applied sciences

MDPI

Article

A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA

Daniel Silva [1,*], Liliana I. Carvalho [1], José Soares [1] and Rute C. Sofia [2,3]

[1] Cognitive and People-Centric Research Unit (COPELABS), University Lusofona de Humanidades e Tecnologias, 1749-024 Lisbon, Portugal; lilinocencio@gmail.com (L.I.C.); jose94soares@gmail.com (J.S.)
[2] Fortiss—Research Institute of the Free State of Bavaria for Software Intensive Services and Systems, 80805 Munich, Germany; sofia@fortiss.org
[3] Department of Informatics Engineering, University Lusofona de Humanidades e Tecnologias, 1749-024 Lisbon, Portugal
* Correspondence: danielmanilha@hotmail.com

Abstract: IoT data exchange is supported today by different communication protocols and different protocolar frameworks, each of which with its own advantages and disadvantages, and often co-existing in a way that is mandated by vendor policies. Although different protocols are relevant in different domains, there is not a protocol that provides better performance (jitter, latency, energy consumption) across different scenarios. The focus of this work is two-fold. First, to provide a comparison of the different available solutions in terms of protocolar features such as type of transport, type of communication pattern support, security aspects, including Named-data networking as relevant example of an Information-centric networking architecture. Secondly, the work focuses on evaluating three of the most popular protocols used both in Consumer as well as in Industrial IoT environments: MQTT, CoAP, and OPC UA. The experimentation has been carried out first on a local testbed for MQTT, COAP and OPC UA. Then, larger experiments have been carried out for MQTT and CoAP, based on the large-scale FIT-IoT testbed. Results show that CoAP is the protocol that achieves across all scenarios lowest time-to-completion, while OPC UA, albeit exhibiting less variability, resulted in higher time-to-completion in comparison to CoAP or MQTT.

Keywords: Internet of Things; networking protocols; networking architectures; performance evaluation

Citation: Silva, D.; Carvalho, L.I.; Soares, J.; Sofia, R.C. A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA. *Appl. Sci.* **2021**, *11*, 4879. https://doi.org/10.3390/app11114879

Academic Editor: Fabrizio Granelli

Received: 11 May 2021
Accepted: 18 May 2021
Published: 26 May 2021

1. Introduction

Internet of Things (IoT) communication architectures and protocols have been evolving in order to cope with new challenges derived from environments involving a large number of heterogeneous, resource-constrained devices. Examples of such challenges are the support for intensive processing of large amounts of data; data filtering; data mining and classification; supporting high heterogeneity in terms of device hardware and software, as well as types of traffic. Moreover, IoT end-to-end services are supported by the *Transmission Control Protocol/Internet Protocol (TCP/IP)*. TCP/IP allows for interoperability, but faces limitations in IoT scenarios due to the large-scale heterogeneous IoT environments requirements, such as: time sensitive data, power constrains, or the need to support low latency (often, sub-second responses), and low jitter. Therefore, IoT scenarios rely on IP-based messaging protocols, or protocolar frameworks such as OPC UA to meet the required application requirements, by providing support to asynchronous communication mediated via a server, or a broker entity. In other words, in practice, IoT scenarios rely on communication protocols and architectures that follow a broker-based publish/subscriber approach, which creates an abstraction between data sources (*producers*), data receivers (*consumers*).

Albeit these solutions assist, for instance, frequent data polling, issues concerning mobility management, privacy, security, or resource consumption subsist. The root of this

problem derives from the underlying networking semantics of TCP/IP, which follow a host-based reachability approach.

Therefore, in order to further evolve communication protocols in a way that best sustains highly heterogeneous IoT environments, there is the need to better understand current implementation limitations, and performance aspects of the different available solutions.

The rationale for this research concerns the need to better understand the different IoT communication protocols and architectures available; differences in their networking semantics; and differences in performance, in particular in regards to time to completion of requests (latency); packet loss. An hypothesis we advance in this context concerns the fact that there may not be a single "better" performance protocol. A second hypothesis we shall be attempting to answer concerns the potential design impact of the protocol in the performance. This is the core of the motivation for this work. This paper main contributions are as follows:

- the work describes the different protocols and protocolar frameworks, covering main aspects, advantages and disadvantages in IoT scenarios.
- the work provides a thorough comparison of the studied solutions.
- the work provides a performance evaluation of 3 specific solutions: MQTT, CoAP, OPC UA. The performance is evaluated in terms of time-to-completion of requests (latency), and packet loss. The performance evaluation has been carried out in a local testbed (COPELABS IoT Lab) and the large-scale testbed FIT-IoT.

The paper is organised as follows. Section 2 describes work that is related to this paper, explaining our contributions. Section 3 goes over the available protocols and also delves on the novel paradigm of *Information-centric Networking (ICN)*, comparing the protocols in terms of architectural aspects such as: transport, communication pattern. Section 4 describes the experimental environments used for experimentation, including selection of hardware, software, as well as scripts developed. Section 5 gives input on the scenarios created to develop experiments. Section 6 covers the performance evaluation aspects on the local testbed, while Section 7 concerns the performance evaluation on FIT-IoT. The paper concludes in Section 8, where directions for future work are also highlighted.

2. Related Work

Related work has been focused on comparing the performance of CoAP and MQTT in regards to *Round-trip Time (RTT)* [1,2], focusing on the case of a single client directly connected to a server (CoAP) or a broker (MQTT). For the cases deployed, usually MQTT resulted in a lower RTT than CoAP. CoAP, on the other hand, showed positive results in specific applications. Liri et al. evaluate the performance of MQTT, MQTT-SN, CoAP, and QUIC in terms of task completion time (*time-to-completion*) for scenarios with challenging network conditions, such as varying loss, delay and disruption conditions for a simple scenario of a client obtaining data from one sensor [3]. The purpose is to show the points of disruption of the different protocols. The authors suggest that QUIC may be a good alternative to CoAP and suggest specific improvement in terms of adaptive timer for CoAP, derived from the learning in terms of delay disruption. Gündoğan et al. [4] provide an analysis on the performance evaluation of CoAP, MQTT against the *Named-data Networking (NDN)* approach. The experiments were carried out in a single-hop topology on the FIT-IoT testbed. The authors conclude that, in a simple and reliable network, NDN provides more robustness and less node resource consumption, while the IP-based messaging protocols result in less overhead and lower time-to-completion.

Another line of work addresses the performance of lightweight protocols in the context of specific environments, e.g., LTE, machine-to-machine communication over satellites, automation. This is, for instance, the case of Durkop et al. who conducted a performance evaluation of CoAP, MQTT, and OPC -UA in an LTE emulated environment, for RTT [5]. On the specific experimental environment, again considering one hop, OPC UA achieved lower RTT. Bacco et al. provide a comparison between CoAP and MQTT for satellite-based machine-to-machine communication [6]. The performance evaluation considers time-to-

completion and average aggregated throughput at brokers/servers. Results provided show that the performance of the two protocols, in this specific case, is dependent upon several aspects that need to be engineered. Profanter et al. provide a comparison of the performance of DDS, OPC UA, MQTT, and sockets in UNIX/ROS (TCPROS) in the context of automation environments [7], where a client machine is connected via a switch to a server machine. Their purpose is to evaluate RTT simulating automation environment conditions, e.g., idle CPU, overloaded CPU. They measured RTT under these conditions and conclude that for this specific environment DDS and OPC UA attain better results than MQTT or the UNIX/ROS socket approach, in particular, if the message size is small. More recently, Proos et al. compared Protofub and Flatbuffers approach against CoAP, AMQP, and MQTT [8]. The work has been evaluated in the context of vehicle-to-Cloud communication having as application digital twins.

Summarising, there is several related literature focusing on the performance of one or several approaches. The experiments are usually focused on specific, relevant scenarios and even domains. Still, the performance evaluation of the different protocols needs to be further pursued, given that there is not a clear direction on a set of protocols that performs (overall) better.

In comparison to related literature, our work follows the line focused on the performance evaluation of IoT communication protocols, contributing first with an analysis on the different architectural features, and then providing a performance evaluation of two of the most relevant performance indicators for IoT environments, namely latency (time-to-completion) and packet loss, varying several conditions, such as message size and message frequency.

3. An Overview on IoT Communication Protocols and Frameworks

This section describes in a concise way several available IoT solutions that are today applied both to Industrial IoT and Consumer IoT, namely AMQP, MQTT, CoAP, OPC UA, QUIC, DDS. The section provides also information on ICN paradigms for IoT. It then provides a comparison of protocol features in Section 3.8, debating differences in Section 3.9.

3.1. AMQP

The *Advanced Message Queueing Protocol (AMQP)* [9–11] (ISO 19464) is an open standard, message-based protocol that assists cross-platform application communication to be easily built via mediator entities, *brokers*, and queues. This protocol is considered to be the "Internet Protocol for Business Messaging" (https://www.amqp.org/ (accessed on 15 May 2021)) and is widely used by entities such as JP Morgan, which relies on AMQP to process over 1 billion messages a day, or the Deutsche Börse. Moreover, *NASA* used it for the Nebula Cloud Computing Platform [12].

AMQP provides a good basis for the integration of software derived from multiple vendors given that it supports common interaction patterns: one way, request/response, publish/subscribe, transactions, and store-and-forward. It does this with flow-control, multiplexing, security, recovery and a portable data representation that enables message filtering. Its messaging strategy supports both point-to-point and hub-and-spoke (broker-based) communications, and it is based on binary messaging. With AMQP, containers communicate via multiplexed sessions that depend on a reliable byte stream with in-order delivery, e.g., TCP.

It is, therefore, a unidirectional binary IP-based messaging protocol, which allows asynchronous communication between publishers and subscribers to occur with some reliability. To do that, AMQP defines the following elements:

- Exchange, entity which receives messages and applies message-based routing.
- Binding, entity in charge of defining rules to bind exchange to queues.

- Queue, which supports the asynchronous communication between publishers and subscribers, by storing messages sent by Exchanges, and routed via the binding rules to subscribers.

An AMQP network consists of a broker and applications connected in a star. Brokers can be added for redundancy, and sometimes connected in federations. AMQP adopts the HTTP view of the Internet as a set of large hubs serving many users. In an AMQP network, applications address the broker, which is the only "stable" piece of the network. This means the only IP addresses or domain name known to all involved applications concerns the brokers. This implies that all publishers and consumers need to know beforehand the URL of brokers. A first advantage of AQMP is the support of assynchronous communication. A second concerns the integrated QoS features, which provide guaranteed message delivery. interoperability and the capability to provide secure connections. However, AMQP transmissions require more bandwidth than its counterpart MQTT, and it does not support resource discovery.

3.2. MQTT

The *Message Queuing Telemetry Transport (MQTT)* protocol [13], developed by *IBM* and currently an *OASIS* standard is, in comparison to AMQP [9,10] (ISO 19464), a lighter message-oriented protocol. It has been designed to support remote monitoring and as such it provides low latency, assured messaging over fragile networks, and efficient distribution of data to one or many receivers. Due to this, it became popular in *Machine-to-Machine (M2M)* scenarios. It is TCP-based and asynchronous, and can integrate a publish-subscribe communication model.

As with AMQP, MQTT relies on the notion of a broker which receives subscription requests from clients on specific topics and which receives messages from publishers and forwards them, based on the client's subscribe/publish policies on topics. MQTT brings in the advantage of having the possibility to consider multiple *Quality of Service (QoS)* levels.

Specifically focusing on Industrial IoT scenarios, there is a specific specification of MQTT, MQTT Sparkplug (https://github.com/eclipse/tahu (accessed on 15 May 2021)), that defines a standardad MQTT namespace optimized for industrial applications. The specification considers an MQTT payload definition that is extensible, but that has been optimized for SCADA implementations. The specification also provides specific architectural design to address redundancy and scale. It is important also to highlight that MQTT can also in a "request/response" pattern, supporting one-to-one messaging that provides specific information based on parameters provided. The MQTT request-response pattern is not similar to a synchronous client/server approach request/response (such as the pattern of HTTP). With MQTT, a request, or a response, can have more than one (or none) subscriber. Moreover, correlation data assists in keeping the relation between request and response stable. The request-response pattern of MQTT is relevant in use-cases where there is the need for acknowledgement functionality.

A first drawback of MQTT in the context of IoT is that it requires TCP support. TCP introduces more reliability but brings in issues in regards to mobility as well as to security. Therefore, MQTT considers SSL/TLS to provide secure communication and data encryption.

A second drawback in comparison to AMQP is that it offers few control options, and real-time communication is considered to be in the order of seconds MQTT has been designed for reliability, and not for speed.

On the other hand, a strong advantage of the MQTT design is scalability. It supports a large number of small, constrained devices, providing a simple way to ensure asynchronous communication of devices.

MQTT has another strong advantage, which is the support of three classes of QoS to messaging: (1) fire-and-forget/unreliable; (2) "at least once" to ensure it is sent a minimum of one time (but might be sent more than one time), and (3) "exactly once".

Finally, its lightweight design and compact binary packet payload are advantages in terms of IoT environments, as shall be further discussed in Section 3.8.

3.3. CoAP

The *Constrained Application Protocol (CoAP)*, defined in the *Internet Engineering Task Force (IETF)* RFC 7252 [14], is a service layer protocol used by resource constrained, low-power sensors and devices connected via lossy networks, especially when there is a high number of sensors and devices within the network. CoAP is one of the most popular IoT communication protocols, in particular in the context of advanced metering and distributed intelligence applications, given that it extends the reach of HTTP to constrained devices. CoAP is being used, for instance, with IEEE802.15.4 low-power radios.

Based on a client/server model, CoAP relies on REST thus increasing its interoperability. Its design has been carefully worked to fit constrained devices in terms of battery, memory, storage. Running over UDP, CoAP clients communicate to servers via 4 messages, GET, PUT, POST and DELETE. Its lightweight design makes it a promising protocol for embedded devices.

In addition to its wide availability, and due to running over UDP, CoAP integrates DTLS (and not SSL/TLS), usually supporting RSA and AES, or ECC and AES. As a service layer protocol, CoAP can also support other application layer protection solutions, such as the Object Security for Constrained RESTful Environments (OSCORE) [15]. Another advantageous design feature of CoAP is resource discovery. CoAP servers provide a list of current resources, with metadata, via a specific file (/.well-known/core, defined in application/link-format media type). Clients can discover which resources are provided and the respective media types. While widely available and highly interoperable, also providing inbuilt support for content negotiation and discovery, CoAP remains a one-to-one protocol based on a client/server model (while in contrast, counterparts such as MQTT provide support to many-to-many communication).

3.4. OPC UA: Open Platform Communications—Unified Architecture

The *Open Platform Communications Unified Architecture (OPC UA)* [16–18] started in 1996 as a concept for automation industry, where the purpose was to have a standard for interacting with control hardware and field-level devices. OPC would shield client applications from the details of the automation equipment, and would provide standardised interfaces to interact with control hardware and field-devices.

The original OPC specification derives from *OLE*, *COM* and *DCOM* technologies, tying it exclusively to the Microsoft platform.

With the extinction of DCOM in 2002 by Microsoft, the OPC community started the definition of a new OPC Data Access standard over *SOAP/XML*. While these new standard addressed portability, the overhead introduced by SOAP/XML was not compatible with several industrial use cases. This led to the beignning of the OPC UA initiative, which delivered the 1.0 version of the standard in 2009. As part of this process, the acronym OPC was changed to be "Open Platform Communication" and the new standard was called OPC UA (OPC Unified Architecture).

Today, OPC UA is a platform-independent standard via which various systems and devices can communicate, by sending messages between clients and servers over various networks. It supports robust, secure communication that assures the identity of clients and servers, and is claimed to resist several types of attacks.

These qualities make OPC UA one of the most promising candidates for unified communication in the context of Industrial IoT, in particular for critical and close environments, such as industrial plants, as it provides in-plant resilient and robust communication.

However, there are a few drawbacks. A major drawback of OPC UA is that it cannot ensure the complete isolation of the plant network when connecting IIoT infrastructures. To access data from a *UA* server, an OPC client outside the plant network needs an open firewall port. The answer currently provided to this issue is to keep OPC UA for the

in-plant communication, and to provide interoperability with other IP-based messaging approaches, for instance OPC UA to MQTT, to support the articulation of communication to the network (Edge-Cloud).

One possible way to increase the relevancy of OPC UA as a possible candidate to unify the communications in IoT, is to consider a push communication model thus allowing the server to make an outbound connection to a client. These are aspects being worked upon by the OPC UA working group which are currently extending the OPC UA client/server model into a publisher/subscriber model, thus allowing the server to take the role of Publisher and to publish data towards an arbitrary number of clients (Subscribers). Two methods are under discussion:

- **Publisher/subscriber over fast, local communication media**. With this method data are sent once (long-lived) and sent to any number of clients over *UDP (secure multicast UDP)*. This model supports 1 to N communication and is relying on UDP directives for *Time Sensitive Networks (TSNs)*.
- **Publisher/subscriber for message exchange in global networks (e.g., Cloud)**. With this method data are sent between OPC UA applications residing in different networks or where data shall be published to clients that reside in the cloud, as well as to relays; brokers; hubs. This model supports N to M communication based on AMQP.

3.5. DDS

The *Object Management Group (OMG) Data Distribution Service (DDS)* [19] is a middleware connectivity framework and API standard for data-centric connectivity, that targets direct communication between devices. Its purpose is to distribute data to other devices while interfacing with an IT infrastructure.

DDS provides low-latency data connectivity, extreme reliability, and a scalable architecture, that fits well business and mission-critical IoT require. DDS was first introduced to overcome limitations of client/server architectures such as CORBA, COM+/DCOM. A main difference of DDS in comparison to AMQP and to MQTT is decentralization. DDS supports 1 to N communication. A second relevant difference is that DDS provides direct communication support, without a broker. A third difference is the support for automatic discovery of devices.

With DDS, "real-time" is in the order of milliseconds. DDS offers powerful ways to filter and select exactly which data goes where, and "where" can be thousands of simultaneous destinations. Some devices are small, so there are lightweight versions of DDS that run in constrained environments.

To allow such control, instead of relying on a hub-and-spoke broker model such as AMQP and MQTT, or on a client/server model such as CoAP and OPC UA, DDS implements direct device-to-device "bus" communication with a relational data model, coined *DataBus*. This data-centric messaging bus model connects publishers and subscribers directly. Detailed QoS policies controls the information flow, and the application matches publishers and subscribers based on topics and on QoS.

3.6. QUIC

The *Quick UDP Internet Connections (QUIC)* protocol [20] is a flexible multiplexed and secure general-purpose transport protocol that supports multiplexed streams. Based on UDP, it has been developed to provide security protection equivalent to TLS/SSL along with reduced connection and transport latency, and bandwidth estimation in each direction to avoid congestion. Hence, QUIC relies on UDP, and integrates encryption. As with STCP, QUIC developed support for multiple sub-streams instead of using multiple transport connections.

Originally developed by Google, today QUIC corresponds to the IETF changed version of the Google QUIC (gQUIC) [21]. It is a relevant protocol in the context of the Internet's evolution and in particular of the Web evolution from TCP to UDP.

In the context of IoT, as it is kernel independent and can run across NATs, it is a relevant protocol in comparison to HTTP (request-response) approaches.

3.7. Information-Centric Networking Approaches in IoT

Information-centric Networking (ICN) is emerging as a new stack which is being explored in multiple areas. However, it is in the field of IoT that ICN is gaining ground. There are several architectures being explored from an end-to-end perspective, being the most popular today the *Content centric Networking (CCN)* architecture (2010, PARC and Partners), the *Named Data Networking (NDN)* architecture (2014, coordinated by UCLA), and *Hybrid ICN (hICN)* (2014), by Cisco. In what concerns relevancy for IoT, the most popular approach today is NDN as it supports all IoT basic requirements [22].

CCN has been adopted by the *Internet Research Task Force (IRTF)* working group *Information-centric Networking Research Group (ICNRG)* [23] and gave rise to both NDN and hICN. Both software architectures are therefore quite similar, being the main difference the fact that hICN is focused on IP interoperability, while NDN is focused on the development of clearer networking semantics, that support better the aspects that IP cannot cope by design, such as security, mobility.

3.7.1. Named Data Networking for IoT

NDN brings in several advantages in the context of IoT scenarios, such as its expressive naming space, support for data fragmentation, intrinsic consumer mobility support, name-based routing [22]. In NDN, consumers express interest on data via sending Interest packets, whose status is kept in routers along the paths traversed, and producers reply with Data packets that match received Interest packets. Data packets traverse the path of Interest packets, following a breadcrumb approach.

Figure 1 provides a simple illustration for the NDN IoT operation, where C is a device with an application wanting to obtain the temperature from A and B. C first sends an Interest packet with the name prefix */ndn/temperature*. A and B reply with Data packets with the requested information.

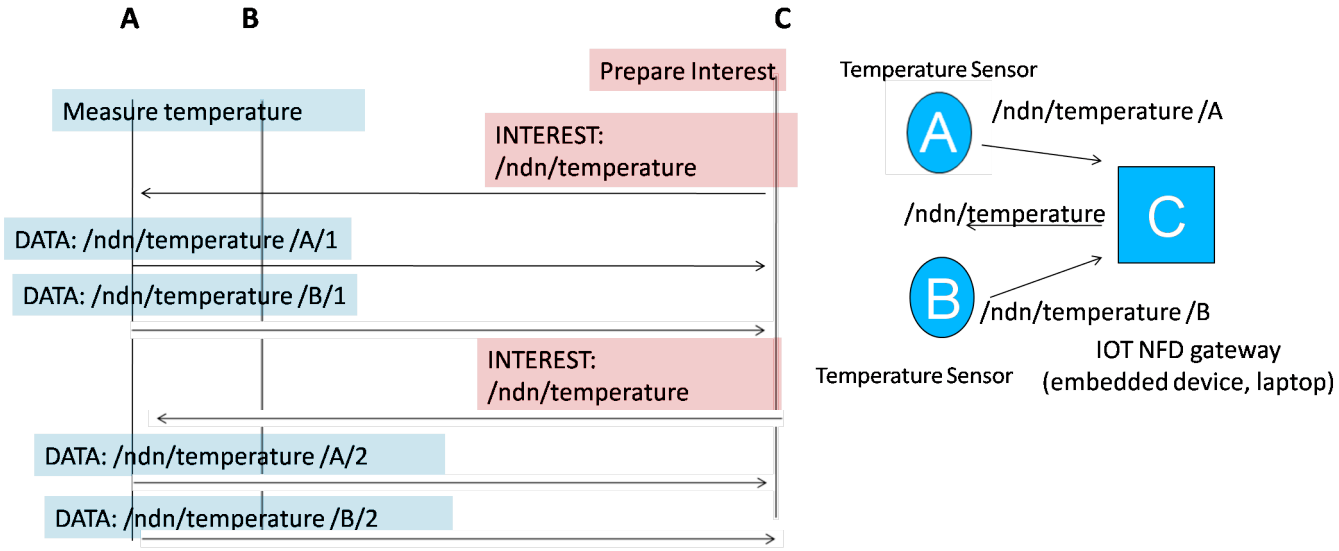


Figure 1. One hop NDN IoT Example, where 2 sensors (**A**), (**B**) publish temperature values, and gateway (**C**) subscribes that information.

NDN follows the ICN *store-and-forward* principle and hence, any node in the network is an NDN router, and holds three different data structures: the *Forwarding Information Base (FIB)*; *Pending Interests Table (PIT)*; *Content Store (CS)*. The FIB holds aggregated name prefixes for data objects matching outgoing Faces (interface abstraction). To fetch content,

a consumer sends an Interest packet to the network containing the name of the required content. When an NDN node receives an Interest message, it first queries matching data in its local CS. If the data are locally available, a matching Data packet is sent back to the consumer through the same Face. Otherwise, the node updates its PIT table with the Interest packet's name prefix, associated to the incoming Face.

If there is no match in the PIT, then the node forwards the Interest packet further over the recorded outgoing interface(s) in the FIB. When the Interest packet reaches a potential data provider or a node with a matching Data packet in its CS, a Data packet is generated and sent back via the path bread-crumbed by the respective Interest packet(s). During the forwarding process, each node replicates the Data packet to all recorded incoming interfaces in the matching PIT entry and keep a copy in the local CS, and then deletes the related PIT record. Thus, NDN traffic is self-regulated, and in each link, for the same object, there is at most one Data and one Interest packet.

The operation of NDN is therefore based on a pull-model, where consumers first express interest about a specific object. Nevertheless, NDN supports a second model, push-based model, derived from applications where data can be directly pushed to multiple consumers, without having these specifically expressing an interest before. As NDN is a network layer solution, push-based models can be implemented by applications in a variety of ways [24].

Furthermore, in large-scale scenarios NDN supports Interest packet aggregation within the PIT structure (aggregation of multiple Interest requests onto a single aggregate request). In-network caching allows consumers to retrieve cached content from intermediate routers, and not necessarily from producers. The different forwarding strategies (e.g., anycast) allow NDN to take into consideration availability and restrictions of devices.

3.7.2. Hybrid ICN

Regular IP devices become hICN-enabled via the installation of hICN middleware. In hICN, the forwarding follows ICN strategies, while the data transmission follows regular IP traffic. Consumers generate the usual ICN Interest packets, which are named in accordance to IP addresses, and are then regularly forwarded. Therefore, in regular IP routers the packet is treated as usual. When it reaches an hICN router the Interest packet is processed accordingly with ICN policies. The PIT entry holds the respective source IP address and the respective interface from which the respective interest packet has been received (*Face*). hICN supports request aggregation following the ICN model. In the case that the content is not found in the router's CS, the Interest packet is then forwarded until it reaches a producer. The producer then sends back a packet that has the same name in the IP source address field and the previous hICN router name (encoded as IP address) in IP destination address field. Requests for the same data initiated by the second consumer will terminate at the first hICN junction point where they are answered either from the local cache (asynchronous multicast) or from the connected data source.

3.8. Comparison of IoT Communication Approaches

Table 1 provides a comparison of the previously debated IoT communication solutions, for the most relevant protocol design aspects. *Transport* (row 1) refers to the supported transport protocols. *Messaging strategy* (row 2) concerns the following messaging patterns: *RR*; *Broker (Publish-subscriber broker model)*; *PS*. Communication model (row 3) corresponds to either client/server *CS* or decentralized *Dec. Security* (row 4) describes the method or protocol that provisions security for each approach, while *binary payload support* (row 5) concerns the transmission of data based on a binary format. *QoS* (row 6) is dedicated to QoS aspects. *data persistence* (row 7) concerns the support of the different approaches for a relevant aspect in IoT (transitive data), while *Data Discovery* (row 8) concerns the methods to discover data sources. *Applicability* (row 9) intends to explain onto which areas of an end-to-end IoT environment the protocols are being applied to, considering the following areas: *Device-to-Edge (D2E)*; *Device-to-device (D2D)*; *Edge to Cloud (E2C)*. *Complexity* (row

10) provides a glimpse at the potential implementation complexity, based upon aspects such as: message header size; implementation size. The next sections go over the different features in detail.

Table 1. Comparison of IoT communication protocols and architectures.

Aspects	HTTP	CoAP	QUIC	AMQP	MQTT	DDS	OPC UA	NDN
Messaging pattern	RR	RR	RR	Broker	RR [1], Broker	RR	RR, PS [2]	PS
Communication Model	CS	CS	CS	CS	CS	Dec	CS	Dec
Security	DTLS, OSCORE, etc.	DTLS, OSCORE, etc.	own encryption	TLS/SSL	TLS/SSL	Specific	Specific	By design
Binary Payload Support	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
QoS	Best Effort	Confirmable, non-confirmable	Congestion control	High availability, broker redundancy	Best Effort, Guaranteed Message Arrival	Parameters on durability, lifespan, destination order, etc.	Best Effort	Forwarding strategy
Data persistence	No	No	Yes	Yes	No	Yes	No	Yes
Data discovery	Manual	Manual and registration	Manual	Manual	Manual	Automatic	No	Automatic
Applicability	D2E, E2E, E2C	D2E	D2E, E2C	D2E	D2E	D2D, D2E, E2E	D2E	D2E, D2C, D2D
Complexity	High	Low	Medium	Low	Low	Low	Low	Low

[1] As described in Section 3.2, MQTT can support a RR pattern supporting one-to-one messaging, for instance, for specific critical scenarios where an acknowledgement is required. [2] under development.

3.8.1. Transport

UDP traffic is supported by all of the approaches, the exception being MQTT, as this protocol needs to have an active connection to allow consumers to be notified of changes. There is, however, an extension of *MQTT for sensor networks (MQTT-SN)* [25] which can support UDP. The trade-off of relying on this extension is a lower control in terms of QoS support.

3.8.2. Messaging Pattern

In terms of the messaging pattern, in the request/response (RR) approach followed by HTTP and CoAP, publishers send data to a server, while consumers request that data from the server. While in the publish/subscribe approach relied upon by, for instance, AMQP or MQTT, publishers send their data to a mediating entity, the broker. The broker distributes that data to consumers that have subscribed a priori the data, via the notion of "topic".

The publish/subscribe strategy is highly relevant from an IoT perspective, given that it provides a way to support asynchronous data exchange and also given that it abstracts, up to some point, the need to handle identifiers of devices.

Publish/subscribe is being adopted by the most relevant protocols, such as OPC UA. DDS mimics a publish/subscribe approach by relying on IP multicast. As for NDN, it integrates a publish/subscriber receiver-driven and pull-based approach (per packet): consumers first express interest, and then the data are sent by producers, based on the expressions of interest. Therefore, NDN is the only approach that integrates a Publish/subscribe approach truly focused on content and not on the entity (host) that generates such content.

3.8.3. Communication Model

The different messaging strategies followed by the approaches require the support of a centralized, client-server model.

From a communication model perspective, the broker-based publish/subscribe is also based on a client/server communication approach.

Therefore, from the analysed protocols, there are only two that integrate a notion of decentralised data exchange, which is a relevant architectural aspect for heterogeneous and dynamic IoT environments: DDS, and NDN. DDS implements a Bus model and requires registration of entities before communication can be held. While NDN relies on the publish/subscribe receiver-driven model, and hence, achieves a higher degree of decentralisation. A main advantage of the NDN publish/subscribe approach is that it can support more control: a broker cannot control aspects such as packet loss or polling, while with NDN forwarding strategies it is possible to adapt the rate of Interest packets sent and as a consequence, improve *Quality of Experience (QoE)*.

3.8.4. Security

Given the urgent requirement for security in IoT environments, the most recent approaches already consider specific security mechanisms. QUIC, for instance, integrates its own encryption mechanism. DDS provides a specification for data security. The most complete approaches in regards to security are OPC UA and NDN. OPC UA integrates a specific security framework covering user security based on X.509; session security based on *AAA*; transport security, based on the encryption of messages. NDN integrates security by design. Not only does this architecture support user, data and channel security; it also secures the binding from name to content, which is relevant, for instance, in situations such as playback attacks.

3.8.5. Binary Payload Support

The support for data transmission of binary payloads is particularly relevant for Industrial IoT scenarios, given the distances to be covered in remote regions, or when addressing large-scale grids (e.g., water pipes). Payloads need to be as compact as possible to minimise battery consumption, and most available protocols support a binary format. All approaches (with the exception of HTTP) already support binary payloads.

3.8.6. QoS

As with security, QoS is a major issue in IoT environments. Most solutions available contemplate a set of classes for guarantees, e.g., guaranteed message delivery. CoAP provides support for confirmable/non-confirmable QoS. AMQP can support high availability based on queueing, and broker redundancy. MQTT integrates 3 levels of QoS: Best Effort, guaranteed message arrival (once), and guaranteed message arrival (more than once). DDS has a finer-granularity QoS, being able to provide parameters on data, even at the granularity of one topic. Examples of QoS parameters supported are: durability; presentation; lifespan, destination order, among others. OPC UA supports Best Effort only as it was designed, having in mind a specific use-case (in-plant communication). NDN has specific forwarding strategies, such as the *QoS-Forwarding Strategy (QoS-FS)* [26]. Furthermore, each NDN node can adjust data transmission based on required QoS and hence, rely on different forwarding strategies over a path between producer and consumer.

3.8.7. Data Persistence

Data persistence is essential to support infrastructure resilience. Message queuing results in lower power usage, and allows for the possibility to send minimized data packets, and efficiently distribute information to one or many subscriber. Still, most protocols analysed lack support for data persistence.

AMQP relies on queuing, while NDN integrates into its design in-networking caching, aspect which becomes relevant in the context of guaranteeing low latency interactions in comparison to cloud computing [27].

3.8.8. Data Discovery

IoT infrastructures are large distributed systems, where each application needs to have a way to find the information it needs; ensuring adequate communication; connect sources and destinations. For broker-based approaches such as AMQP and MQTT, the broker provides an abstraction layer and therefore, applications just need to know the identifier of brokers to rely upon. This reduces the need to register every node. Still, nodes need to know broker identifiers. OPC UA supports discovery of servers (via multicast DNS), which assists in reducing costs associated with configuration. While NDN supports direct data searches, via named-based routing.

3.8.9. Applicability Area

The different approaches have been developed considering different areas of application, from an end-to-end communication perspective. For instance, CoAP has been devised to support HTTP in communicating to resource constrained nodes and hence, it is usually applied in device to edge communication. From the studied approaches, DDS and NDN are the single proposals that contemplate also D2D communication support. Then, from a perspective of edge to edge communication, HTTP, DDS, NDN are the approaches that can, by design, easily be used to support edge to edge communication. Approaches such as MQTT, CoAP or even OPC UA can also be used to support edge to edge communication. However, this would imply integrating both a client and server (or broker) entity in each Edge node.

3.8.10. Complexity

Today the available approaches take into consideration aspects such as small header size, lightweight implementation, and the possibility to run in embedded devices. CoAP, for instance, has an average header size of 4 bytes; requires 10 KiB of RAM, and 100 KiB of code space [14]. AMQP has a very short header size (2 bytes) and, similarly to MQTT, it holds a lightweight implementation that can run in devices with less than 64 Kb of RAM. DDS holds a lightweight implementation (CoreDX DSS [28]) which can run in Android devices. The MicroDDS implementation of DDS, which provides publishing support only, runs on 8-bit microcontrollers with 2 KB of SRAM, and 32 KB of ROM. DDS RTI Connext Micro implementation runs on 16-bit ARM. OPC UA servers can run in devices with just 15 KB of RAM and 10 KB of ROM. NDN for RIoT has been implemented in a very light way when compared to 6LowPAN [29].

3.9. Performance Aspects

Recent studies addressed different performance aspects of NDN against IP-based approaches, to understand up to which point can NDN sustain large-scale IoT infrastructures. Gündoğan et al. have analyzed the performance of NDN between sensors and gateway/broker against CoAP and MQTT, evaluating different scenarios in the context of the FIT-IoT testbed [4]. For NDN, the code of CCN-lite has been considered. Experiments run considered 1-hop as well as multi-hop topologies, involving 70 and 50 nodes, respectively. Albeit this is an initial study, it shows that the NDN publish/subscribe model attains a better performance in the verge of topological variability lower packet loss. The results also show that the rate of Interest packets is an issue which delays content delivery larger time-to-completion occurs, as devices are constrained and PITs have limited sizes. This is an issue for unscheduled publishing scenarios. Nevertheless, packet loss is still reduced in comparison to the IP-based approaches evaluated MQTT and CoAP.

The performance of the available protocols is an aspect that requires further investment of research. There are no clear *Key Performance Indicators* for the protocols analysed.

Therefore, in addition to an overall analysis on protocol features for available IoT communication protocols, this works proceeds with a performance evaluation for 3 of the most relevant and flexible protocols available for both consumer and industrial IoT scenarios: MQTT, CoAP, OPC UA.

4. Experimental Environment

This section describes the experimental environments that have been set to evaluate the performance of MQTT, CoAP, OPC UA.

A first experimental environment used is a local, constrained environment illustrated in Figure 2 (http://copelabs.ulusofona.pt/index.php/research/projects/324-iotlab (accessed on 15 May 2021)). The experiments run in this testbed have as aim to evaluate the performance of the different protocols in terms of *packet loss* and *time-to-completion* of requests.

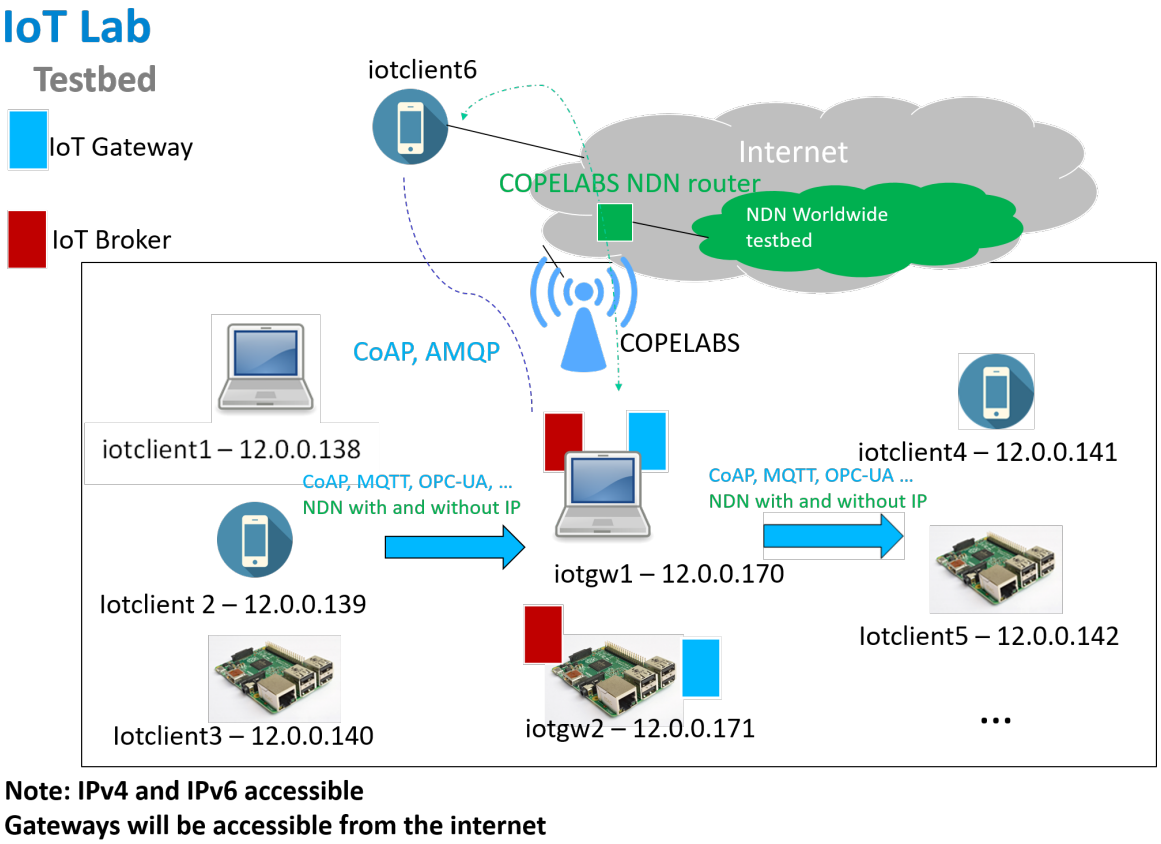


Figure 2. The IoT Testbed at COPELABS.

A second experimental environment used has been the large-scale FIT-IoT testbed [30] (https://www.iot-lab.info/ (accessed on 15 May 2021)).

The protocols evaluated on FIT-IoT were MQTT and CoAP, as the support for OPC UA in FIT-IoT was not, at the time when we have run the experiments, out-of-the-box. We have tried to install the required libraries to run OPC UA. However, the quota provided to users was not sufficient to enable the installation of an open version of OPC UA.

For both experiments and regardless of the protocol *time-to-completion* means the time spent sending the message through the sender and receiving the message by the receiver, considering the time spent by the broker/server. However, the definition of *packet loss* has differences due to the different characteristics of the protocols. For MQTT, which has the characteristic of forming queues for the delivery of messages at the Broker, the packet loss occurs when a message packet sent is no longer received. For CoAP and OPC UA that relies on REST to increasing its interoperability, the *packet loss* is the failure of a GET or PUT command sent to the server.

4.1. IoT Testbed

The COPELABS IoT Testbed [31] comprises, among others, 20 Android smartphones; 2 dedicated machines to support IoT gateways and brokers; 6 Raspberry Pi devices, some of which are attached to temperature/luminosity sensors.

Figure 2 provides a simplified illustration of the testbed, where field devices are connected via wireless to an IoT gateway device. The local testbed is connected both to the Internet and to the experimental NDN worldwide testbed, thus facilitating experimentation of IP-based approaches and of NDN solutions. Further information on the testbed can be found in [31].

The equipment available in the testbed and which has been used in the experiments is summarized in Table 2, which describes aspects such as equipment type and model; CPU, RAM, storage and NIC details.

Table 2. The IoT COPELABS testbed equipment.

Type	Model	CPU	RAM	Storage	Network
IoTclient1	Laptop Toshiba Satellite Pro T130-15C	Genuine Intel® U7300 1.30 GHz Dual Core	3.7 GB	TOSHIBA MK3263GSX 320 GB	Realtek Semiconductor Co., Ltd. RTL8191SEvB Wireless LAN Controller
IoTclient2	Samsung S5 neo	Octa-core 1.6 GHz Cortex-A53	2 GB	16 GB and SD card slot up to 256 GB	GSM/HSPA/LTE
IoTclient3	Raspberry Pi 3 B+ Board	ARMv8 CPU 64-bit Quad Core	1 GB LPDDR2 SDRAM	16 GB Micro SD NOOBS(OS)	2.4 & 5 GHz 802.11b/g/n/ac Wireless LAN
IoTgw1	Laptop Toshiba Satellite Pro T130-15C	Genuine Intel® U7300 1.30 GHz Dual Core	3.7 GB	TOSHIBA MK3263GSX 320 GB	Realtek Semiconductor Co., Ltd. RTL8191SEvB Wireless LAN Controller
IoTgw2	Raspberry Pi 3 B+ Board	ARMv8 CPU 64-bit Quad Core	1 GB LPDDR2 SDRAM	16 GB Micro SD NOOBS(OS)	2.4 & 5 GHz 802.11b/g/n/ac Wireless LAN
Server1	Laptop Toshiba Satellite Pro T130-15C	Genuine Intel® U7300 1.30 GHz Dual Core	3.7 GB	TOSHIBA MK3263GSX 320 GB	Realtek Semiconductor Co., Ltd. RTL8191SEvB Wireless LAN Controller
Server2	Raspberry Pi 3 B+ Board	ARMv8 CPU 64-bit Quad Core	1 GB LPDDR2 SDRAM	16 GB Micro SD NOOBS(OS)	2.4 & 5 GHz 802.11b/g/n/ac Wireless LAN

Moreover, the OS considered in the testbed have been derived from an analysis on available OSs for IoT which is condensed in Table 3. The table shows aspects such as differences in memory usage (RAM and ROM); CPU processing; supported network stack; energy saving options; whether or not multi-threading is supported, and year of development.

Table 3. Operating Systems Comparison.

Features	Contiki	RIoT	FreeRTOS	TinyOS	Raspbian
MCU (bits)	8/16/32	8/16/32	16/32/64	8/16	32/64
RAM(KB)	2	1.5	1	10–40	256
ROM(KB)	60	5	4	15–40	16
Network Stack	uIP (IPv4 and IPv6) and RIME	6LowPAN, GNRC stack, CCN-lite	FreeRTOS + TCP	6LowPAN	6LowPAN, IPv4/v6 CCN-lite
Energy Saving Options	No integration, allows implementation, Scheduler	High energy-efficiency, Scheduler and sleep nodes	Tick-less option	CPU power management, Power management interfaces, HW/SW transparency	Scheduler, Energy optimization
Real-time OS	No	Yes	Yes	No	No
Multi-threading	Yes	Yes	Yes	Yes	Yes
Year	2002	2012	2003	2000	2012

The different operating systems are being adapted to support resource-constrained devices. For this case, Contiki, RIoT and TinyOS provide the best support, based on different ports. In terms of memory consumption (RAM and ROM), Contiki, RIoT, FreeRTOS, and OpenWSN can work in extremely limited devices.

In regards to the supported network stacks, Contiki comes with the uIP TCP/IP stack, as well as with Rime, a set of lightweight networking protocols designed for low-power wireless networks (http://contiki.sourceforge.net/docs/2.6/a01793.html (accessed on 15 May 2021)).

RIoT supports different network stacks, such as 6LowPAN and the *Generic Network Stack (GNRC)*. More importantly, RIoT supports an ICN stack, CCN-lite thus providing good support to explore new networking paradigms. FreeRTOS integrates a specific network stack, FreeRTOS+TCP which is scalable, open source and thread-safe. TinyOS provides support for 6LowPAN/RPL IPv6, while Raspbian provides support for 6LowPAN, IPv4/v6, and CCN-lite. In terms of energy saving options, RIoT and Raspbian already integrate options to support better energetic efficiency. As also shown, only RIoT provides support for real-time applications. Multi-threading is supported in all OSs. Out of the several OSes, based on established requirements and also on the features of each OS, the choice of OS went for Raspbian, and RIoT.

4.2. Gateway Software

The installed gateway software considers the following requirements:

- To be open-source.
- To support the most popular protocols, and at least MQTT/AMQP, CoAP, OPC UA.
- To be modular and allow for the development of extensions.
- To be runnable in constrained devices.

Based on these requirements, two different open-source solutions have been analyzed, Open IoT (http://www.openiot.eu/ (accessed on 15 May 2021)), and ThingsBoard (https://thingsboard.io/ (accessed on 15 May 2021)), as described next.

OpenIoT [32] is an open-source Service-as-a-Platform middleware, which supports the communication between heterogeneous sensors and applications via the cloud. Released in 2012 (ICT-2011.1.3, Internet of Connected Objects https://github.com/OpenIotOrg/

openiot (accessed on 15 May 2021)), OpenIoT stems from an industry-academia effort aiming at developing a platform for connecting physical and virtual sensors to the Cloud. OpenIoT interoperability is derived from semantic sensor integration.

OpenIoT provides two packages: OpenIoT and the OpenIoT-VDK that is a ready-to-use version for academic and training purposes. Its feature integrate:

- Middleware for sensors and sensor networks,
- Ontologies, semantic models and annotations for representing internet-connected objects, along with semantic open-linked data techniques
- Cloud/Utility computing, including utility-based security and privacy schemes.

ThingsBoard is an on-premise and/or cloud platform that helps developers and researchers to test, manage, or interact in IoT environments. It provides a Community edition as well as a Professional edition.

Among other aspects, ThingsBoard provides the following functionality:

- Entities and Relations ability to model world objects (devices and assets) and their relations.
- Telemetry API for collection of time-series data.
- Creation/Management of custom key-value attributes.
- Rule Engine, to do data processing and actions on incoming telemetry and events.
- RPC API and widgets to push commands from apps and dashboards to devices and vice versa.
- Data visualisation DashBoards.
- API limits example: limiting number of requests from a host.
- Audit log track of user activity and API calls usage.

ThingsBoard (TB) operates in *Standalone* or in *Cluster* mode. In Cluster mode, TB nodes can perform automatic discovery of new TB servers (nodes). All TB nodes form a cluster, and all of those TB nodes are identical. On this operation mode there is not a coordinator node, which prevents single points of failure.

Based on the different features and requirements, the experiments described in this paper consider the IoT gateway ThingsBoard.

4.3. Broker Software

For the brokers, both RabbitMQ and Mosquitto have been analysed. The requirements for the selection of the broker software were:

- To run in embedded devices.
- To have a user-friendly configuration.
- If possible, to provide support for several protocols.

RabbitMQ [33] is an open-source message broker that supports many communication protocols such as AMQP, STOMP, MQTT, HTTP, WebSockets. It provides good interoperability, being easily extensible, and a simple user interface.

Mosquitto [34] is an MQTT lightweight broker. It is also quite easy to install and use. In contrast to RabbitMQ, MQTT is a broker for MQTT only.

Although RabbitMQ provides support for MQTT, on our tests the experiments suffered issues when running MQTT with QoS level 2, which leads us to believe that RabbitMQ may have some open issues in the MQTT implementation. Therefore, for MQTT experimentation, we have considered also Mosquitto.

4.4. FIT-IoT Environment

The FIT-IoT testbed is an outcome of the IoT-LAB project (https://www.iot-lab.info/ (accessed on 15 May 2021)), and provides support for large-scale remote testing [30]. FIT-IoT is used worldwide for IoT experimentation in different research areas (https://www.iot-lab.info/publications/ (accessed on 15 May 2021)).

The FIT-IoT testbed comprises six physical sites across France and access to 1786 wireless sensors nodes. For the experimental implementation and evaluation, this paper relies on the Saclay Île-de-France FIT-IoT site, which had 264 nodes available in 2019.

A global networking backbone provides power and connectivity to all FIT-IoT nodes and guarantees the out of band signalling network needed for power and command purposes and monitoring feedback. A FIT-IoT representation to a client site is illustrated in Figure 3.

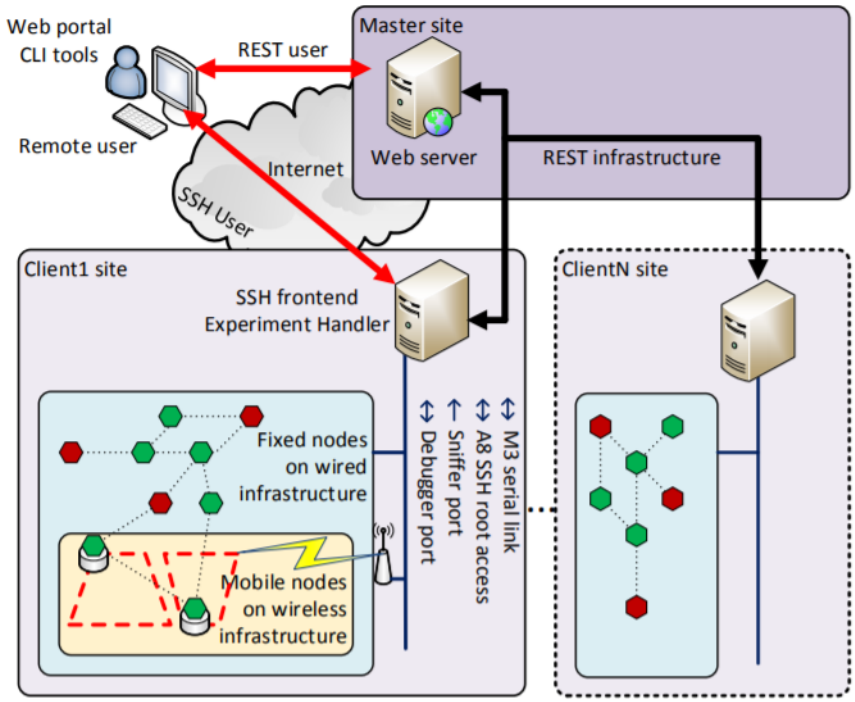


Figure 3. FIT-IoT Infrastructure [30].

As illustrated, the FIT-IoT infrastructure consists of a set of testbed nodes, tied within a global networking backbone that provides power, connectivity, inband and out-of-band signal network capacity for command and monitoring, various servers, and disk space. Each FIT-IoT node consists of three main components: Open Node, Gateway, and Control Node [35].

FIT-IoT supports also the integration of embedded software, ranging from direct access to node hardware, to operating system (OS). Some operating systems such as FreeRTOS, Contiki, TinyOS, OpenWSN and RIoT, may be run on open nodes, depending on software maturity and node capacity. Figure 4 illustrates the software components of a node architecture in Fit-IoT.

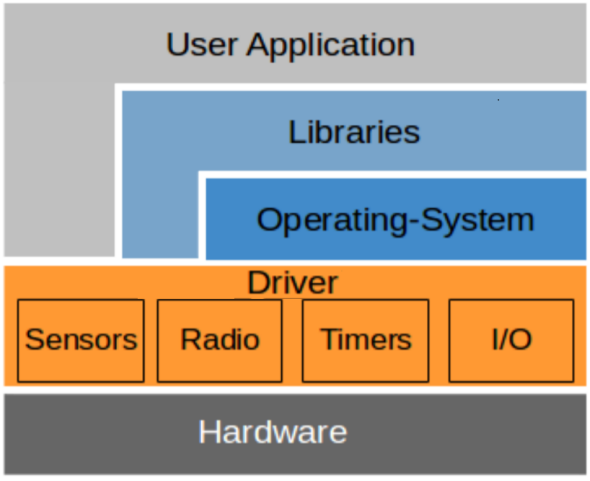


Figure 4. FIT-IoT node architecture.

4.5. Implementation Aspects

This section covers implementation aspects, namely scripts and programs developed to support the implementation of publishers and subscribers, integrating the different entities and protocols described in Section 2. All of the developed scripts and programs are available via GitLab (https://gitlab.com/iotlab_copelabs/ (accessed on 15 May 2021)).

4.5.1. MQTT

The MQTT experimentation environment has been developed in Python with the *paho* library, and the Mosquitto broker. The scripts created for MQTT (https://gitlab.com/iotlab_copelabs/mqtt (accessed on 15 May 2021)) are:

- Subscriber: mqttsubs.py
- Publisher: mqttgen.py
- Report Generator: monitor.py

The Subscriber script (*mqttsubs.py*) emulates a subscribre application which applies, in Mosquitto, to a topic *"sensors"*.

The script assumes a timeout of 5 seconds for message reception. If the timeout occurs, the script stops and generates a log file named *"receiver.txt"* and another file named by default *"ttc_file.txt"*, containing the timestamps of each message in the format: *"t1-t2"*, where *t1* corresponds to the timestamp registered by the publisher at the instant when the message is sent, and *t2* corresponds to the timestamp registered by a subscriber, once a message is received. These timestamps are then used in the experiments to compute time-to-completion.

The Publisher script (*mqttgen.py*) generates and sends a specific number of messages to the broker to the topic *"sensors"*, each separated by a specific interarrival time interval. These attributes are specified in the code. By default, the message size has been set to 7 KB. Any other number can be provided as argument to the script. The payload of each of these messages contains the timestamp taken at the instant the messages have been sent, so that once messages are received, the subscribers can rely on such timestamp to locally compute the time it took for completion. Once all messages are sent, the publisher script generates a log file named *"sender.txt"*, containing only the number of messages sent.

Both publisher and subscriber scripts have been configured to connect to the same broker identified in our testbed by a private IP: "12.0.0.170".

A third script has been created, to provide a report based on the publisher and subscriber logs. This report consists of, per presented order: number of messages sent by the publisher; number of messages received by the subscriber; *time to completion (TTC)*; packet loss.

4.5.2. CoAP

For the CoAP testing we have opted to consider the Eclipse Californium library (https://www.eclipse.org/californium/ (accessed on 15 May 2021)), where we have developed a Java program to push and obtain requests.

Moreover, the computation of packet loss took into consideration CoAP ACKs and not data packets. Assuming, for instance, that a client sends five messages to a CoAP server, and then the respective receiver node receives only 3 in CoAP this does not necessarily imply packet loss. It may simply mean that the sending node updated results faster to the server, than the receiver could obtain on the receiver's point of view it sent a get request and obtained an answer (no packets lost) even though the sender already changed the previous value that the receiver did not have. Therefore, for packet loss, we considered the number of acknowledgements both the sender and received nodes.

The scripts generated for the testing of CoAP (https://gitlab.com/iotlab_copelabs/coap (accessed on 15 May 2021)) are:

- Subscriber: mqttsubs.java
- Publisher: mqttgen.java

- Server: Server.java
- Report Generator: Monitor.java

4.5.3. OPC UA

For the OPC UA experimentation, our choice went to Python with the opcua library. The methodology applied is similar to the one already described for MQTT and CoAP. The scripts developed for OPC UA (https://gitlab.com/iotlab_copelabs/opc-ua (accessed on 15 May 2021)) are:

- Subscriber: receiver.py
- Publisher: sender.py
- Server: server.py
- Report Generator: monitor.py

5. Experimental Settings

To define different, heterogeneous scenarios, we have considered four different aspects:

- **Topology**. The execution of the experiments consider 2 different network topologies, where the main change was to have the gateway/broker either installed on an embedded device, or on a laptop.
  - *Topology 1* consists of using a laptop, iotgw1 in Figure 1, as a server/broker to control communication between clients IoTclient1 (a laptop) and Iotclient3 (a Raspberry Pi) on a Wi-Fi network (rf. to Figure 1).
  - *Topology 2* consists of using the device iotgw2 (a Raspberry Pi) as server/broker to control the communication between two clients.
- **Packet size**. The experiments have considered both fixed and variable size data packets (scenarios A or B, respectively):
  - a fixed-size data packet of 7 KB, standing for an example of a small packet.
  - a variable-size data packet randomly selected from an interval between 7 KB and 1000 KB (standing for an example of a large message).
- **Number of messages**. Three specific scenarios have been considered in terms of messages to be sent: 5, 500, and 5000 messages per unit of time (second).
- **Interarrival time (IAT)**. Three specific situations have been considered in terms of IAT, to respectively emulate examples of frequent, average, and not so frequent message exchange: 1 ms, 10 ms and 1000 ms.

For all local experiments, each scenario has been run for the different evaluated protocols, namely CoAP, MQTT, and OPC UA. Each experiment has been repeated 10 times.

Table 4 provides the different scenarios set for local experiments, derived from the combination of the three mentioned components.

The scripts and raw results extracted for each protocol are available via GitLab (https://gitlab.com/iotlab_copelabs (accessed on 15 May 2021)).

Later, and in order to compare the behaviour of the selected protocols on a larger scale, the FIT-IoT platform has been used.The protocols evaluated on FIT-IoT were MQTT and CoAP, as the support for OPC UA in FIT-IoT is not out-of-the-box. We have tried to install the required libraries to run OPC UA. However, the quota provided to users was not sufficient to enable the installation of this communication protocol.

The FIT-IoT scenarios are provided in Table 5, where *IAT* corresponds to Interarrival Time in milliseconds; messages sent correspond to the number of messages sent per unit of time (seconds); number of senders and receivers cover examples of a small number of senders, medium (50), or large (94). For all of the experiments run in FIT-IoT, we considered a message size randomly selected from a uniform interval between 7 KB and 1000 KB.

Table 4. Local experimental scenarios with combination of the different parameters.

Experiments	Message Size	Number of Messages	Topology
EX1	Fixed	5000	1
EX2	Fixed	500	1
EX3	Fixed	5	1
EX4	Fixed	5000	2
EX5	Fixed	500	2
EX6	Fixed	5	2
EX7	Variable	5000	1
EX8	Variable	500	1
EX9	Variable	5	1
EX10	Variable	5000	2
EX11	Variable	500	2
EX12	Variable	5	2

Table 5. FIT-IoT experimental scenarios.

Experiment	IAT (ms)	Message Sent	Number of Senders	Number of Receivers
MQTT 1	1000	5	1	1
MQTT 2	1	5000	1	1
MQTT 3	1000	5	1	94
MQTT 4	1	5000	1	94
MQTT 5	1000	470 (5 each)	94	1
MQTT 6	1	4700 (50 each)	94	1
MQTT 7	1	5000 (100 each)	50	47
CoAP 1	1000	5	1	1
CoAP 2	1	5000	1	1
CoAP 3	1000	5	1	94
CoAP 4	1	5000	1	94
CoAP 5	1	470 (5 each)	94	1
CoAP 6	1	4700 (50 each)	94	1
CoAP 7	1	5000 (100 each)	50	47

Therefore, for the experiments with the CoAP and MQTT protocols different network scenarios have been devised. Varying the number of nodes (senders and receivers), message sizes, number of messages sent, and the inter-arrival time. For all scenarios, one single broker for MQTT and one server for CoAP have been considered.

Moreover, all of the experiments have been repeated 10 times, and the values provided in this report correspond to the *Average (AVG), Standard Deviation (STDEV), Maximum (MAX), Minimum (MIN), Median (MED)*, and *Confidence Interval (IC)*.

6. Local Experiments Performance Evaluation

6.1. Topology 1 and Fixed Message Size

A first set of experiments has been run by combining Topology 1 and fixed message size, being the results for MQTT, CoAP, and OPC UA respectively presented in Tables 6–8.

Looking into the results obtained for the time-to-completion (TTC, in milliseconds), CoAP provides the lowest values for frequent messages (short and average IAT). However, for larger IAT, CoAP is the protocol that provides the largest TTC (1051 ms). An explanation for this pattern concerns the acknowledgement behaviour of CoAP. For messages that have a large interarrival time (IAT), lack of synchronization may occur. However, we have also observed some peaks in TTC derived from the shared usage of the Internet access connection between the local testbed, and the Internet campus access.

A similar behavior, showing more variability, is also observed in MQTT.

OPC UA is the protocol that provides the most stable results for TTC, when the IAT is varied, even though the resulting TTC is slightly, but not significantly, higher. In the experiments carried out at the local testbed, the experiments incurred zero packet loss, for all rounds of execution in the three protocols tested.

Table 6. MQTT results for topology 1 and fixed message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	20.86	6.34	27.61	9.31	22.30	3.93
10 ms	TTC	18.81	13.00	38.20	6.75	11.85	8.06
1000 ms	TTC	84.62	17.48	110.81	57.40	89.25	10.83

Table 7. CoAP results for topology 1 and fixed message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	11.99	0.92	13.37	10.68	12.02	0.57
10 ms	TTC	14.88	1.23	16.76	13.06	14.92	0.76
1000 ms	TTC	1015.28	119.60	1349.20	950.60	977.30	74.13

Table 8. OPC UA results for topology 1 and fixed message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	22.02	1.34	24.82	20.31	22.02	0.83
10 ms	TTC	30.92	3.76	39.90	27.05	29.54	2.33
1000 ms	TTC	47.42	34.54	125.23	26.56	32.43	21.41

As for packet loss, none of the protocols showed any packet loss for the scenarios run, as expected, given that the generated message rate does not fill the available link capacity.

6.2. Topology 1, Variable Message Size

The second set of experiments considers topology 1 and messages of variable size. The message size has been randomly selected from an interval between 7 KB and 1000 KB, as explained before. Results for the 3 protocols are respectively provided in Tables 9–11.

Table 9. MQTT results for topology 1 and variable message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	138.62	171.90	485.18	22.55	47.37	106.54
10 ms	TTC	10.67	6.92	29.46	6.79	8.06	4.29
1000 ms	TTC	80.32	9.55	94.27	68.01	79.87	5.92

Table 10. CoAP results for topology 1 and variable message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	16.81	2.84	22.22	14.54	15.61	1.76
10 ms	TTC	12.52	2.01	16.11	9.70	12.21	1.24
1000 ms	TTC	74.09	9.09	92.50	59.20	74.10	5.64

Table 11. OPC UA results for topology 1 and variable message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	59.41	0.76	67.10	59.41	65.61	0.47
10 ms	TTC	42.13	32.31	67.10	43.13	59.41	20.02
1000 ms	TTC	437.07	1279.65	4079.00	26.76	32.79	793.12

To better compare the results, Figure 5 provides all of the results obtained with the experiments based on topology 1.

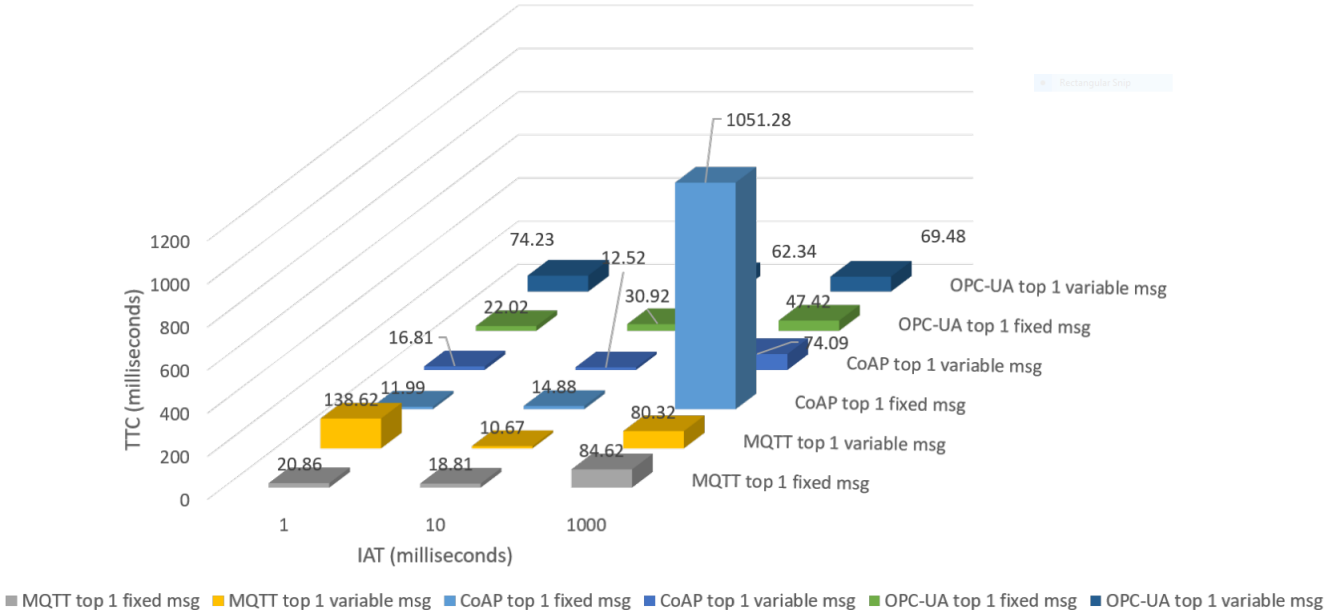


Figure 5. Topology 1 results.

In terms of performance when considering fixed message sizes vs. variable message sizes, all protocols exhibit a similar performance, even though CoAP shows a spike when messages are less frequent (IAT of 1000 ms). This behavior was observed in all of the repeated experiments, thus implying that the setup mechanism of CoAP impacts significantly TTC.

If we consider now the performance due to changes in the frequency of messages (IAT), then OPC-UA is the protocol that provides closer TTC results across all experiments, and CoAP the protocol that is more sensitive to the frequency of messages less frequent messages have more impact in the TTC.

Overall, all protocols show good performance for this set of experiments (minimum TTC value: 10.67 ms, MQTT; maximum TTC value: 1051 ms, CoAP).

6.3. Topology 2, Fixed Message Size

The next set of experiments considers Topology 2, where gateways and brokers have been installed in embedded devices, and where the message size is fixed. Results for MQTT, CoAP and OPC UA are respectively provided in Tables 12–14.

Globally, CoAP is again the protocol that provides the lowest TTC values for frequent messages (short and average IAT). MQTT also attains low TTCs, but shows more variability, as occurred with the experiments run on topology 1.

OPC UA is again the protocol that shows less sensitivity to the changes in IAT, with TTC values between 56 ms and 65 ms.

Table 12. MQTT results for topology 2 and fixed message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	44.15	24.44	80.35	16.14	45.33	45.33
10 ms	TTC	34.32	40.22	107.43	8.69	9.22	24.93
1000 ms	TTC	76.91	10.28	88.41	58.31	75.80	6.37

Table 13. COAP results for topology 2 and fixed message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	7.28	1.34	9.66	5.36	7.06	0.83
10 ms	TTC	16.20	7.73	28.72	7.15	14.68	4.79
1000 ms	TTC	70.22	9.82	89.60	55.60	71.50	6.08

Table 14. OPC UA results for topology 2 and fixed message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	74.23	13.06	97.30	62.78	68.72	8.09
10 ms	TTC	62.34	6.00	73.02	55.51	61.23	3.72
1000 ms	TTC	69.48	29.66	152.81	51.21	61.83	18.38

6.4. Topology 2, Variable Message Size

The experiments have been run again now with variable message size. Results are provided in Tables 15–17.

Table 15. MQTT results for topology 2 and variable message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	105.26	209.97	702.01	20.30	39.41	130.14
10 ms	TTC	15.76	4.76	26.16	11.95	13.53	2.95
1000 ms	TTC	90.99	11.96	106.36	67.60	92.02	7.41

Table 16. CoAP results for topology 2 and variable message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	10.47	4.05	18.89	7.55	8.73	2.51
10 ms	TTC	20.25	14.51	50.72	9.67	13.50	8.99
1000 ms	TTC	118.33	144.31	517.00	22.67	79.70	89.44

Table 17. OPC UA results for topology 2 and variable message size.

Test		AVG	STDEV	MAX	MIN	MED	IC
1 ms	TTC	65.62	0.76	67.10	64.67	65.61	0.47
10 ms	TTC	60.50	3.61	66.78	56.71	59.79	2.24
1000 ms	TTC	56.30	7.22	72.09	47.42	55.60	4.48

To better provide a comparison of results obtained with the experiments, Figure 6 provides results obtained for all of the experiments run with topology 2.

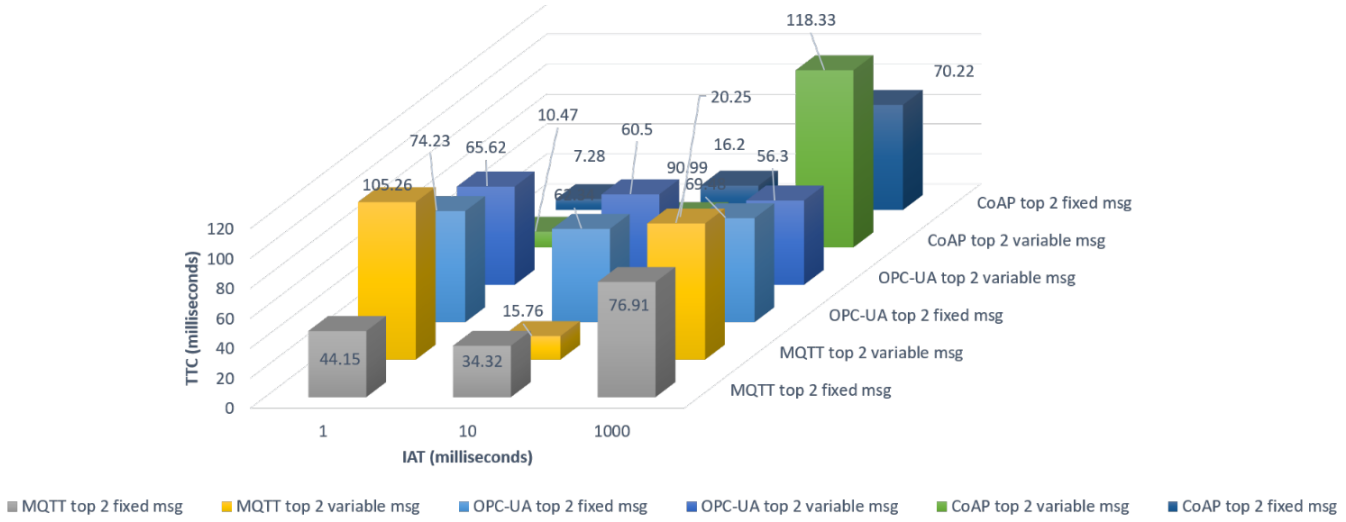


Figure 6. Full set of topology 2 results.

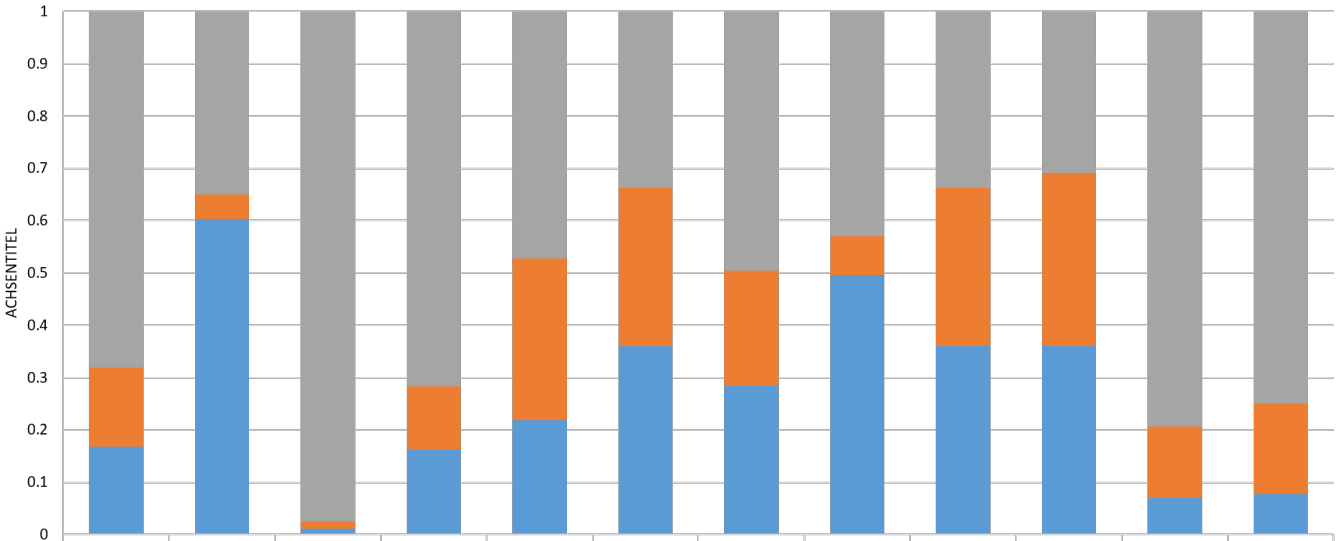
In terms of performance when considering fixed message sizes vs. variable message sizes, all protocols exhibit a similar performance, even though MQTT shows slightly more variability when the message size is varied. In contrast to the results obtained with topology1, CoAP shows stable results.

If we consider now the performance due to changes in the frequency of messages (IAT), OPC UA is again the protocol that provides closer TTC results across all experiments, and CoAP the protocol that is more sensitive to the frequency of messages less frequent messages have more impact in the TTC. MQTT has values similar to CoAP.

Overall, all protocols again show good performance for this set of experiments. The minimum TTC value is 7.27 ms, MQTT (in topology 1:10.67 ms, MQTT); the maximum TTC value is 118.33 ms, CoAP (in topology 1:1051 s, CoAP). Therefore, the fact that the broker and/or server was installed on an embedded device did not have much impact on the performance of the protocols, for the experiments run.

6.5. Summary of Results

This section summarizes the results obtained across all experiments and which are represented in Figure 7.



	MQTT top 1 fixed msg	MQTT top 1 variable msg	CoAP top 1 fixed msg	CoAP top 1 variable msg	OPC-UA top 1 fixed msg	OPC-UA top 1 variable msg	MQTT top 2 fixed msg	MQTT top 2 variable msg	OPC-UA top 2 fixed msg	OPC-UA top 2 variable msg	CoAP top 2 variable msg	CoAP top 2 fixed msg
1000	84.62	80.32	1051.28	74.09	47.42	69.48	76.91	90.99	69.48	56.3	118.33	70.22
10	18.81	10.67	14.88	12.52	30.92	62.34	34.32	15.76	62.34	60.5	20.25	16.2
1	20.86	138.62	11.99	16.81	22.02	74.23	44.15	105.26	74.23	65.62	10.47	7.28

Figure 7. TTC achieved by MQTT, CoAP and OPC UA across all experiments, for an IAT of 1,10 and 1000 milliseconds.

The main aspects to highlight are:

- MQTT behavior is highly variable for different IAT and for different message sizes. In some experiments, the resulting TTC is high for smaller IATs, and on other experiments, becomes larger for higher IATs. What would be expected would be to see an increase of TTC with an increase in IAT. This situation was observed several times, and we believe it is due to the fact that there is no queuing support in MQTT. The broker has to handle very different message sizes.
- CoAP presents more stability for the variation of low/medium IATs and message size. However, for the case of large IAT, the resulting TTC is significantly higher in comparison to other CoAP TTCs, as well as to the overall results obtained for the other 2 protocols.
- OPC UA is less sensitive to variations of the IAT. The fundamental reason for the operation of OPC UA is, however, not clear to us, and requires further investigation to understand this stability whether it is derived from the selection of scenario parameters, or really derived from the communication semantics of OPC UA. In regards to the impact of the message size, OPC UA shows more sensitivity to variable message sizes. Moreover, when compared to MQTT and to CoAP, the resulting TTC is overall higher, even though the difference is in milliseconds.
- The lowest TTC is achieved by CoAP, for the case of an IAT of 1ms (topology 2, fixed message size). We believe this happens due to the reliability exponential backoff mechanism of CoAP, which is beneficial for frequent messages.
- When the IAT is 1000 ms, MQTT and OPC UA result in similar TTCs, while CoAP shows significantly higher TTCs. Although the IAT impacts the TTC computation, the CoAP behavior seems to be derived from the fact that with larger intervals between messages, the CoAP establishment process has implications on the TTC. In other words, CoAP seems to suit best scenarios where messages are more frequent.
- In regards to the change of topology, all protocols experience some impact, as expected. However, the impact is not significant. The difference observed for all protocols in terms of differences of TTC is in the order of milliseconds.
- Across the different experiments, the overall TTC is low and suitable for critical IoT environments.

7. FIT-IoT Performance Evaluation

Several experiments have been carried out in FIT-IoT, being the parameters of each scenario, as well as the results obtained for the TTC and packet loss summarised in Table 18. The different experiments are debated next.

Table 18. FIT-IoT scenarios with parameters and results obtained.

Experiment	Senders	Receivers	Messages	IAT (ms)	TTC (ms)	Packet Loss (Percentage)
Controlled Experiments						
MQTT 1	1	1	5	1000	35.17	0
CoAP 1	1	1	5	1000	185.17	0
MQTT 2	1	1	5000	1	19.61	0
CoAP 2	1	1	5000	1	246.33	0
1 to Many Experiments						
MQTT 3	1	94	5	1000	136.97	0
CoAP 3	1	94	5	1000	4220.02	0
MQTT 4	1	94	5000	1	78,897.46	65.16
CoAP 4	1	94	5000	1	4526.39	0.04
Many to 1 Experiments						
MQTT 5	94	1	5/sender	1000	102.232	0
CoAP 5	94	1	5/sender	1000	2631.62	0
MQTT 6	94	1	50/sender	1000	5419.86	0
CoAP 6	94	1	50/sender	1000	4351.83	0.003
Many to Many Experiments						
MQTT 7	50	47	100/sender	1000	117,814.03	0.01
CoAP 7	50	47	100/sender	1	3611.39	0.131

7.1. Controlled Experiments

The first controlled experiments, MQTT-1 and CoAP-1, involve 1 sender, 1 receiver, and a small number of messages (5) with a large IAT. Experiments MQTT-2 and CoAP-2 experiments involve also just 1 sender and 1 receiver, but a higher number (5000) of very frequent messages (1 ms IAT). Results obtained for the initial set of experiments (control experiments) are illustrated in Figure 8.

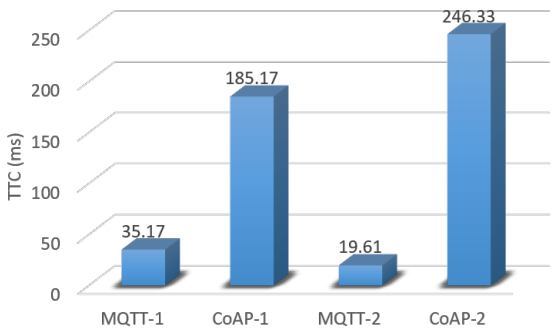


Figure 8. Controlled Experiments.

For both sets of experiments, MQTT attains a lower TTC on average, compared to the one of CoAP. This difference is possibly due to the stop-and-wait retransmission reliability with exponential back-off process of CoAP.

7.2. 1 to Many Experiments

The first set of 1-to-many experiments, MQTT-3 and CoAP-3, involve 1 sender and 94 receivers, where the sender sends 5 messages (subscribed by each receiver/client), separated by an IAT of 1000 ms. Then, MQTT-4 and CoAP-4 involve a higher number (5000) of very frequent messages (1 ms IAT). Results are summarised in Figure 9.

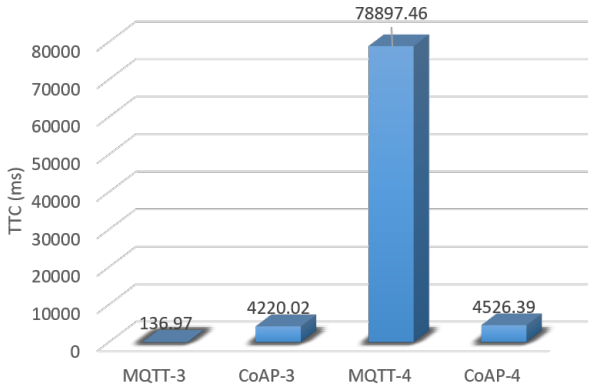


Figure 9. 1 to Many Experiments.

In comparison to the prior experiment with 1 sender and 1 receiver (rf. to Figure 8), MQTT slightly increases the average TTC from 35 ms to 136.97 ms, while CoAP shows a significant difference, from 185.17 ms to 4220 ms.

Increasing the frequency of messages has more impact in MQTT than in CoAP. In fact, for the MQTT case there is a packet loss of 65%. While CoAP incurred a packet loss of 0.04%. This seems to imply that MQTT brokers cannot cope well with a lot of information being requested in parallel by different subscribers. We believe that this is due to the lack of queueing in MQTT, and also to the way that the messages are served.

CoAP, on the other hand, attained a slightly higher average TTC (approximately 300 ms) when compared to the experiment run with not so frequent messages. We believe this occurs due to the way the CoAP server stores the content by senders: the initial setup process of communication between the sender and CoAP is possibly only set once; the reliability process with exponential backoff adjusts to the frequency of messages, thus resulting in a more stable TTC across all experiments.

7.3. Many to 1 Experiments

In this set of experiments we have first run MQTT-5 and CoAP-5, which are based on 94 senders, each of which sends 5 messages with an IAT of 1000 ms.

Then, in order to understand the impact of the frequency of messages, we have run experiments MQTT-6 and CoAP-6, where each of the 94 senders now send 50 messages separated by an IAT of 1000 ms. Results for this set are provided in Figure 10.

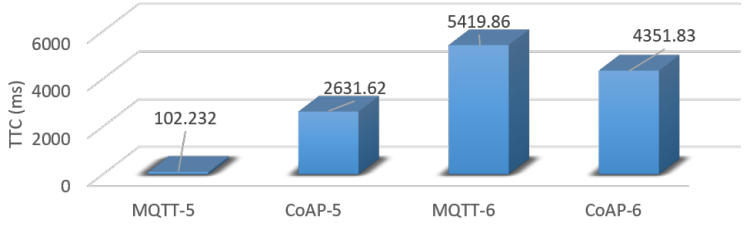


Figure 10. Many to 1 Experiments.

If we compare this experiment with the results obtained in the prior set (1 to many, rf. to Figure 9, MQTT shows a similar TTC, while CoAP shows a reduced value in comparison to the TTC it attained for the many to 1 experiments. The values obtained for MQTT can be explained by the fact that the frequency of messages is low. Even though there are several senders in parallel, MQTT can handle the service well without queueing. As for the resulting average TTC for CoAP, the lower value is, in our opinion, simply due to the queuing process of CoAP, which stores data for each sender, thus allowing a better queuing service from the client perspective: in the prior case, the clients obtain the information from the same memory space (1 sender, many receivers).

In comparison to the 1-to-many set of experiments (rf. to Figure 9, MQTT seems to be able to handle better a large number of senders, even if the frequency of messages increases. CoAP, on the other hand, keeps its stability, achieving a similar TTC to most cases. However, we should highlight that for experiment 6 (CoAP-6) there is also a slight packet loss (0.3%).

7.4. Many to Many Experiment

A final set of experiments involving a larger number of senders and receivers, comprising MQTT-7 and CoAP-7, has been carried out. The frequency of messages has also been increased.

For this set of experiments (rf. to Table 7), the TTC obtained for MQTT is the highest (of all cases), while CoAP keeps an average of 3611 ms, quite similar to the other experiments involving high intensity of messages. Both protocols experienced a small level of packet loss (0.01% for MQTT; 0.13% for CoAP). This again seems to imply that MQTT experiences more problems with an increase in the number of receivers than CoAP, in particular assuming a higher frequency of sent messages.

7.5. Discussion

Figure 11 illustrates the full set of results for MQTT and CoAP. Even though MQTT attains in 4 out of 7 experiments a lower TTC, on the other 3 experiments MQTT shows a greater variability in results associated with both a higher frequency of messages and a higher number of receivers (MQTT-4, MQTT-6, MQTT-7).

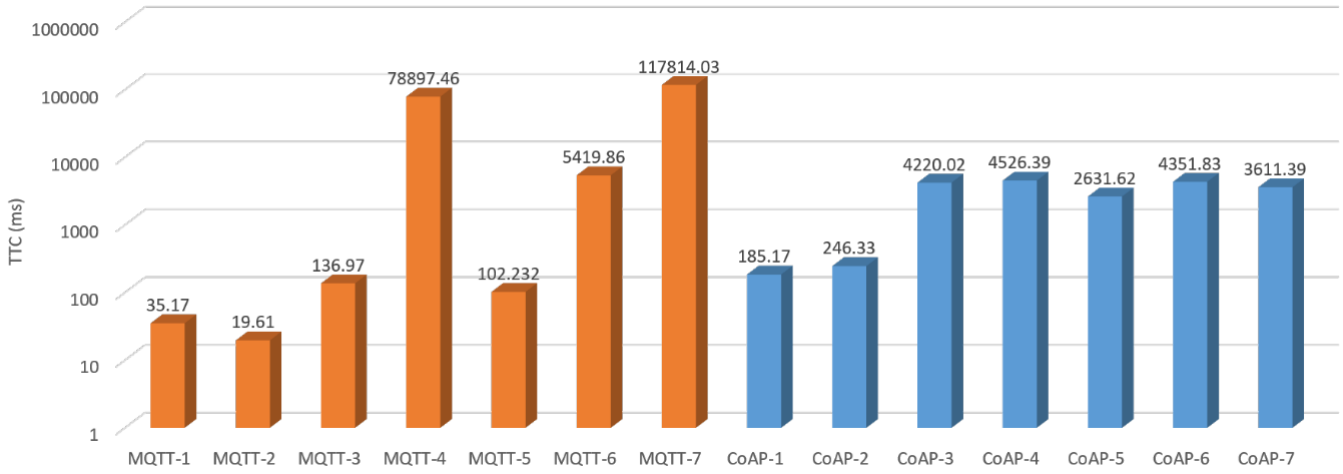


Figure 11. FIT-IoT results obtained for MQTT and CoAP.

CoAP attains a higher TTC for scenarios when the frequency of messages is low (CoAP-1 and CoAP-2), but overall it shows a similar TTC across all scenarios. This seems to imply that CoAP is therefore more stable than MQTT to an increase in the number of senders/receivers, as well as to an increase in the frequency of messages to be exchanged.

In both protocols messages are exchanged asynchronously. In CoAP, messages are stored per sender/server connection, while in MQTT they are only stored if receivers

(client) subscribed them first. For a large number of receivers, there is the need to complete first subscription. MQTT therefore results in higher TTCs and packet loss for scenarios with a larger number of receivers (MQTT-4 and MQTT-7). While for the case where there is a large number of senders involved, but a low number of receivers (rf. to MQTT-5 or MQTT-6), there is an increase of TTC as expected, but no packet loss.

The CoAP reliability mechanism provides more stability to an increase in the number of both senders and receivers, thus making this protocol more suitable for large-scale heterogeneous IoT environments.

8. Conclusions and Future Work

This paper contributes to the performance analysis and evaluation of different IoT communication solutions, by first providing a comparison of different protocolar features, to then perform experiments with MQTT, CoAP, and OPC UA in both local experimentation environment and in the large-scale experimental facility FIT-IoT.

In terms of protocolar features most protocols already support both TCP and UDP traffic, being the exception MQTT. UDP support is highly relevant in future IoT environments, as the devices present will become more mobile. Another aspect which requires further future work is service decentralisation, and for that, communication protocols need to be designed in a way that address variable topologies. Today, only DDS and NDN truly support decentralised communication. In regards to security, some solutions, such as QUIC, OPC UA and DDS, provide a specific security framework, which brings in more flexibility and better security in terms of specific domains. Only NDN provides security by design, while the remainder solutions recur to existing solutions, such as DTLS. Other relevant aspects in IoT, such as data/device discovery, are still articulated in a manual way, being the exception DDS and NDN. In terms of end-to-end coverage, the different solutions have mostly been developed to provide support between devices and the edge. They can be applied to cover other regions, e.g., edge to cloud. D2D support is only directly supported by DDS and NDN.

Summarising, from a design perspective, DDS and NDN are the solutions that seem to provide the most flexible design, integrated by design a set of features that are relevant when thinking about the potential evolution of IoT environments. DDS is, however, domain specific (manufacturing), and NDN is still not being developed at a commercial IoT level.

In regards to the other solutions, OPC UA provides good support in terms of in-plant communication, thus being the de facto communication standard in industrial automation environments. CoAP provides a high level of flexibility and is today one of the most popular solutions for IoT communication. Its main advantage is its flexibility and interoperability towards HTTP.

From a performance evaluation, this work has focused on comparing MQTT, CoAP, and OPC UA in regards to time-to-completion and packet loss, under different conditions. The experiments run on local testbed showed that any of these protocols can be run in an embedded environment without significant impact in the selected KPIs. Moreover, all protocols achieved low time-to-completion times across all of the tested scenarios, aspect which is relevant for their application in critical environments.

CoAP is the protocol that achieved the lowest time-to-completion, which we believe is due to the reliability exponential backoff mechanism, beneficial for environments with frequent messaging.

MQTT results in lower TTCs, but exhibited a larger variability for the different scenarios, being highly dependent on the frequency of messages and on message size.

OPC UA shows less sensitivity to variations on the frequency of messages. The fundamental reason for this requires further analysis and is an aspect to pursue as future work whether or not this stability is derived from the communication semantics of OPC UA. However, OPC UA resulted overall in higher TTCs, when compared to CoAP and to MQTT.

The experiments that have been repeated in large-scale environments (FIT-IoT) for CoAP and MQTT show that these protocols have a similar behavior to the one observed in the testbed. A detected limitation of the experimentation carried out concerns the lack of experimental, large-scale platforms, easy to use, to carry out experiments in scenarios with more variability. A second detected limitation concerned the impossibility, at the time of the experiments, to have OPC UA running in FIT-IoT.

As future work, we shall continue with performance evaluation of the different protocols. Specifically, OPC UA will be tested for large-scale scenarios, and we intend to also experiment with DDS and MQTT Sparkplug for the specific case of Industrial IoT environments. In our opinion, the comparison should be done to CoAP at least, given that this protocol provides better results than MQTT. Moreover, further experimentation comparing OPC UA with DDS and with NDN is being pursued.

Author Contributions: Conceptualization, D.S., L.I.C. and R.C.S.; methodology, D.S. and R.C.S.; software, D.S. and J.S.; validation, R.C.S.; formal analysis, D.S. and R.C.S.; investigation, L.I.C.; data curation, D.S., J.S. and R.C.S.; writing original draft preparation, D.S. and R.C.S.; writing review and editing, R.C.S.; All authors have read and agreed to the published version of the manuscript.

Funding: This paper has been funded by the strategic FCT research project project UIDB/04111/2020, associated with the research unit COPELABS, University Lusofona.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. De Caro, N.; Colitti, W.; Steenhaut, K.; Mangino, G.; Reali, G. Comparison of two lightweight protocols for smartphone-based sensing. In Proceedings of the 2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT), Namur, Belgium, 21 November 2013; pp. 1–6.
2. Amaran, M.H.; Noh, N.A.M.; Rohmad, M.S.; Hashim, H. A comparison of lightweight communication protocols in robotic applications. *Procedia Comput. Sci.* **2015**, *76*, 400–405. [CrossRef]
3. Liri, E.; Singh, P.K.; Rabiah, A.B.; Kar, K.; Makhijani, K.; Ramakrishnan, K. Robustness of IoT Application Protocols to Network Impairments. In Proceedings of the 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Washington, DC, USA, 25–27 June 2018; pp. 97–103.
4. Gündoğan, C.; Kietzmann, P.; Lenders, M.; Petersen, H.; Schmidt, T.C.; Wählisch, M. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. *arXiv* **2018**, arXiv:1806.01444.
5. Durkop, L.; Czybik, B.; Jasperneite, J. Performance evaluation of M2M protocols over cellular networks in a lab environment. In Proceedings of the 2015 18th international conference on Intelligence in Next Generation Networks, Paris, France, 17–19 February 2015; pp. 70–75.
6. Bacco, M.; Colucci, M.; Gotta, A. Application protocols enabling internet of remote things via random access satellite channels. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
7. Profanter, S.; Tekat, A.; Dorofeev, K.; Rickert, M.; Knoll, A. OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols. In Proceedings of the IEEE International Conference on Industrial Technology (ICIT), Melbourne, VIC, Australia, 13–15 February 2019.
8. Proos, D.P.; Carlsson, N. Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV. In Proceedings of the 2020 IFIP Networking Conference (Networking), Paris, France, 22–26 June 2020; pp. 10–18.
9. Adlink Tech. Messaging Technologies for the Industrial Internet and the Internet of Things Whitepaper. Technical Report, Adlink Tech. 2017. Available online: https://iotbusinessnews.com/download/white-papers/PRISMTECH-messaging-technologies-for-Industrial-Internet-and-IoT.pdf (accessed on 22 June 2019).
10. OASIS. AMQP Advanced Message Queuing Protocol. 2018. Available online: http://www.amqp.org/ (accessed on 10 June 2019).
11. Vinoski, S. Advanced message queuing protocol. *IEEE Internet Comput.* **2006**, *10*, 87–89. [CrossRef]
12. Garcia, C.G.; Garcia-Diaz, V.; Garcia-Bustelo, B.; Lovelle, J.M.C. *Protocols and Applications for the Industrial Internet of Things*; IGI Global: Hershey, PA, USA, 2018.
13. Sebastian Raff. The MQTT Community. Available online: https://github.com/mqtt/mqtt.github.io/wiki (accessed on 10 June 2019).

14. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). Technical Report. 2014. Available online: https://www.rfc-editor.org/info/rfc7252 (accessed on 11 May 2021).
15. Selander, G.; Mattsson, J.; Palombini, F.; Seitz, L. IETF RFC8313 Object Security for Constrained RESTful Environments (OSCORE). Technical Report. 2019. Available online: https://www.hjp.at/doc/rfc/rfc8613.html (accessed on 11 May 2021).
16. Cavalieri, S.; Chiacchio, F. Analysis of OPC UA performances. *Comput. Stand. Interfaces* **2013**, *36*, 165–177. [CrossRef]
17. OPC Foundation. OPC Unified Architecture Interoperability for Industrie 4.0 and the Internet of Things. pp. 1–44. Available online: https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf (accessed on 22 June 2019).
18. Leitner, S.-H.; Mahnke, W. *OPC-UA/Service-Oriented Architecture for Industrial Applications*; Technical Report; ABB Corporate Research Center: Västerås, Sweden, 2006.
19. OMG. DDS Data Distribution Service. Available online: http://portals.omg.org/dds/what-is-dds-3/ (accessed on 10 June 2019).
20. Langley, A.; Iyengar, J.; Bailey, J.; Dorfman, J.; Roskind, J.; Kulik, J.; Westin, P.; Tenneti, R.; Shade, R.; Hamilton, R.; et al. The QUIC Transport Protocol. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication SIGCOMM '17, Los Angeles, CA, USA, 21–25 August 2017; pp. 183–196. [CrossRef]
21. Eggert, L.; Nottigham, M.; Dawkins, S. QUIC Working Group. Available online: https://datatracker.ietf.org/wg/quic/about/ (accessed on 27 August 2019).
22. Meddeb, M. Information-Centric Networking, A Natural Design for IoT Applications? Ph.D. Thesis, INSA Toulose, Toulouse, France, 2017.
23. Kutscher, D. IRTF Information-Centric Networking Research Group (ICNRG). Available online: https://datatracker.ietf.org/rg/icnrg/about/ (accessed on 11 May 2021).
24. C Sofia, R.; M Mendes, P. An Overview on Push-Based Communication Models for Information-Centric Networking. *Future Internet* **2019**, *11*, 74. [CrossRef]
25. Stanford-Clark, A.; Truong, H.L. MQTT For Sensor Networks (MQTT-SN) Protocol Specification. *IBM Protoc. Specif.* **2013**, 28.
26. Kerrouche, A.; Senouci, M.R.; Mellouk, A. QoS-FS: A new forwarding strategy with QoS for routing in Named Data Networking. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 22–27 May 2016; pp. 1–7. [CrossRef]
27. Amadeo, M.; Campolo, C.; Molinaro, A.; Ruggeri, G. IoT Data processing at the Edge with Named Data Networking. In Proceedings of the European Wireless 2018, 24th European Wireless Conference, Catania, Italy, 2–4 May 2018.
28. Yang, J. Data Distribution Service for Industrial Automation. In Proceedings of the IEEE 17th Conference in Emerging Technologies and Factory Automation (ETFA), Krakow, Poland, 17–21 September 2012; pp. 1–8.
29. Baccelli, E.; Mehlis, C.; Hahm, O.; Schmidt, T.; Wählisch, M. Information Centric Networking in the IoT: Experiments with NDN in the Wild. In Proceedings of the 1st ACM Conference on Information-Centric Networking, Paris, France, 24 September 2014; pp. 77–86.
30. Adjih, C.; Baccelli, E.; Fleury, E.; Harter, G.; Mitton, N.; Noel, T.; Pissard-Gibollet, R.; Saint-Marcel, F.; Schreiner, G.; Vandaele, J.; et al. FIT-IoT Lab: A large scale open experimental IoT testbed. In Proceedings of the Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum, Milan, Italy, 14–16 December 2015; pp. 459–464.
31. Soares, J.; Silva, D.; Sofia, R.C. *The IoT Testbed at Copelabs Descripion and Experiments*; Technical Report; COPELABS, University Lusofona: Lisboa, Portugal, 2019; [CrossRef]
32. Kim, J.; Lee, J.W. OpenIoT: An open service framework for the Internet of Things. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; pp. 89–93.
33. RabbitMQ. Available online: https://www.rabbitmq.com (accessed on 8 July 2019).
34. Mosquitto. Available online: https://mosquitto.org (accessed on 8 July 2019).
35. Fambon, O.; Fleury, E.; Harter, G.; Pissard-Gibollet, R.; Saint-Marcel, F. FIT-IoT Lab tutorial: Hands-on practice with a very large scale testbed tool for the Internet of Things. *10èmes J. Francoph. Mobilité Ubiquité UbiMob2014* **2014**.