A Scalable and Secure Publish/Subscribe-based Framework for Industrial IoT

# A Scalable and Secure Publish/Subscribe-based Framework for Industrial IoT

Michele Amoretti, *Senior Member, IEEE,* Riccardo Pecori, Yanina Protskaya, Luca Veltri, and Francesco Zanichelli, *Member, IEEE*

***Abstract*—In the emerging Industrial Internet of Things (IIoT) scenario machine-to-machine communication is a key technology to set up environments wherein sensors, actuators, and controllers can exchange information autonomously. However, many current communication frameworks do not provide enough dynamic interoperability and security. Hence, we propose a novel communication framework, based on MQTT broker bridging, which, in an Industrial IoT scenario, can foster dynamic interoperability across different production lines or industrial sites, guaranteeing, at the same time, a higher degree of isolation and control over the information flows, thereby increasing the overall security of the whole scenario. The solution we propose does also support dynamic authentication and authorization and has been practically implemented and evaluated in a proper small-scale IIoT testbed, encompassing PLCs, IIoT gateways, as well as MQTT brokers with novel and extended capabilities. The evaluation results demonstrate a linear time complexity for all the considered implementations and bridging modes of the extended brokers. Moreover, all considered access token encapsulation techniques demonstrate a minimum overhead in comparison with standard MQTT brokers.**

***Index Terms*—MQTT; Industrial IoT; Broker Bridging; Security; Authentication; Authorization.**

## I. Introduction

The concepts of Internet of Things (IoT), Industrial Internet of Things (IIoT) and Cyber-Physical Systems (CPSs) are often overlapping when the definition of Industry 4.0 is concerned [1]. Indeed, IIoT is a subset of IoT, regarding the application of traditional IoT principles to the manufacturing industry. More precisely, an IIoT system involves networked smart objects, cyber-physical assets, and optional Cloud or edge computing platforms, to enable real-time, intelligent, and autonomous access, collection, analysis, communications, and exchange of processes, products and services, in the industrial environment [2]. Thus, IIoT focuses on the integration of operational technology with information technology, on machine-to-machine (M2M) communications between modern and legacy isolated plants, on a very high number of interconnected devices, and on mission critical requirements. Therefore, IIoT generally results into a peculiar distributed system [3], which has to address a series of challenges and issues such as: interoperability, scalability, and dependability, inflected as reliability, safety against potential catastrophes, availability over very long periods, self maintainability and cybersecurity, to guarantee secure and reliable communications, as well as sophisticated identity and access management of machines and users. However, the constrained and heterogeneous nature of IoT devices, as well as the peculiarities of industrial environments, make it very difficult to apply well-known standard security solutions.

On the other hand, Message Queuing Telemetry Transport (MQTT) [4], is becoming the *de facto* standard messaging protocol for many M2M communications in IIoT scenarios, due to its lightweight overhead, publish/subscribe (pub/sub) model, and bidirectional capabilities. A standard MQTT system consists of one or more publishers and one or more subscribers communicating via a broker node performing message dispatching. Brokers can be also connected together: i) either grouped in a cluster, exchanging data according to their current clients' subscriptions and publications, or ii) linked by a sort of bridging [4], wherein a broker is instructed to connect to another broker, acting as a fictitious MQTT client. However, these solutions require manual configuration, thus hindering scalability; furthermore, they do not provide standard security mechanisms, and cannot cope with highly dynamic and mutable IIoT scenarios.

In the light of the aforementioned limitations, this paper introduces the following contributions:

- a novel MQTT-based framework making the M2M messaging secure and scalable, by leveraging an extended broker bridging mechanism also exploiting new authentication and authorization schemes;
- the application of the novel MQTT-based framework to an IIoT scenario, providing better manageability and isolation of information flows across different industrial sites and production lines;
- a practical implementation of the proposed framework in a small-scale industrial testbed, featuring PLCs, IIoT gateways and brokers endowed with the novel extended capabilities;
- the evaluation and comparison of different implementations of the extended brokers and of the various bridging modes therein, as well as of the overhead introduced by the novel extended capabilities of the brokers with that of standard MQTT-based brokers.

The rest of this paper is organized as follows. In Section II, we present a review of a few MQTT approaches for IIoT scenarios, and a few techniques to secure MQTT. In Section III, a high-level description of the organization

M. Amoretti, L. Veltri, and F. Zanichelli are with the Dept. of Engineering and Architecture, University of Parma, Italy (email: name.surname@unipr.it).

R. Pecori is with the Dept. of Engineering, University of Sannio, Benevento, Italy (email: rpecori@unisannio.it). His work has been supported by the PON R&I 2014-2020 AIM project.

Y. Protskaya is with Maps Group, Parma, Italy.

of IIoT production systems based on a pub/sub paradigm is provided. In Section IV, we present the new dynamic bridging mechanism, how to secure it and its authorization capabilities. In Section V, a practical implementation of the proposed messaging framework, together with some experimental evaluations, are described. Section VI draws some conclusions and shows possible future works.

## II. Related Work

In this section, we briefly review some recent usages of MQTT in IIoT scenarios, classifying them in those involving MQTT but neither broker bridging nor security, those involving only MQTT broker bridging or only MQTT security, and finally those encompassing both.

Regarding solutions which apply MQTT in an IIoT scenario without featuring neither broker bridging nor security, we can find the work of Peralta et al. [5], wherein MQTT brokers, acting also as edge gateways, are used together with Fog Computing in an IIoT architecture. The brokers perform local predictions to save transmission energy, but they do not perform any action with respect to security nor bridging capabilities are considered. In [6], an MQTT broker is used as an IIoT gateway for a CPS in an Industry 4.0 scenario. The author proposes an 8-component architecture for improving horizontal integration in the overall framework. However, the proposed architecture still lacks security, left as a future work, and broker bridging is not considered at all. The application of MQTT to manufacturing factories is also testified by the studies in [7] and [8]. In the former, MQTT is used to extend the Manufacturing Message Specification for controlling multiple devices in an IIoT real-time platform, while in the latter, a framework for the integration of IoT technologies into industrial systems of small and medium enterprises is presented. The framework encompasses 5 layers and MQTT is employed in both the communication and middleware layers to manage data communications. However, both works completely disregard any consideration about security or broker bridging.

On the other hand, we can find solutions considering security, but no broker bridging, like the contribution in [9] which does not even consider MQTT, but proposes a lightweight authentication mechanism for M2M IIoT communications, based only on hash and XOR operations. The procedure prevents devices from revealing their real identities in authenticating to the routers, and is based on authentication servers. A secure communication scheme named MQTT-Auth has been recently proposed in [10], [11]. It is a distributed solution, based on the AugPAKE algorithm for setting up a session key and on authentication and authorization tokens, which are used to publish on a certain topic and to grant access to a specific topic, respectively. Even if tackling security aspects, this solution does not consider broker bridging at all. In [12] the authors design and implement secure versions of MQTT and MQTT-SN, in which security is based on Key/Ciphertext Policy-Attribute Based Encryption (KP/CP-ABE). This solution supports broadcast encryption that allows an entity to deliver a message to multiple intended users performing only one encryption; however, broker bridging is not taken into account. Finally, in [13] an access control system to manage the authentication of a publisher using an authentication server for communication over MQTT has been described. The assumed network architecture is based on Fog computing and publishers obtain tokens from the authentication server via HTTPS.

Regarding dynamic MQTT bridging, but with no focus onto security capabilities, we can name EMMA [14], a middleware that migrates clients to brokers, by means of edge gateways and according to the average latency and load balancing of connections. Another possible solution featuring broker bridging can be found in [15], wherein an overlay of brokers and bridging manager servers are employed. This solution provides a great overhead in terms of exchanged control messages. Finally, in [16] the authors analyze different distributions of brokers, but only in a shallow way. Moreover, some solutions are static, while the dynamic ones involve numerous other entities, i.e., rendez-vous nodes, or rely onto very invasive solutions such as Information Centric Networking.

As regards contributions dealing with both broker bridging and security in MQTT, we can name [17], where the authors exploit dynamic bridging, but in the anonymization context, with no focus onto authentication and authorization. In [18] the authors exploit, in a dynamic and distributed way, the usage control some of them had previously presented in [19]. The brokers can share attributes or retrieve the values of remote attributes, but they need to exploit a modified Chord hashtable [20]; moreover, the brokers and all devices need to be endowed with proper software modules, which perform a continuous and ongoing re-verification of the authorization attributes, considering also the context. This may lead to an overhead in the number of exchanged messages for continuously verifying attributes. In [21], Schmitt et al. tackle the broker bridging topic by taking advantage of an agents' system. However, the proposed solution does not employ a standard MQTT mechanism, it is not clear which clients can modify the bridging topology, and the security aspect is demanded to a very weak ACL-based solution.

In the light of what said above, our solution is one of the few that feature both security and dynamic broker bridging capabilities. Moreover, it exploits standard non-invasive security mechanisms and is perfectly tailored for an IIoT scenario as described in the following.

## III. MQTT-based framework for Production Systems

In this section, we introduce a general organization of the production resources of a manufacturing company, as well as a commonly used communication framework. The notation used in all the following figures is described in Table I. In Fig. 1, we depict a company, supposed to have one or more production sites ($Site_i$) and headquarters. Each site $i$ may include one or more production lines ($Line_{ij}$), formed of different machines ($M_{ij}^k$).

PLCs, SCADAs, and distributed sensing systems, formed of IoT devices and organized as wireless sensor networks,

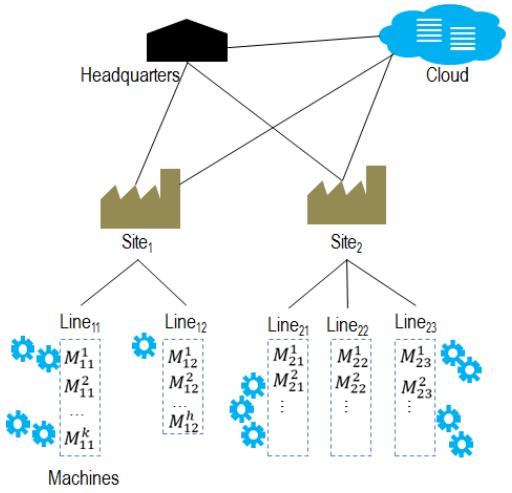



Fig. 1. General organization of machines in a manufacturing company.

TABLE I
DESCRIPTION OF THE NOTATION USED IN THE CONSIDERED COMMUNICATION FRAMEWORKS.

| Symbol | Description |
|---|---|
| $Site_i$ | Industrial Site $i$ |
| $Line_{ij}$ | Production Line $j$ of site $i$ |
| $M_{ij}^k$ | $k^{th}$ Machine of line $j$ of site $i$ |
| $M_{ij}^{mk}$ | $k^{th}$ Machine of manufacturer $m$ in line $j$ in site $i$ |
| $B_{ij}^m$ | Broker of manufacturer $m$ in line $j$ in site $i$ |
| $MB_m$ | Manufacturer Broker aggreg. all machines of manufacturer $m$ |
| $LB_{ij}$ | Line Broker aggreg. all machines of line $j$ in site $i$ |
| $SB_i$ | Site Broker aggreg. all machines of site $i$ |

are used to monitor and control manufacturing machines. These systems are interconnected to per-line and per-site remote controllers. These may also be interconnected to the headquarters and/or to an external Cloud system to enable cross-site monitoring and control. All these devices and nodes form a complex IIoT network, which, in some cases, should be also interconnected to external entities, e.g., to the machine manufacturers or to third-party maintenance companies, for monitoring purposes.

With respect to the communication technology, either proprietary ad-hoc solutions or standard protocols are currently used in Industry 4.0. Obviously, the latter option has advantages of interoperability, scalability, and long-term maintainability. The most promising standard communication paradigm is the one based on the pub/sub scheme, wherein data are published to a server by producers (sources) and relayed to different consumers (receivers). In the industrial IoT scenario, MQTT is becoming a *de facto* standard when adopting the pub/sub paradigm, which is used for either collecting data from different data sources, such as PLCs and sensors, or for sending commands from controllers to the target devices. The pub/sub mechanism is usually implemented through a central node that receives the published data and selectively relays them to the proper subscribers. According to the MQTT notation, we denote this node as a *broker*.

Generally, we can suppose that machines of the same manufacturer may share the same communication system based on a single broker. This leads to having one or more brokers per line, and thus various brokers may be present on one site.

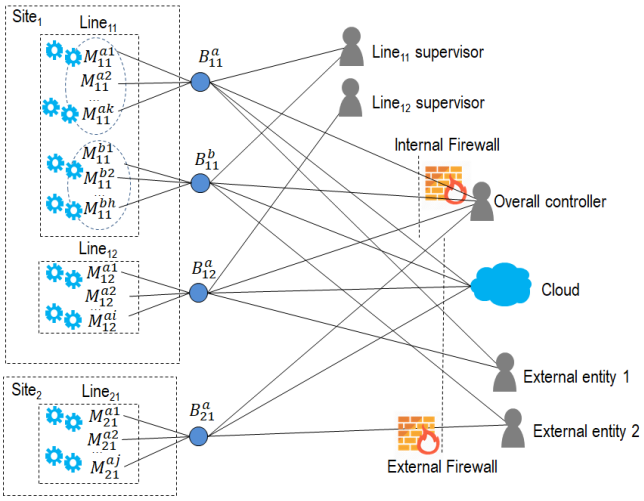



Fig. 2. A general IIoT MQTT-based framework.

In Fig. 2, we show, on the left side, some production lines with their machines and one or more broker per line ($B_{ij}^a$, $B_{ij}^b$, etc.). On the right side, we depict entities that may interact with the production lines via those brokers. In general, at least a per-line control system is present, together with a possible per-site control system (not in the figure), and an overall control system. Data may also be collected on an external Cloud system or by third parties, e.g., to outsource maintenance.

In order to provide a suitable level of protection and of isolation of the different information flows, a proper dynamic authentication and authorization system must be implemented. Unfortunately, this is complicated by the high number of possible M2M interactions to be considered separately. Moreover, at a lower level, those interactions have to be explicitly configured onto intermediate network nodes (NATs and firewalls). Hence, the overall framework may result into a very complex network, lacking in scalability and proving difficult to configure and administrate. These are the main motivations that led us to propose the novel pub/sub framework described in the following, and to apply it to the above described IIoT scenario.

## IV. PROPOSED MQTT-BASED MESSAGING FRAMEWORK FOR IIOT

As discussed in Sec. III, an IIoT system based on the classic pub/sub paradigm may lead to a complex, undependable and non-scalable framework, due to the high number of M2M interactions that have to be separately managed at both the application level, for properly configuring authentication and authorization functions, and the network level, for selectively enabling the traversal of intermediate nodes like NATs and firewalls. In this section, we try to solve these issues by proposing a new, flexible, and scalable framework, still based on the pub/sub paradigm, enhanced by a multi-stage broker system. Although this approach could be mapped on different types of pub/sub communication systems, for the sake of

simplicity and of interoperability, the following description will consider the use of MQTT (as M2M protocol).

Starting from the original framework depicted in Fig. 2, which suffers from scalability and security problems, our idea is to try to group several client-to-broker interactions and map them into a new multi-stage framework, where different stages of brokers are used to enable clients to separately access different groups of devices.

The resulting framework is depicted in Fig. 3. On the left side of the figure the production machines are grouped by line. By supposing that different machine manufacturers connect their machines to (different) manufacturer brokers (regardless of whether they use the same pub/sub protocol or not), the machines are also grouped by manufacturer.

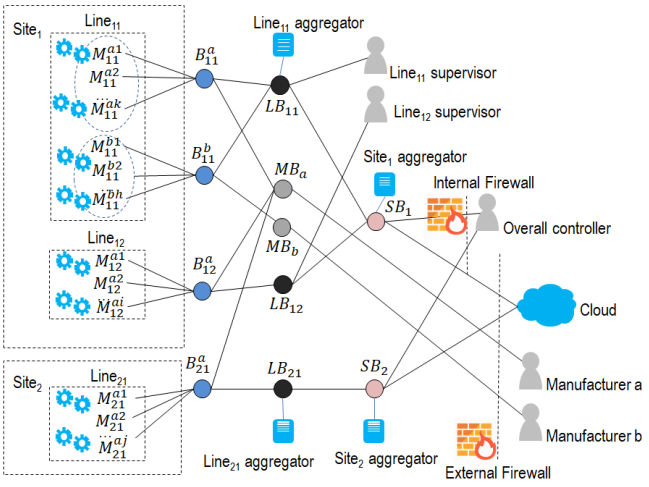



Fig. 3. Proposed framework showing different tiers of brokers aggregating data for different entities.

Rather than connecting the controllers and data consumers (represented at the right side) directly to the brokers that manage the devices/topics they are interested into, a second level of brokers is introduced, which groups machines according to different access classes. For example there could be:

i. brokers associated to production lines, with one broker for each line (brokers $LB_{ij}$ in Fig. 3),
ii. brokers associated to different machine manufacturers, one broker per manufacturer ($MB_m$ in Fig. 3).

Of course, other types of classes and corresponding brokers can be added.

There could be some *data aggregation* functions, associated to each broker, that are in charge of processing input data and re-publishing them processed and (possibly) aggregated according to the profiles of some consumers. For example, per-line data can be aggregated providing overall measurements, mean values, new alarms, as a function of a given sequence of events, etc.

Similarly, a third stage of brokers can be introduced for further grouping different types of M2M interactions and for providing enhanced aggregation functions. For example, in the case of a multiple-site factory, per-site brokers can be introduced (brokers $SB_i$ in Fig. 3).

The resulting communication framework has the following advantages:

1) it groups M2M interactions, allowing for a great simplification of client-to-broker relations that have to be taken into account for the authentication and authorization function, as each user client interacts only with second and higher stage brokers;
2) it reduces the number of network level relations that have to be considered in NAT and firewall configurations (from as many relations as the number of brokers that manage the devices/topics, to as many relations as the number of second level brokers, i.e., access classes), simplifying network administration, and drastically reducing the chance of vulnerabilities due to misconfiguration of network-level intermediate boxes;
3) it is more scalable, since it reduces the total number of flows that each broker has to manage. Assuming $c$ entities on the right of the figures, $d$ devices per broker, $m$ brokers connected to the devices (first layer), $m'$ second-level brokers ($m' < c$ and $m' < m < d$) and that no third-level brokers are present, in the traditional scenario (Fig. 2), each broker has to manage $c \times d$ flows, while in the proposed framework (Fig. 3) the brokers connected to the devices (first layer) have to manage $m' \times d$ flows and the second-level brokers have to manage $c \times m$ flows, hence fewer information flows per broker given the aforementioned inequalities;
4) it simplifies the implementation of new data processing functions, fully integrated within the multistage pub/sub framework; indeed, second stage brokers can feature and gather data processing functions.

A simple way for implementing the framework described above is by using MQTT as messaging protocol, and the "bridging" mechanism offered by most of the current MQTT broker implementations as multi-hop relay mechanism. A broker is configured as a "bridge" when, in addition to its normal broker functions, it acts like a client by: i) subscribing to some pre-configured topics on another broker (or multiple brokers), ii) relaying *PUBLISH* requests targeted to those topics to the selected broker(s). As a result, the given topics are shared among two (or more) brokers. This, in turn, enables i) the creation of clusters of brokers, with the same topic shared amongst clients connected to different brokers of the cluster, and ii) the creation of an MQTT proxying mechanism, with clients subscribing to a topic via a broker different from the one the publishers send messages to.

However, since this mechanism is currently based on manual and static (platform-specific) configuration, it lacks in flexibility and scalability. Moreover, this may possibly lead to security flaws due to erroneous, or not updated, bridging configurations. To overcome these issues and to create a fully flexible and scalable broker network, a new dynamic bridging mechanism and novel multi-stage authentication and authorization functions are desirable. They are described in the following Sec. IV-A and Sec. IV-B, respectively.

### A. Dynamic Broker Bridging

In this subsection, we detail our proposal for transforming the static bridging into a dynamic and flexible multi-broker routing. The idea is to let a client subscribe or publish messages into topics handled by brokers different from the broker to which the client is connected to. To allow the client to specify the remote broker that is in charge of handling the topic, in which the client is interested, at least two different approaches could be used:

1) the definition of a new MQTT header field that carries this information;
2) the exploitation of the current MQTT protocol at the application level, possibly overloading some header fields.

Although the former approach may be simpler, the latter has the advantage that it does not require any modification of the MQTT protocol specification and implementations. For this reason, we decided to follow the second approach. In particular, we use the *Topic Name* string to encode both the identifier of a remote broker and the real topic name. As a result, the new *Topic Name* string is transformed into a URL-like *topic-at-broker* field. Different formats could be used to encode this information. A simple solution could be using the '@' character as a separator between the topic name and the identifier of the broker in charge of this topic. Thus, the resulting MQTT *Topic Name* is constructed as follows:

$$Topic_{new} = Topic_{actual} \ @ \ Broker \tag{1}$$

When a broker receives a *SUBSCRIBE* or *UNSUBSCRIBE* request, in which the *Topic Name* value (or one of the topic names, in case a list of topics has been received) corresponds to the template $Topic_x$ @ $Broker_y$, the broker has to perform the following actions:

1) subscribe/unsubscribe the sender of the request to/from topic $Topic_x$;
2) send a subscription or a subscription cancellation request to broker $Broker_y$ with *Topic Name* equal to $Topic_x$.

When a broker receives a publication request, in which *Topic Name* value corresponds to the template $Topic_x$ @ $Broker_y$, it forwards the message to $Broker_y$ setting the *Topic Name* value to $Topic_x$.

The proposed mechanism can be further extended by allowing multi-broker topics, like $Topic_x$ @ $Broker_n$ @ $Broker_{n-1}$ @ ... @ $Broker_1$, which will allow a message to be forwarded through $(n-1)$ intermediate brokers until it reaches the destination broker $Broker_n$. An example of this case is depicted in Fig. 4. In the example, client $C_1$ subscribes to topic $T_1$ on broker $B_3$. However, rather than sending the request directly to $B_3$, it exploits the sequence of intermediate brokers $B_1$ and $B_2$. Then client $C_2$ publishes a message $M_1$ in topic $T_1$ on broker $B_3$ using $B_4$ as an intermediate broker toward $B_3$. Hereafter, we describe in detail the steps of these subscription and publication procedures:

1) Client $C_1$ starts by sending to $B_1$ a *SUBSCRIBE* request with *Topic Name* equal to $T_1$ @ $B_3$ @ $B_2$.
2) When broker $B_1$ receives the request, it follows the following procedure: i) it saves the subscription of $C_1$ to topic $T_1$ @ $B_3$ @ $B_2$ in its database, and ii) it forwards the request to $B_2$ changing *Topic Name* value to $T_1$ @ $B_3$.
3) When broker $B_2$ receives this request, it: i) stores a record that $B_1$ has subscribed to topic $T_1$ @ $B_3$, ii) sends a request to broker $B_3$ to subscribe to topic $T_1$.
4) Broker $B_3$ receives the new request and processes it as a standard subscription request.
5) When client $C_2$ wants to publish a message $M_1$ to topic $T_1$ on broker $B_3$, it sends a *PUBLISH* request to broker $B_4$, setting *Topic Name* value to $T_1$ @ $B_3$.
6) Broker $B_4$ receives the request and forwards it to broker $B_3$, changing *Topic Name* value to $T_1$.
7) $B_3$ receives the request and processes it as a standard publication request. In particular: i) it retrieves the list of subscribers to topic $T_1$; in this example the list includes (only) $B_2$, ii) it sends message $M_1$ to the subscribers ($B_2$).
8) When $B_2$ receives the *PUBLISH* message, it retrieves the list of clients subscribed to that topic ($T_1$), together with the original *Topic Name* values, and sends the new request to those clients using the proper *Topic Name*. In the provided example it sends the *PUBLISH* message to $B_1$ with topic name equal to $T_1$ @ $B_3$.
9) Similarly, when broker $B_1$ receives the request, it retrieves the list of subscribers and *Topic Name* values, and forwards the message accordingly. In the example, it sends message $M_1$ to client $C_1$ with *Topic Name* equal to $T_1$ @ $B_3$ @ $B_2$.

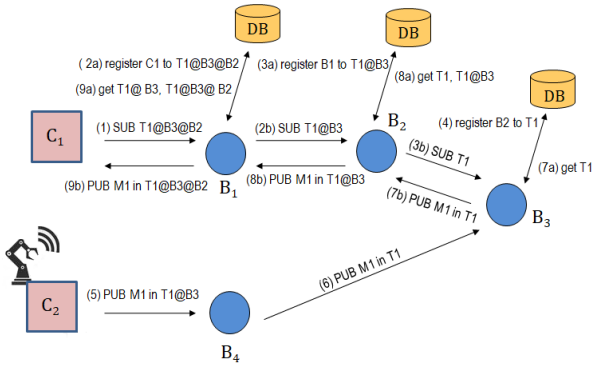



Fig. 4. An example of multi-broker communication.

Generally, broker bridging can be used for: i) security reasons, for example when only selected brokers are allowed to forward messages in and out of a network domain, ii) inter-domain routing, for creating a hierarchy of brokers capable of interconnecting two or more different network domains, iii) scalability, for creating a larger and scalable pub/sub distribution framework. In this context, dynamic broker bridging can be useful to dynamically let the clients route messages through one or more intermediate brokers, without requiring any static pre-configuration of the brokers themselves.

The reliability of the broker chain could be an issue, yet current MQTT implementations can be easily clustered on different virtual machines on the same physical hardware [16], thus providing some sort of fault tolerance should overloads or failures take place. In such situations, a load balancer can

be exploited as a unique entry point for all communications, thus creating a single logical broker from the points of view of clients and fostering vertical scalability in addition to reliability.

### B. Authentication and Authorization

Regarding authentication and authorization, MQTT supports both basic authentication and enhanced authentication, exploiting the payload of *CONNECT* messages in MQTT v3.1.1 [22] or using the *AUTH* packets in MQTT v5.0 [4], respectively. These methods are quite limited and currently they are going to be further extended by the IETF ACE working group, by adding third party authentication and authorization delegation. This is achieved by porting the well-known OAuth 2.0 framework into an IoT constrained environment [23], and mainly assuming that access to resources takes place using CoAP protected by DTLS. An MQTT-TLS profile of ACE [24] has also been specified to enable authorization in an MQTT-based pub/sub system. It uses an authentication server that releases access tokens to the clients and relies on TLS for providing confidentiality and integrity. Considering the constraints normally featured by IoT nodes, exploiting a trusted third party, e.g., an authorization server, is useful to decrease the load of the nodes. This approach, used in our proposal as well, eliminates the devices' need to be acquainted with all the other participants of the network.

In this section, we further extend the authentication and authorization schemes introduced in [23] and [24], by considering the scenario of multiple brokers previously described, and we define a new third party-based authentication and authorization mechanism, encompassing the presence of a single trusted Authentication and Authorization Server (AS). In all considered cases, the AS is in charge of the clients' authentication and of the generation of authorization/access tokens, used by the clients to prove rights to subscribe to a topic or to publish a message in a topic on one or more brokers. Depending on the type of used scheme, the access token could be a simple opaque token (explicitly verified by the broker by interacting with the AS), or a fully self-contained cryptographic token, allowing the broker to verify directly the access rights.

Considering authentication/authorization with multiple MQTT brokers, different cases could be considered:

- *client-to-all-brokers* authorization - The AS, after having successfully authenticated the client, releases to the client different access tokens, one token for each broker through which the client wants to route the request. The client includes all these tokens in the request sent to the first broker. The tokens are then used and verified hop-by-hop by all brokers the request is routed through.
- *hop-by-hop* authorization - The AS releases an access token valid only for the first broker the client will communicate with. All other brokers relaying the request have to explicitly communicate with the AS in order to obtain a new token to forward the message further.

In the first approach, the access is directly granted by the AS to the client, and it has the advantages of minimizing the interactions with the AS and the number of exchanged messages. Conversely, in the second approach, the grant is provided only for single interactions. In this case, the intermediate brokers implement a sort of authorization delegation chain, and, although the interactions with the AS increase, the amount of authorization data to be included in each request reduces, resulting in a smaller packet size. Hereafter, these two approaches are separately described.

Note that, when the number of brokers reduces to 1, both approaches are the same, and they also correspond to the standard IETF ACE for MQTT [23] [24]. This consideration will help us in Sec. V to compare the performance of our solution to the IETF standard one.

### C. Client-to-All-Brokers Authorization

In Fig. 5 an example of this scenario is depicted. Client $C_1$ wants to subscribe to a topic $T_1$ on broker $B_N$ or to publish a message $M_1$ into topic $T_1$ on broker $B_N$. Client $C_1$ wants to route this request through the sequence of intermediate brokers $B_1$, $B_2$, ..., $B_{N\text{-}1}$, before reaching $B_N$.

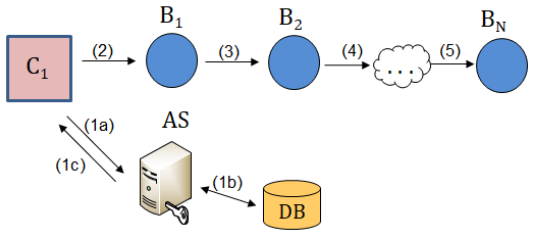



Fig. 5. Client-to-all-brokers authorization scheme.

The complete authentication and authorization procedure is as follows:

1) Before sending the request to the first broker, $C_1$ authenticates with the $AS$ and obtains a set of valid access tokens $authz_1$, $authz_2$, ..., $authz_N$, for the corresponding sequence of brokers, granting the access to a given topic $T_1$. The request may include the list of brokers, one or more topics (in this example $T_1$), and the action to be performed (pub/sub). The request is authenticated through $C_1$'s credentials (e.g., username and password), or by using a different authentication scheme, e.g., based on $C_1$'s secret key or private/public keys.
   If $C_1$'s request is successfully authenticated, the $AS$ matches the request parameters to the authorization policy stored in the policy database. In case of success, the $AS$ returns a list of access tokens $authz_1$, $authz_2$, ..., $authz_N$, for the corresponding brokers $B_1$, $B_2$, ..., $B_N$. A standard way for obtaining the access token can be the one described in [25] for OAuth2.0, or the one currently proposed in [23] for constrained environments.
2) $C_1$ sends the request to $B_1$, where the *Topic Name* value is set to $T_1$ @ $B_N$ @ $B_{N\text{-}1}$ @ ... @ $B_2$ (see Sec. IV-A). The request contains the access tokens obtained from the AS as well. How the access tokens can be attached to the request is discussed in Sec. IV-E.

3) $B_1$ receives and processes the request, checking if the request contains a valid token $authz_1$ for this broker. If this is the case, the broker removes it from the list. The request message is then processed according to the procedure described in Sec. IV.
4) The same operations are performed by all other brokers until $B_{N\text{-}1}$.
5) When the request reaches destination broker $B_N$, it verifies the last access token $authz_N$ and finally processes the request as a standard MQTT broker.

### D. Hop-by-Hop Authorization

In Fig. 6 an example of this scenario is shown. Like in the previous case, client $C_1$ wants to subscribe to topic $T_1$ on broker $B_N$ or to publish a message $M_1$ into topic $T_1$ on that broker, through the sequence of intermediate brokers $B_1$, $B_2$, ..., $B_{N\text{-}1}$.

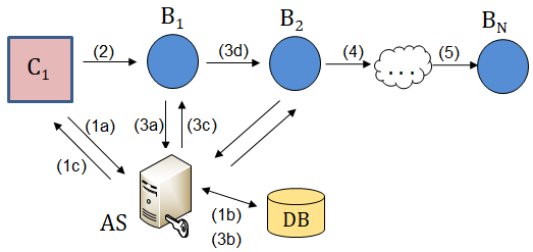



Fig. 6. Hop-by-hop authorization scheme.

Differently from the previous case, the client obtains an access token valid only for the first broker $B_1$, which is in charge of requesting a new access token to relay the request message to broker $B_2$. The same must happen for the following brokers until $B_{N\text{-}1}$, as hereafter described:

1) Client $C_1$ authenticates with the $AS$ and requests an access token for granting the access to $B_1$ for topic $T_1$ in $B_N$. In case of success, the requested token $authz_1$ is returned to $C_1$.
2) $C_1$ sends the request to $B_1$ with *Topic Name* value equal to $T_1$ @ $B_N$ @ $B_{N\text{-}1}$ @ ... @ $B_2$ (see Sec. IV-A). The request contains also the access token $authz_1$.
3) $B_1$ processes the received request and checks if the authorization token is valid. In case, $B_1$ authenticates with AS and requests an access token to forward the request for $T_1$@$B_N$ to $B_2$. After receiving the token $authz_2$, $B_1$ adds $authz_2$ to the request and sends it to $B_2$ (see Sec. IV).
4) The same operations are repeated until reaching $B_{N\text{-}1}$.
5) When the request arrives at broker $B_N$, it verifies the access token $authz_N$ and processes the MQTT request.

### E. Access Token Request and Validation

When requesting an access token, the client sends to the AS a request with the type of desired permissions, in terms of scope and duration. In case of MQTT, the scope may be specified by:

- the target *broker id*;
- a *topic* filter, specifying one or more topic names;
- a *lifetime*, i.e., the expiration time or validity interval;
- the *action* to be performed: *SUBSCRIBE* or *PUBLISH*.

The access token could be either: i) an opaque identifier, or ii) a self-contained and cryptographically verifiable authorization token. In the first case, the target broker asks the AS to validate the token and to return the actual authorization information (client id, broker id, topic name(s), action, expiration time). Instead, in the latter case the token will contain the aforementioned information and either a *signature*, computed through the AS private key, or a *MAC*, computed by means of a symmetric key shared between the AS and the client.

A standard format to encode the token could be CBOR Web Token (CWT) [26]. It is a compact mechanism for representing claims to be transferred between two parties and where data are encoded in the Concise Binary Object Representation (CBOR) [27].

Using MQTT, there are two standard methods for the client to send the access token to the broker. In particular, in MQTT v3.1 the client can use the *username* field in the *CONNECT* packet to carry the token, while in MQTT v5, used in the current IETF proposal [24], the specific packet type AUTH can be employed. However, in both cases the client is requested to provide a valid access token during the client-to-broker connection setup, and not when the request (*SUBSCRIBE* and *PUBLISH*) is actually sent.

This per-connection authorization has the drawback that a broker cannot share the same connection toward a bridged broker to send requests from different clients and for different topics with their own access tokens.

For this reason we explored additional methods to associate an access token to a standard MQTT *SUBSCRIBE* or *PUBLISH* message.

In case of a *SUBSCRIBE* message, a possibility is to exploit the *topic name* field, by encapsulating the access token in place of the actual topic (the real topic name can be also attached or retrieved from the access token when explicitly included). In our implementation (see Sec. V) we called this method as 'topic'-access token mode.

The same method can be used to authorize PUBLISH requests. In addition, in case of a *PUBLISH* request, the *payload* field of the MQTT message can be also exploited, by encapsulating both the access token and the original payload in the *payload* field. In our implementation we called this method as 'payload'-access token mode. Using this latter method in case of *PUBLISH* requests has the advantage of simplifying the implementation and integration in existing MQTT brokers, since the analysis of the topic name and the subscriber lookup can be performed by the broker without any code modification.

In case of a "Client-to-All-Brokers Authorization" scenario, a practical alternative for the client to use a list of access tokens is to let the AS provide only a single token with a multi-recipient audience, including the authorization information for all brokers. In case of cryptographic tokens, they must include either a single signature provided through the private key of the AS, or multiple MAC values computed with the different secret keys shared with the $N$ listed brokers.

## V. Testbed and Experiments

In order to prove the feasibility of the proposed architecture, a small-scale implementation has been carried out and tested.

One of the main requirements of our implementation was the compatibility with current industrial systems; thus, no ad-hoc PLC or SCADA systems are required. Standard off-the-shelf PLCs, namely Siemens Simatic S7-300, have been used instead. In order to make our implementation work with different devices, an IIoT gateway entity, capable of connecting to different types of PLCs, has been included, using an off-the-shelf system (Alleantia ISC[1]). This provides a very large set of standard PLC interfaces and simple interfaces to make queries or receive PLC data, by either accessing outgoing data collected in standard formats (e.g., xls, csv), or using a proper REST API. In our system, the REST API has been exploited to get PLC data in almost real-time. This is done by a HTTP-to-MQTT adapter that reads data from the IIoT Gateway, and publishes them on the first MQTT X-Broker (eXtended MQTT broker, i.e., a broker with the functionalities described in Sec. IV). For implementing such an adapter, although some open-source HTTP/MQTT gateways are available, like Ponte[2], we preferred to develop it in Java language using the Paho MQTT library[3], for the MQTT client side, and standard Java libraries for the HTTP client side. The adapter has been designed to periodically retrieve data from the IIoT Gateway, by using the Alleantia ISC REST API through the HTTP client, and to publish them to the X-Broker. In accordance with our proposed framework, data are then relayed to next stage brokers depending on the subscribed clients (data consumers) and on the enforced security policies. The resulting testbed is depicted in Fig. 7.

If an IIoT gateway with MQTT output or PLCs that natively support MQTT are used, the adapter or the IIoT gateway+adapter cascade can be removed, respectively. To emulate an MQTT-native PLC, a Raspberry Pi with an MQTT client (implemented using the Paho MQTT library) has been also used.

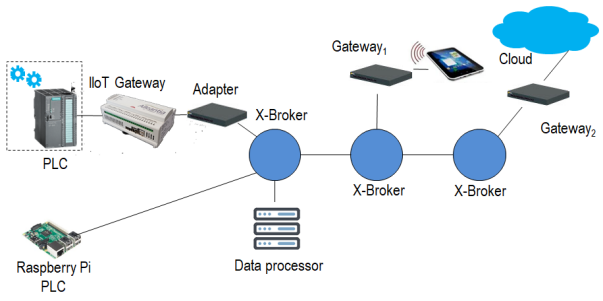



Fig. 7. Implemented testbed.

For the realization of the X-Broker, two different Java-based third-party brokers have been used: the Moquette broker[4] and the Jmqtt broker[5]. They both guarantee high performance thanks to Netty[6] as underlying network framework. With respect to reliability, both Moquette and Jmqtt support MQTT standard Quality of Service (QoS) levels 0, 1, and 2. QoS 0 means that each message is delivered at most once, i.e., best effort delivery. QoS 1 means that each message is delivered at least once. QoS 2 means that each message is delivered exactly once.

Both MQTT types of brokers (Moquette and Jmqtt) have been properly extended as described in Sec. IV, by adding a new bridge module. Two different bridging modes, called 'spiral' and 'straight', have been designed and implemented in both brokers. The two modes are sketched in Fig. 8. When a *SUBSCRIBE* message reaches an X-Broker with a topic name targeting a different broker, this is passed to the bridge module that processes it according to the mechanism described in Sec. IV. The mapping between the requested topic and the client is stored locally, together with the mapping between the incoming and outgoing topic names. Then the topic name is updated, and the message is forwarded to the next broker. This mechanism applies for both bridging modes, spiral and straight (messages 1, 2, and 3 in Fig. 8-a and 8-b). The only difference is that, in the case of straight mode, the subscribers are registered in a table within the bridge module, while, in the case of spiral mode, they are registered in the standard broker table. When a *PUBLISH* message reaches the X-Broker, this message is also passed to the bridge module. In case of spiral bridging mode, the module modifies the message according to the registered topic mapping and passes it back to the broker that processes and forwards it toward the registered subscribers (messages 4, 5, and 6 of Fig. 8-a). On the other hand, in case of straight bridging mode, the message is processed, updated and forwarded to the registered subscribers directly by the bridge module (messages 4 and 5 of Fig. 8-b).

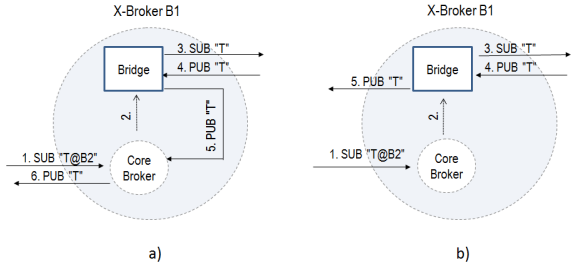



Fig. 8. Bridge module operation modes: a) 'spiral', b) 'straight'.

The two bridge modules have been implemented using two different MQTT libraries (Paho and Jmqtt), leading to two different versions that have been compared. For integrating the bridge module into the two brokers, only few classes had to be slightly modified (*MQTTConnection*, *NewNettyAcceptor*, and *PostOffice* for the Moquette broker, and *ConnectProcessor*, *PublishProcessor*, *SubscribeProcessor*, *BrokerController*, and *BrokerStartup* for the Jmqtt broker). All the implemented software is publicly available open source[7].

[1]https://www.alleantia.com/products/isc-software/
[2]https://www.eclipse.org/ponte/
[3]https://www.eclipse.org/paho/
[4]https://moquette-io.github.io/moquette/
[5]https://github.com/Cicizz/jmqtt
[6]https://netty.io
[7]https://github.com/mqtt-dbb/mqtt-dbb

To provide an on-the-way processing function of the PLC data, a particular MQTT client (the *Data processor*, in Fig. 7), connected to the first broker, has also been added. The processor subscribes for some PLC topics, receives the corresponding data, processes them, and then publishes the resulting data on a new topic on the same broker.

Two different MQTT clients have been added and connected to two different brokers. Both clients subscribe to some topics and present the received data (possibly processed and aggregated) to the users, through two different HTTP-based interfaces, acting as HTTP servers (the MQTT-to-HTTP Gateways 1 and 2 in Fig. 7). In particular, the first entity provides a server-side REST API to client applications (in our implementation a simple mobile app), while the second entity presents aggregated data through a standard web interface, accessible through standard browsers. In order to emulate the client-to-broker networks, the NEMO [28] network emulator has been used between the various clients and their brokers.

We have also carried out a performance evaluation campaign in order to test the applicability of the proposed framework in terms of end-to-end real-timeliness. In particular, we have analyzed and compared the performance of an X-Broker under different functional modes and implementations. All presented results refer to the X-Broker based on Jmqtt, since preliminary tests showed that it exhibited better performance in terms of forwarding delay. The brokers have been run on standard PCs with Intel Core i7 2.70 GHz processor, 16 GB RAM, and Windows 7 64-bit OS.

In Fig. 9 the time spent by a *PUBLISH* message to be properly processed and to cross $n$ X-Brokers is reported versus $n$. The two bridge implementations are compared. In the case of the Paho-based implementation, the three MQTT QoS types (0='at most one', 1='at least one', 2='exactly one'), are shown as well. The results, an average over 50 packets, show a linear increase of time with the number of crossed brokers and an increase of time according to the required QoS.

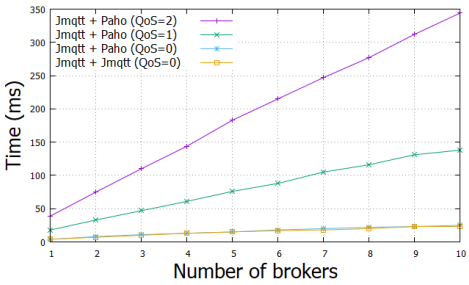



Fig. 9. Comparison of different X-Broker implementations.

In Fig. 10 the two different bridging modes are compared, in terms of forwarding delay, varying the number of crossed brokers ($n$). The broker and the bridge modules are both implemented in Jmqtt and the considered QoS level is 0. The results, an average over 50 messages, confirm a greater increase of time in the case of the spiral implementation.

Finally, different access token encapsulation methods have been tested. In all cases, a cryptographic access token, signed through a 2048-bit RSA private key of the AS is used. In a

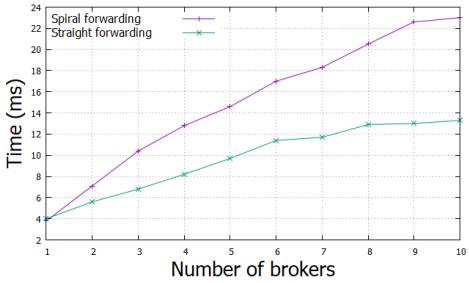



Fig. 10. Comparison of different bridging modes.

first set of tests, the 'topic' method has been used, and the access token is processed and removed by the first broker. In a second set of tests, the 'payload' method has been used, by letting all broker process and explicitly validate the access token. The results shown in Fig. 11, which are an average over 150 messages and consider an implementation through Jmqtt of both the broker and the bridge, with QoS = 0 and spiral forwarding, demonstrate that the overhead introduced by the verification of the access token is still manageable, with an additional delay of 1.2 $ms$ per single broker on average in the case of the 'payload' method.

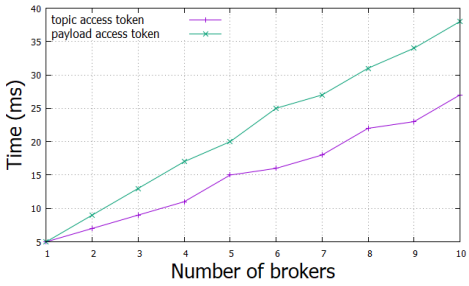



Fig. 11. Comparison of different access token encapsulation modes.

The results in Fig. 11 allow also the comparison of our solution with the standard authentication and authorization approach defined in [23] and [24] for a single broker. In fact, as observed in Sec. IV-B, the proposed multi-broker authentication and authorization mechanism, in case of a single broker corresponds to the standard ACE for MQTT mechanism. Since the delay grows linearly with the number of the brokers, and the delay introduced for processing the authorization token is low, especially in all practical IIoT scenarios with a limited number of broker stages (2 or 3), the overall delay will remain low and of the same order of magnitude of that of the standard one-broker solution.

To summarize, the experimental results have shown that i) the time spent by a *PUBLISH* message to be properly processed and to cross $n$ X-Brokers is linear in $n$ and increases according to the required QoS, as expected; ii) the forwarding delay with the spiral bridging mode is almost double than the delay resulting when the bridging mode is straight; iii) the verification of the access token introduces a small overhead. For applications that may settle for QoS 0, the forwarding overhead is acceptable even with spiral bridging. When QoS 2 is required, straight bridging should be considered, in order

to keep the forwarding overhead small.

## VI. Conclusions

In this paper, we have introduced a novel dynamic multi-broker framework to relay secure and scalable M2M communications based on a pub/sub paradigm. Our MQTT-based framework can help securely managing and isolating the large number of information flows present in complex IIoT deployments. An authorization and authentication mechanism has been designed specifically for the scheme we proposed: all requests contain authorization tokens released by an authorization server, so only authorized entities can subscribe or publish in a particular topic. The solution we proposed has also been implemented practically and evaluated in a small-scale, yet realistic IIoT testbed. In order to assess the performance of the proposed framework in terms of end-to-end real-timeliness, we have tested different functional modes and implementations of its core component, i.e., the X-Broker. In general, QoS requirements determine the X-Broker configuration to use for achieving a small forwarding overhead.

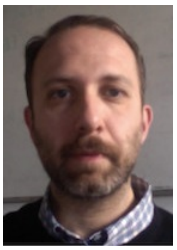
As future works, we plan to evaluate the effectiveness of this scheme against different cyber-physical attacks, concerning mainly authentication and authorization issues between the communicating machines and third parties. Additionally, we intend to put the proposed framework through its paces in the context of challenging real-world IIoT application scenarios, such as predictive maintenance and cloud manufacturing.

## Acknowledgments

The authors would like to thank Mr. Antonio Enrico Buonocore for his help in carefully proofreading the paper. The work of Riccardo Pecori has been supported by the Italian Ministry of University and Research (MUR), in the framework of the PON R&I 2014-2020 “AIM: Attraction and International Mobility” project.

## References

[1] E. Sisinni *et al.*, “Industrial Internet of Things: Challenges, Opportunities, and Directions,” *IEEE Trans. on Ind. Inform.*, vol. 14, no. 11, pp. 4724–4734, Nov 2018.
[2] H. Boyes *et al.*, “The Industrial Internet of Things (IIoT): An analysis framework,” *Computers in Industry*, vol. 101, pp. 1 – 12, 2018.
[3] K. Iwanicki, “A Distributed Systems Perspective on Industrial IoT,” in *IEEE Int. Conf. ICDCS*, July 2018, pp. 1164–1170.
[4] A. Banks *et al.*, “MQTT Version 5.0,” OASIS, Standard, March 2019.
[5] G. Peralta *et al.*, “Fog computing based efficient IoT scheme for the Industry 4.0,” in *2017 IEEE Int. Workshop ECMSM*, 2017, pp. 1–6.
[6] J.-R. Jiang, “An improved cyber-physical systems architecture for Industry 4.0 smart factories,” *Adv. in Mech. Eng.*, vol. 10, no. 6, pp. 1–15, May 2018.
[7] D. Kim, H. Lee, and D. Kim, “Enhanced industrial message protocol for real-time IoT platform,” in *Int. Conf. ICEIC*, Jan 2018, pp. 1–2.
[8] A. Vakaloudis and C. OLeary, “A framework for rapid integration of IoT systems with industrial environments,” in *IEEE WF-IoT*, April 2019, pp. 601–605.
[9] A. Esfahani *et al.*, “A Lightweight Authentication Mechanism for M2M Communications in Industrial IoT Environment,” *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 288–296, Feb 2019.
[10] M. Calabretta, R. Pecori, and L. Veltri, “A Token-based Protocol for Securing MQTT Communications,” in *SoftCOM 2018*, Sep. 2018, pp. 1–6.
[11] M. Calabretta *et al.*, “MQTT-Auth: a Token-based Solution to Endow MQTT with Authentication and Authorization Capabilities,” *J. Comm. Software and Systems*, vol. 14, Dec. 2018.
[12] M. Singh *et al.*, “Secure MQTT for Internet of Things (IoT),” in *Int. Conf. CSNT*, April 2015, pp. 746–751.
[13] A. A. Wardana and R. S. Perdana, “Access Control on Internet of Things based on Publish/Subscribe using Authentication Server and Secure Protocol,” in *Int. Conf. ICITEE*, July 2018, pp. 118–123.
[14] T. Rausch, S. Nastic, and S. Dustdar, “EMMA: Distributed QoS-Aware MQTT Middleware for Edge Computing Applications,” in *Int. Conf. IC2E*, April 2018, pp. 191–197.
[15] V.-N. Pham and E.-N. Huh, “An efficient edge-cloud publish/subscribe model for large-scale iot applications,” in *Proc. 13th Int. Conf. on Ubiquitous Information Management and Communication (IMCOM) 2019*, S. Lee, R. Ismail, and H. Choo, Eds. Cham: Springer International Publishing, 2019, pp. 130–140.
[16] A. E. C. Redondi, A. Arcia-Moret, and P. Manzoni, “Towards a Scaled IoT Pub/Sub Architecture for 5G Networks: the Case of Multiaccess Edge Computing,” in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 436–441.
[17] Y. Protskaya and L. Veltri, “Broker Bridging Mechanism for Providing Anonymity in MQTT,” in *2019 10th Int. Conf. on Networks of the Future (NoF)*, 2019, pp. 110–113.
[18] A. Rizos, D. Bastos, A. Saracino, and F. Martinelli, “Distributed UCON in CoAP and MQTT Protocols,” in *Computer Security*. Cham: Springer International Publishing, 2020, pp. 35–52.
[19] A. La Marra *et al.*, “Improving MQTT by Inclusion of Usage Control,” in *Security, Privacy, and Anonymity in Computation, Communication, and Storage*. Cham: Springer International Publishing, 2017, pp. 545–560.
[20] A. L. Marra, F. Martinelli, P. Mori, and A. Saracino, “Implementing usage control in internet of things: A smart home use case,” in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 1056–1063.
[21] A. Schmitt, F. Carlier, and V. Renault, “Data Exchange with the MQTT Protocol: Dynamic Bridge Approach,” in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, pp. 1–5.
[22] A. Banks and R. Gupta, “MQTT Version 3.1.1,” OASIS, Standard, October 2014.
[23] L. Seitz *et al.*, “Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth),” IETF, Internet-Draft draft-ietf-ace-oauth-authz-24, Mar. 2019.
[24] C. Sengul, A. Kirby, and P. Fremantle, “MQTT-TLS profile of ACE,” IETF, Internet-Draft draft-ietf-ace-mqtt-tls-profile-00, May 2019.
[25] D. Hardt, “The OAuth 2.0 Authorization Framework,” RFC 6749, Oct. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6749.txt
[26] M. Jones *et al.*, “CBOR Web Token (CWT),” RFC 8392, May 2018.
[27] C. Bormann and P. Hoffman, “Concise Binary Object Representation (CBOR),” RFC 7049, Oct. 2013.
[28] L. Veltri *et al.*, “NEMO: A flexible and highly scalable Network EMulatOr,” *SoftwareX*, vol. 10, p. 100248, 2019.



**Michele Amoretti** (S’01–M’06–SM’19) received his PhD in Information Technologies in 2006 from the University of Parma, Parma, Italy. He is Associate Professor of Computer Engineering at the University of Parma. In 2013, he was a Visiting Researcher at LIG Lab, in Grenoble, France. He authored or co-authored over 100 research papers in refereed international journals, conference proceedings, and books. He serves as *Associate Editor* for the journals: IEEE Trans. on Quantum Engineering and International Journal of Distributed Sensor Networks. He is involved in the Quantum Information Science (QIS) research and teaching initiative at the University of Parma, where he leads the Quantum Software research unit. He is the CINI Consortium delegate in the CEN-CENELEC Focus Group on Quantum Technologies. His current research interests are mainly in High Performance Computing, Quantum Computing, and the Internet of Things.

**Riccardo Pecori** received his Ph.D. in Information Technologies from University of Parma in 2011. He has been an Assistant Professor of Computer Engineering at eCampus University, Italy, since 2015 to 2019, and at University of Sannio, Italy, since August 2019 until now. His main research interests regard network security, Internet of Things, machine and deep learning applications, analysis of complex systems as well as educational Big Data mining. He has been leading organizer and program chair of computer science-related special sessions and workshops at international conferences and has published more than 40 papers in refereed international journals and conferences. He is currently in the editorial board of Future Generation Computer Systems and SoftwareX.

**Yanina Protskaya** got her master degree in System Analysis, Control, and Processing of Information from the Belarusian-Russian University, Mogilev, Belarus in 2016, and her Ph.D. in Information Technologies from the University of Parma, Italy, in 2020. Her main research interests regard security in the Internet of Things and novel anonymity protocols. She currently works as Analyst Programmer at Maps Group in Parma, Italy.

**Luca Veltri** received his Master degree in Telecommunication Engineering and the Ph.D. degree in Information and Telecommunication Engineering from University of Rome “La Sapienza” in 1994 and 1999, respectively. From 2002 he is Assistant Professor at University of Parma, where he teaches classes in Network Security and Communication Networks. From 1999 to 2002 he has been with CoRiTeL, a research institute funded by Ericsson Telecomunicazioni, where he leaded research projects in networking and multimedia communications. He has participated in several research projects funded by the European Commission by the European Space Agency, and by the Italian Ministry of University and Research. His current research interests include Internet of Things, Network Security, and Cyber Security.

**Francesco Zanichelli** received the Dr.Eng. degree in Electronic Engineering at the University of Bologna where he also obtained a research grant from IBM Italy in the area of industrial robotics. Later he spent a research period at the University of Florida to work on mobile robotics and in 1994 he received his Ph.D. degree in Information Technologies at the University of Parma. He is currently an Associate Professor with the Department of Engineering and Architecture of the University of Parma. Current research interests are related to distributed systems and to middleware for service oriented peer-to-peer systems, to cloud platforms for the Internet-of-Things and predictive maintenance as well as to blockchain-based security applications. The research activity, resulting into over 80 international journal and conference papers, has been carried out in the framework of several regional, national and international research programmes, such those funded by region Emilia-Romagna, the national Research Ministry, NATO and the European Commission.