs to specify the requested resource in the URI (e.g. */farm/animals/1*), which returns, in this example, the JSON representation of the animal with this id from the server. Similarly, the PUT method is used when some specific resource entry needs to be updated, in this case, in the resource URI is specified for the parameter

Fig. 7. REST HTTP request/reply interaction model

Fig. 8. Example of REST HTTP methods for CRUD operations; C: Client, S: Server)

```java
private static Response postAnimal(Animal animal) {
    String uri = "http://localhost:8080/farm/animals";
    RestTemplate rest = new RestTemplate();
    rest.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
    Response response = null;
    try {
        response = rest.postForObject(uri, animal, Response.class);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return response;
}
```

Fig. 9. Example of a HTTP Post request in Java using Spring Framework

to be changed and the current value (e.g., for instance indicating that cow is currently walking, */farm/animals/1?state=walking*). Finally, the DELETE method is used equally than the GET method, but just deleting the resource as a result of the operation. As mentioned in Section 3.1, REST HTTP has many available frameworks which makes its utilization an easy and logical choice. For instance, for Java developers, Spring Framework [121] facilitates the implementation of RESTful web services. We illustrate an example in Fig. 9. This method creates an HTTP POST request to the specified URI parsing an (animal) Java object into a JSON format. The response object maps the response from the server to a Java object in order to be managed by the application.

5.1.2 MQTT as a single protocol solution. Let us use the same example of a smart farm, but in this case for the communication instead of REST HTTP, MQTT protocol is used, as illustrated in Fig. 10. Let us first describe IoT to fog communication, and then extend it to the third abstract layer - the cloud. The local server with the installed Mosquitto library has a role of the broker, in this example a simple off the shelf personal computer (denoted as farm server). A Raspberry Pi serves as a MQTT client, realized by installing MQTT Paho Library that is fully compatible with the Mosquitto broker. This client corresponds to the IoT abstraction layer, representing a

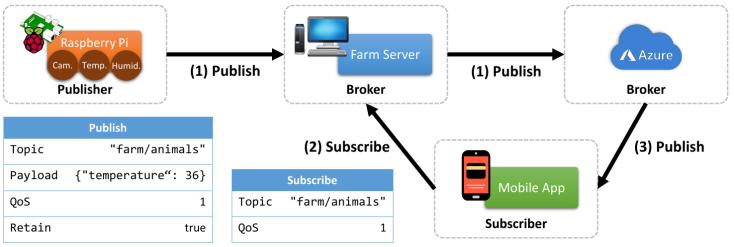


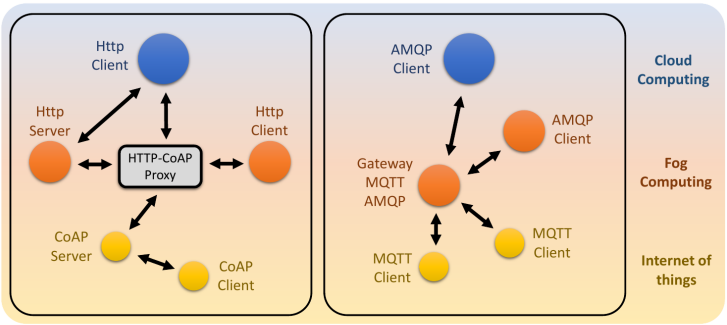
Fig. 10. Connecting the IoT, Fog and the Cloud - MQTT example

device with sensing and computing capabilities. The broker, on the other hand, corresponds to the higher abstraction layer representing a fog computing node, characterized by larger computing and storage capacities. In the proposed smart farm scenario Raspberry Pi is connected to accelerometer, GPS and temperature sensors and publishes data from these sensors to a broker fog node. As explained Section 3.3 topics in MQTT are treated as a hierarchy. One MQTT publisher can publish messages to a defined set of topics, in this case, three topics. For the sensor that measures the temperature in an animal shed, a client will publish the temperature under the following topic: *animalfarm/shed/temperature*. In the same manner, for the sensors that measure GPS location and animal movement through accelerometer, a client will publish the corresponding updates under the following topics: *animalfarm/animal/GPS* and *animalfarm/animal/movement*. This information will be published to the broker which can temporally store it in a local database in case that later another interested subscriber appears.

In addition to a local server that has a role of a fog MQTT broker to which Raspberry Pis that serve as MQTT clients publish data from the sensors, there can be another MQTT broker in the cloud layer. In this case, the information published to the local broker can temporally be stored in a local database and/or transmitted to the cloud. Here, the fog MQTT broker is actually used to bridge all the data to another MQTT broker that is represented by a cloud based instance. With such an architecture, the user with a mobile application can be subscribed to both brokers. In this way, if connection with one of the broker fails, say cloud, the end user has the option of receiving the information from the other one, e.g. fog. This is a salient feature of combined fog and cloud computing systems. By default, mobile application can be configured to first connect to the fog MQTT broker, and if not successful, to connect to the cloud MQTT broker. This MQTT bridging solution between one local broker serving as a fog node and one cloud broker is just one of the possible solutions in IoT-F2C systems. Related work so far has more commonly considered MQTT from the IoT device layer to fog/edge nodes, while the communication from fog to cloud was left to other candidate application layer protocols [122]. This leads us to the solution with multiple communication protocols as we will discuss next.

5.2 Multiple protocol solutions based on a combination of communication protocols

While the single protocol solutions have been popular because of their easier implementation, it is obvious that in IoT-F2C systems it would make sense to combine different protocols. One of the findings of this survey is in fact that individual protocols can be better positioned within parts of the overall system, as illustrated in Fig. 6. Let us consider, for illustration, three abstraction layers of IoT, fog and cloud computing. The devices in the IoT layer are generally considered as constrained. For the sake of this survey, let us consider the IoT layers as the most constrained, cloud the least constrained and fog computing as "somewhere in between." Based on this assumption, we find and



(a) HTTP-CoAP solution (b) MQTT-AMQP solution

Fig. 11. Connecting the IoT, Fog and the Cloud

Fig. 6 shows that between the IoT and the fog abstractions, the current protocol solutions include MQTT, CoAP and XMPP. Between the fog and the cloud, on the other hand, AMQP is one of the main protocols used, together with REST HTTP, which due to its flexibility is also used between IoT and fog layers. At the first sight, however, RESTful HTTP protocol and the newly proposed DDS protocol can be used in all layers. We will get back to this and other observations in the following sections, with more in-depth discussions on single- and multiple protocol solutions.

For solving the communication among two lower abstract layers, that is IoT and fog, the common denominator that determines the suitability of a protocol is the ability to run as a lightweight protocol on constrained devices. On the other hand, this requirement is not necessary for the communication among fog and cloud layer. Based on their characteristics and the scenarios we encountered during our survey, the straightforward solution would include the combination of lightweight protocol between IoT and the fog and a protocol not restricted to the constrained devices between the fog and the cloud. The main issue with these kind of solutions, however, is protocol interoperability and the ease of translating the communication from one protocol to another, presenting also a challenge for recent research efforts [123]. Ideally, in the future, an IoT-F2C system architecture will be independent on the communication protocol used, and will provide integration among different protocols. Since this is not the case at the moment, in order to avoid additional implementation difficulties it makes more sense to combine protocols without significant conceptual differences. To this end, one potential solution is based on the combination of two protocols that follow the same architectural style, REST HTTP and CoAP. The other proposed solution is based on the combination of two protocols that follow the publish-subscribe interaction, MQTT and AMQP. Following the similar concept (both MQTT and AMQP are broker based, CoAP and HTTP both use REST style) makes these combination easier to implement, and require less integration efforts. It should be noted that other combinations are also possible, such as MQTT and REST HTTP, but are more difficult for realization.

5.2.1 REST HTTP-CoAP example. Fig. 11 (a) shows the two request-reply based models, HTTP and CoAP, and their possible placement in an IoT-F2C solution. Since HTTP is one of the best known and adapted protocols in current networks, it is unlikely that it will be completely replaced with other messaging protocols. Among the nodes that present powerful devices, which would be between cloud and fog, REST HTTP is a reasonable solution. On the other hand, for resource constrained devices that communicate between fog and IoT layer it is more efficient to use CoAP. One of the big advantages of CoAP is in fact in its interoperability with HTTP, being that the both protocols are based on REST principles. For this interoperability to work it is necessary

Table 3. Performance comparisons related to application layer protocols

Protocol	Latency	Bandwidth utilization and throughput	Energy consumption	Security	Developer's choice
REST HTTP	[95, 96, 101, 102] [16, 95, 103, 105]	[103, 107]	[96, 110]	[52, 53]	✓
MQTT	[96–98] [88, 99, 100] [16, 49, 104]	[87, 97, 104, 107]	[96, 108, 109, 111]	[77]	✓
CoAP	[97–100] [101–104] [16, 105, 106]	[87, 97, 103, 107]	[108–110]	[61–63] [13, 64]	
AMQP	[16, 49, 88]	-	[111]	-	
DDS	[49, 104]	[104]	-	[84, 85]	
XMPP	[49]	-	-	[90]	
HTTP/2.0	[105, 106]	-	-	-	

to deploy a proxy between them that will allow HTTP clients to request resources from CoAP servers and CoAP clients to request resources from HTTP servers [124] also presented in Fig. 11 (a). The reference information for implementing a proxy that performs translation between HTTP and CoAP is given in an document by CoRE working group [125]. A lot of research effort is put into developing and analysing HTTP-CoAP proxies and mappings between the two [126–131].

5.2.2 MQTT-AMQP example. Fig. 11 (b) alternatively shows two publish-subscribe interaction based models in the same scenario, including MQTT and AMQP. While hypothetically, both protocols could be used for communication among nodes in every abstraction layer, their position should be decided based on the performance. MQTT was built as a lightweight protocol for devices with limited resources so it could be used for communication between IoT constrained nodes and fog nodes. AMQP is also a lightweight protocol; however, with additional support for security, reliability, provisioning and interoperability the overhead and message size also increase, thus degrading its performances in nodes with limited processing power. For these reasons, AMQP is more suitable in the more powerful devices, which would position it ideally between fog and cloud nodes. Instead of MQTT in IoT to cloud domain, based on the fact that is considered lightweight and as such adjusted for constrained devices, it is possible to use XMPP protocol. But, at this moment, similar to DDS, it is not as widely accepted in this kind of scenarios.

5.3 Open Issues and Challenges Summarized

Until now, with the IoT advancements, a significant amount of the research efforts has been put into the comprehensive comparisons of the different communication protocols that could potentially be used in the IoT related applications and scenarios. These efforts include comparisons based on the different characteristics of these protocols (underlying transport protocol, interaction model, security, quality of service) as well as based on their individual performance strengths and weaknesses in different IoT related systems.

There is overall a lack of a comparative study of all of the mentioned protocols in a scenario that would cover a broader architectural paradigm that combines IoT, fog and cloud computing

systems, leaving it as a grand challenge for a future research. The next step towards this goal would be to evaluate performance of various protocols surveyed in a useful application scenario. This should include evaluation of the communication in IoT-F2C when each of the individual protocols is used, as well as the scenarios when the two, or more of these protocols are used at the same time in the system. Using multiple communication protocols solutions in fact presents an important new research direction, their interoperability and interaction models. As mentioned in the Section 5.2 there is still an open issue of combining a publish-subscribe and client-server communication between different parts of IoT-F2C. For some of the protocols, such as DDS, which follows completely different architecture from the others, it us unlikely that is even possible to use it in a combination with other protocols, which is an open issue.

While there are many comprehensive studies on different protocol parameters, the most important one that is constantly being adapted for IoT purposes is the security, and it should be privacy as well. Some of these aspects can be addressed by using the TLS and DTLS in combination with surveyed communication protocols, though by doing so they would lose their lightweight properties. The research on how to adapt these two security mechanisms is still ongoing. The area of privacy remains generally under-addressed, including aspects of anonymous communication and censorship applications. While some papers [59, 112, 132] tackle general IoT or specific protocol privacy issues the area of privacy still requires major efforts to advance, both in application layer which was the subject of this layer, and in IoT-F2C solutions.

Finally, a straightforward next future direction need to consider recent developments in newly proposed protocols, most notably HTTP 2.0 and QUIC, that with no doubt will leave their mark towards seamless integration and coordination of IoT, fog and cloud computing systems.

6 CONCLUSIONS AND OUTLOOK

We surveyed application layer protocols designed or adapted for utilization in IoT solutions, focusing on their possible implementation in the IoT-based fog and cloud computing systems. For a system that has to take into account different requirements for IoT, fog computing and cloud computing, it is not likely that any of the surveyed protocols alone will be enough to cover the entire communication in the system, starting from resource constrained devices over to the cloud servers. The survey found that the two most mature choices to consider, which also are favored by developers, to be MQTT and RESTful HTTP. These two protocols are not only the most mature and stable ones, but also include many well documented and successful implementations and online resources. Based on its stability and simple configuration MQTT is the protocol that has proven over time to have excellent performance when used in IoT layer with constrained devices. In the parts of the system where the constrained communication and battery consumption are not an issue, such in some fog and most cloud computing systems, RESTful HTTP is a straightforward choice. CoAP should also be taken into consideration as it is also rapidly evolving as an IoT messaging standard and it is likely that in the near future it will reach a level of stability and maturity similar to MQTT and HTTP. But the standard is evolving for now, which carries short-term interoperability challenges.

One of the major challenges we identified, which is among key factors when choosing appropriate protocols, is that of defining standards to unify varying architectures and interfaces with a goal of achieving a combined management of IoT, cloud and fog. While there are architecture and system proposals that offer IoT-cloud based solution, integrating them with fog computing paradigm is still a novel proposition. Because of the different architectural possibilities, each protocol performs differently in different segments and is thus suited for different types of applications. One of the challenges is also related to the usage of proprietary protocols and their interoperability with increasingly important open protocols. Another issue is the one of implementation stability, as mentioned above, which is a key factor for protocol choice for system developers. Implementing

application protocols other than HTTP requires training for developer's teams. Performance studies have shown that the REST HTTP is however not sufficient in combined IoT, fog and cloud solutions, and this is an open issue for reserach. It remains to be seen whether the future protocol choice will include other messaging protocols, or focus on improving HTTP as it is. Other important features, such as security and privacy, need to be also further analyzed on the overhead they bring, as the current solutions are far from optimal. This creates not only challenges but also exciting opportunities in novel architectures that without doubt will need to combine IoT, fog and cloud computing systems to meet the requirements of future applications.

ACKNOWLEDGMENT

This work has been partially performed in the framework of mF2C project funded by the European Union's H2020 research and innovation programme under grant agreement 730929.

GLOSSARY

AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
DDS	Data Distribution Service
DTLS	Datagram Transport Layer Security
GDS	Global Data Space
HTTP	Hyper Text Transport Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
MQTT	Message Queue Telemetry Transport Protocol
QUIC	Quick UDP Internet Connections
REST	Representational State Transfer
SASL	Simple Authentication and Security Layer
SPDY	Speedy
TLS	Transport Layer Security
XMPP	Extensible Messaging and Presence Protocol

REFERENCES

[1] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes. Operating systems for low-end devices in the internet of things: A survey. *IEEE Internet of Things Journal*, 3(5):720–734, Oct 2016.

[2] Y. Xu, V. Mahendran, W. Guo, and S. Radhakrishnan. Fairness in fog networks: Achieving fair throughput performance in mqtt-based iots. In *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 191–196, Jan 2017.

[3] P. Sethi and Smruti R. Sarangi. Internet of things: Architectures, protocols, and applications. 2017:1–25, 01 2017.

[4] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. On the integration of cloud computing and internet of things. In *Proceedings of the 2014 International Conference on Future Internet of Things and Cloud*, FICLOUD '14, pages 23–30, Washington, DC, USA, 2014. IEEE Computer Society.

[5] C. Huo, T. C. Chien, and P. H. Chou. Middleware for iot-cloud integration across application domains. *IEEE Design Test*, 31(3):21–31, June 2014.

[6] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio. Integration of agent-based and cloud computing for the smart objects-oriented iot. In *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 493–498, May 2014.

[7] T. Pflanzner and A. Kertesz. A survey of iot cloud providers. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 730–735, May 2016.

[8] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[9] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob. Big iot data analytics: Architecture, opportunities, and open research challenges. *IEEE Access*, 5:5247–5261, 2017.

[10] S. M. Babu, A. J. Lakshmi, and B. T. Rao. A study on cloud based internet of things: Cloudiot. In *2015 Global Conference on Communication Technologies (GCCT)*, pages 60–65, April 2015.

[11] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3(1):70–95, Feb 2016.

[12] S. S. Solapure and H. Kenchannavar. Internet of things: A survey related to various recent architectures and platforms available. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2296–2301, Sept 2016.

[13] J. Granjal, E. Monteiro, and J. SÃą Silva. Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys Tutorials*, 17(3):1294–1312, thirdquarter 2015.

[14] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi. Toward better horizontal integration among iot services. *IEEE Communications Magazine*, 53(9):72–79, September 2015.

[15] Andrew (PrismTech) Foster. Messaging Technologies for the Industrial Internet and the Internet of Things, 2014.

[16] N. Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, Oct 2017.

[17] J. Ramirez and C. Pedraza. Performance analysis of communication protocols for internet of things platforms. In *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–7, Aug 2017.

[18] S. N. Swamy, D. Jadhav, and N. Kulkarni. Security threats in the application layer in iot applications. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 477–480, Feb 2017.

[19] M. B. Yassein, M. Q. Shatnawi, and D. Al-zoubi. Application layer protocols for the internet of things: A survey. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–4, Sept 2016.

[20] L. Nastase. Security in the internet of things: A survey on application layer protocols. In *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, pages 659–666, May 2017.

[21] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015.

[22] Pavel Masek, Jiri Hosek, Krystof Zeman, Martin Stusek, Dominik Kovac, Petr Cika, Jan Masek, Sergey Andreev, and Franz Kropfl. Implementation of true iot vision: Survey on enabling protocols and hands-on experience. *International Journal of Distributed Sensor Networks*, 12(4):8160282, 2016.

[23] OpenFog Consortium. OpenFog, 2016.

[24] Edge Computing Consortium (ECC) and Alliance of Industrial Internet (AII). Edge Computing Reference Architecture 2.0. 2017.

[25] mF2C Consortium. mF2C project, 2017.

[26] E Marin-Tordera, Xavi Masip, Jordi Garcia AlmiÃśana, Admela Jukan, Guang-Jie Ren, and Jiafeng Zhu. Do we all really know what a fog node is? current trends towards an open definition. 109, 05 2017.

[27] Y. Jia, E. Bodanese, C. Phillips, J. Bigham, and R. Tao. Improved reliability of large scale publish/subscribe based moms using model checking. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–8, May 2014.

[28] A. Antonić, M. Marjanović, P. Skočir, and I. P. Žarko. Comparison of the cupus middleware and mqtt protocol for smart city services. In *2015 13th International Conference on Telecommunications (ConTEL)*, pages 1–8, July 2015.

[29] Samia Allaoua Chelloug and Mohamed A. El-Zawawy. Middleware for internet of things: Survey and challenges. *Intelligent Automation & Soft Computing*, 0(0):1–9, 2017.

[30] A. Azzará, S. Bocchino, P. Pagano, G. Pellerano, and M. Petracca. Middleware solutions in wsn: The iot oriented approach in the icsi project. In *2013 21st International Conference on Software, Telecommunications and Computer Networks - (SoftCOM 2013)*, pages 1–6, Sept 2013.

[31] M. Veeramanikandan and S. Sankaranarayanan. Publish/subscribe broker based architecture for fog computing. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 1024–1026, Aug 2017.

[32] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*, pages 262–272, 1999.

[33] Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 1:1–1:15, New York, NY, USA, 2009. ACM.

[34] Satvik Patel, Sunil Jardosh, Ashwin Makwana, and Amit Thakkar. Publish/subscribe mechanism for iot: A survey of event matching algorithms and open research challenges. In Nilesh Modi, Pramode Verma, and Bhushan Trivedi, editors, *Proceedings of International Conference on Communication and Networks*, pages 287–294, Singapore, 2017. Springer Singapore.

[35] Kai Sachs, Stefan Appel, Samuel Kounev, and Alejandro Buchmann. Benchmarking publish/subscribe-based messaging systems. In Masatoshi Yoshikawa, Xiaofeng Meng, Takayuki Yumoto, Qiang Ma, Lifeng Sun, and Chiemi Watanabe, editors, *Database Systems for Advanced Applications*, pages 203–214, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[36] A. Kerenen M. Koster and J. Jimenez. Publish-Subscribe Broker for the Constrained Application Protocol. Rfc, RFC Editor, July 2017.

[37] I. Fette and A. Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011.

[38] S. Cook, B. Mathieu, P. Truong, and I. Hamchaoui. Quic: Better for what and for whom? In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2017.

[39] Y. Cui, T. Li, C. Liu, X. Wang, and M. KÃijhlewind. Innovating transport with quic: Design approaches and research challenges. *IEEE Internet Computing*, 21(2):72–76, Mar 2017.

[40] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Http over udp: An experimental investigation of quic. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 609–614, New York, NY, USA, 2015. ACM.

[41] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999.

[42] Edited by Andrew Banks and Rahul Gupta. Mqtt version 3.1.1. Oasis standard, 29 October 2014.

[43] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014.

[44] OASIS. Advanced message queuing protocol (amqp) version 1.0. Oasis standard, 29 October 2012.

[45] Object Management Group (OMG). Data distribution service (dds) version 1.4. pages 1–20, March 2015.

[46] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. RFC 3920, RFC Editor, October 2004.

[47] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2). RFC 7540, RFC Editor, May 2015.

[48] C. Severance. Roy t. fielding: Understanding the rest style. *Computer*, 48(6):7–9, June 2015.

[49] Z. B. Babovic, J. Protic, and V. Milutinovic. Web performance evaluation for internet of things applications. *IEEE Access*, 4:6974–6992, 2016.

[50] W. Shang, Y. Yu, R. E. Droms, and L. Zhang. Challenges in iot networking via tcp/ip architecture. Number 2, page 7, 2016.

[51] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008.

[52] K. Bhargavan, B. Blanchet, and N. Kobeissi. Verified models and reference implementations for the tls 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 483–502, May 2017.

[53] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu. Multiple handshakes security of tls 1.3 candidates. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 486–505, May 2016.

[54] T. Savolainen, N. Javed, and B. Silverajan. Measuring energy consumption for restful interactions in 3gpp iot nodes. In *2014 7th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 1–8, May 2014.

[55] Daniel Stenberg. Http2 explained. *SIGCOMM Comput. Commun. Rev.*, 44(3):120–128, July 2014.

[56] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, B. Silverajan, and B. Raymor. Coap (constrained application protocol) over tcp, tls, and websockets. RFC 8323, RFC Editor, February 2018.

[57] H. V. Nguyen and L. L. Iacono. Rest-ful coap message authentication. In *2015 International Workshop on Secure Internet of Things (SIoT)*, pages 35–43, Sept 2015.
[58] N. Correia, D. Sacramento, and G. SchÃijtz. Dynamic aggregation and scheduling in coap/observe-based wireless sensor networks. *IEEE Internet of Things Journal*, 3(6):923–936, Dec 2016.
[59]
[60] E. Rescorla and N. Modadugu. Datagram transport layer security version 1.2. RFC 6347, RFC Editor, January 2012.
[61] M. Panwar and A. Kumar. Security for iot: An effective dtls with public certificates. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 163–166, March 2015.
[62] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. Lithe: Lightweight secure coap for the internet of things. *IEEE Sensors Journal*, 13(10):3711–3720, Oct 2013.
[63] S. Raza, D. Trabalza, and T. Voigt. 6lowpan compressed dtls for coap. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, pages 287–289, May 2012.
[64] V. Lakkundi and K. Singh. Lightweight dtls implementation in coap-based internet of things. In *20th Annual International Conference on Advanced Computing and Communications (ADCOM)*, pages 7–11, Sept 2014.
[65] Eclipse Mosquitto. Eclipse mosquitto, 2018.
[66] E M Q Documentation and Feng Lee. Emq 2.0 documentation, 2016.
[67] The Apache Software Foundation. Apache ActiveMQ, 2010.
[68] HiveMQ. HiveMQ - Enterprise MQTT Broker, 2015.
[69] Nick Maynard. Mqtt and ibm messagesight : Secure, reliable communications for the next generation of resilient mobile applications, January 2015.
[70] The JoramMQ by ScalAgent. JoramMQ , a distributed broker for the Internet of Things, 2017.
[71] RabbitMQ by Pivotal Software. Rabbit MQTT Broker, 2017.
[72] VerneMQ. VerneMQ Broker, 2017.
[73] N. Tantitharanukul, K. Osathanunkul, K. Hantrakul, P. Pramokchon, and P. Khoenkaw. Mqtt-topics management system for sharing of open data. In *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*, pages 62–65, March 2017.
[74] J. E. Luzuriaga, J. C. Cano, C. Calafate, P. Manzoni, M. Perez, and P. Boronat. Handling mobility in iot applications using the mqtt protocol. In *2015 Internet Technologies and Applications (ITA)*, pages 245–250, Sept 2015.
[75] A. Stanford-Clark and H. Linh Troung. MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2. *Mqtt.Org*, 2013.
[76] K. Govindan and A. P. Azad. End-to-end service assurance in iot mqtt-sn. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 290–296, Jan 2015.
[77] C. Lesjak, D. Hein, M. Hofmann, M. Maritsch, A. Aldrian, P. Priller, T. Ebner, T. Ruprechter, and G. Pregartner. Securing smart maintenance services: Hardware-security and tls for mqtt. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1243–1250, July 2015.
[78] A. Corsaro. *The Data Distribution Service Tutorial*. 05 2014.
[79] G. Farabaugh G. Pardo-Castellote and R. Warren. An introduction to dds and data-centric communications. *Real-Time Innovations*, (August), 2005.
[80] Gustavo B Baptista, Felipe Carvalho, Sergio Colcher, and Markus Endler. A Middleware for Data-centric and Dynamic Distributed Complex Event Processing for IoT Real-time Analytics in the Cloud. 2001.
[81] J. Yang, K. SandstrÃűm, T. Nolte, and M. Behnam. Data distribution service for industrial automation. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, pages 1–8, Sept 2012.
[82] M. Hamilton H. Choi S. Rhee G. Subramanian Y. Dai E. Sin S. Sonck Thiebaut, G. Pardo-Castellote and A. Bose. Real-time publish subscribe (rtps) wire protocol specification. Rfc, RFC Editor, February 2002.
[83] J. 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*, 1(3):176–187, June 2017.
[84] Inc Twin Oaks Computing. What can DDS do for You?, 2011.
[85] S. Pradhan, W. Emfinger, A. Dubey, W. R. Otte, D. Balasubramanian, A. Gokhale, G. Karsai, and A. Coglio. Establishing secure interactions across distributed applications in satellite clusters. In *2014 IEEE International Conference on Space Mission Challenges for Information Technology*, pages 67–74, Sept 2014.
[86] Object Management Group (OMG). OpenDDS Developer's Guide, 2017.
[87] J. L. Fernandes, I. C. Lopes, J. J. P. C. Rodrigues, and S. Ullah. Performance evaluation of restful web services and amqp protocol. In *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 810–815, July 2013.
[88] E. C. M. van der Linden, Jonas Wallgren, and Peter Jonsson. A latency comparison of iot protocols in mes. 2017.

[89] A. Hornsby and R. Walsh. From instant messaging to cloud computing, an xmpp review. In *IEEE International Symposium on Consumer Electronics (ISCE 2010)*, pages 1–6, June 2010.

[90] D. Conzon, T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M. A. Spirito. The virtus middleware: An xmpp based architecture for secure iot communications. In *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, July 2012.

[91] D. Schuster, P. Grubitzsch, D. Renzel, I. Koren, R. Klauck, and M. Kirsche. Global-scale federated access to smart objects using xmpp. In *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pages 185–192, Sept 2014.

[92] X. Che and S. Maag. A passive testing approach for protocols in internet of things. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 678–684, Aug 2013.

[93] A. Hornsby and E. Bail. μxmpp: Lightweight implementation for low power operating system contiki. In *2009 International Conference on Ultra Modern Telecommunications Workshops*, pages 1–5, Oct 2009.

[94] H. Wang, D. Xiong, P. Wang, and Y. Liu. A lightweight xmpp publish/subscribe scheme for resource-constrained iot devices. *IEEE Access*, 5:16393–16405, 2017.

[95] Istabraq M. Al-Joboury and Emad H. Al-Hemiary. Performance analysis of internet of things protocols based fog/cloud over high traffic. *Journal of Fundamental and Applied Sciences*, 10(6S):176–181, 2018.

[96] 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. In *2017 International Conference on Information Networking (ICOIN)*, pages 468–473, Jan 2017.

[97] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan. Performance evaluation of mqtt and coap via a common middleware. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, April 2014.

[98] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali. Comparison of two lightweight protocols for smartphone-based sensing. In *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, pages 1–6, Nov 2013.

[99] S. Mijovic, E. Shehu, and C. Buratti. Comparing application layer protocols for the internet of things via experimentation. In *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–5, Sept 2016.

[100] 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 *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pages 1–6, May 2017.

[101] T Dimčić, S Krčo, and N Gligorić. Coap (constrained application protocol) implementation in m2m environmental monitoring system. pages 229–234, 2012.

[102] M. Saleh, M. A. Abdou, and M. Aboulhassan. Assessing the use of ip network management protocols in smart grids. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–6, Nov 2016.

[103] W. Gao, J. H. Nguyen, W. Yu, C. Lu, D. T. Ku, and W. G. Hatcher. Toward emulation-based performance assessment of constrained application protocol in dynamic networks. *IEEE Internet of Things Journal*, 4(5):1597–1610, Oct 2017.

[104] Y. Chen and T. Kunz. Performance evaluation of iot protocols under a constrained wireless access network. In *2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*, pages 1–7, April 2016.

[105] Laila Daniel, Markku Kojo, and Mikael Latvala. Experimental evaluation of the coap, http and spdy transport services for internet of things. In Giancarlo Fortino, Giuseppe Di Fatta, Wenfeng Li, Sergio Ochoa, Alfredo Cuzzocrea, and Mukaddim Pathan, editors, *Internet and Distributed Computing Systems*, pages 111–123, Cham, 2014. Springer International Publishing.

[106] Diego Londoño and Sandra Céspedes. Performance evaluation of coap and http/2 in web applications. *CEUR Workshop Proceedings*, 1727:25–27, 2016.

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

[108] S. Bandyopadhyay and A. Bhattacharyya. Lightweight internet protocols for web enablement of sensors using constrained gateway devices. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 334–340, Jan 2013.

[109] P. Thota and Y. Kim. Implementation and comparison of m2m protocols for internet of things. In *2016 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)*, pages 43–48, Dec 2016.

[110] W. Colitti, K. Steenhaut, N. De Caro, B. Buta, and V. Dobrota. Evaluation of constrained application protocol for wireless sensor networks. In *2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, pages 1–6, Oct 2011.

[111] 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. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 931–936, Jan 2015.

[112] M. FRUSTACI, P. PACE, G. ALOI, and G. FORTINO. Evaluating critical security issues of the iot world: Present and future challenges. *IEEE Internet of Things Journal*, PP(99):1–1, 2017.

[113] Qi Jing, Athanasios Vasilakos, Jiafu Wan, Jingwei Lu, and Dechao Qiu. Security of the internet of things: Perspectives and challenges. 20:2481–2501, 11 2014.

[114] R. T. Tiburski, L. A. Amaral, E. D. Matos, and F. Hessel. The importance of a standard securit y archit ecture for soa-based iot middleware. *IEEE Communications Magazine*, 53(12):20–26, Dec 2015.

[115] Paul Fremantle and Philip Scott. A security survey of middleware for the internet of things. 01 2015.

[116] D. Dragomir, L. Gheorghe, S. Costea, and A. Radovici. A survey on secure communication protocols for iot systems. In *2016 International Workshop on Secure Internet of Things (SIoT)*, pages 47–62, Sept 2016.

[117] N. J. Al Fardan and K. G. Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, May 2013.

[118] R. T. Tiburski, L. A. Amaral, E. de Matos, D. F. G. de Azevedo, and F. Hessel. Evaluating the use of tls and dtls protocols in iot middleware systems applied to e-health. In *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 480–485, Jan 2017.

[119] Eclipse IoT Working Group, IEEE IoT, and AGILE IoT. IoT Developer Survey. pages 1 – 39, 2016.

[120] Francisco Carpio, Admela Jukan, Ana Isabel Martín Sanchez, Nina Amla, and Nicole Kemper. Beyond production indicators: A novel smart farming application and system for animal welfare. In *Proceedings of the Fourth International Conference on Animal-Computer Interaction*, ACI2017, pages 7:1–7:11, New York, NY, USA, 2017. ACM.

[121] Spring Project. Building a RESTful Web Service.

[122] G. Peralta, M. Iglesias-Urkia, M. Barcelo, R. Gomez, A. Moran, and J. Bilbao. Fog computing based efficient iot scheme for the industry 4.0. In *2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM)*, pages 1–6, May 2017.

[123] P. Desai, A. Sheth, and P. Anantharam. Semantic gateway as a service architecture for iot interoperability. In *2015 IEEE International Conference on Mobile Services*, pages 313–319, June 2015.

[124] C. Lerche, N. Laum, F. Golatowski, D. Timmermann, and C. Niedermeier. Connecting the web with the web of things: lessons learned from implementing a coap-http proxy. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, volume Supplement, pages 1–8, Oct 2012.

[125] A.Rahman T. Fossati A. Castellani, S.Loreto and E. Dijk. Guidelines for HTTP-CoAP Mapping Implementations. Rfc, RFC Editor, March 2015.

[126] F. Van den Abeele, E. Dalipi, I. Moerman, P. Demeester, and J. Hoebeke. Improving user interactions with constrained devices in the web of things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 153–158, Dec 2016.

[127] J. Esquiagola, L. Costa, P. Calcina, and M. Zuffo. Enabling coap into the swarm: A transparent interception coap-http proxy for the internet of things. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, June 2017.

[128] A. B. Sulaeman, F. A. Ekadiyanto, and R. F. Sari. Performance evaluation of http-coap proxy for wireless sensor and actuator networks. In *2016 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pages 68–73, Sept 2016.

[129] N. Le Sommer, L. Touseau, Y. MahÃĺo, M. Auzias, and F. Raimbault. A disruption-tolerant restful support for the web of things. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 17–24, Aug 2016.

[130] A. P. Castellani, T. Fossati, and S. Loreto. Http-coap cross protocol proxy: an implementation viewpoint. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, volume Supplement, pages 1–6, Oct 2012.

[131] M. Buschsieweke and M. GÃijneÅ§. Authentication for the web of things: Secure end-to-end authentication between coap and http. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–5, Oct 2017.

[132] M. Buschsieweke and M. Gunes. Authentication for the web of things - secure end-to-end authentication between coap and http. 2018.