Information Systems 62 (2016) 29–41

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

AUPS: An Open Source AUthenticated Publish/Subscribe system for the Internet of Things

Alessandra Rizzardi[a], Sabrina Sicari[a,*], Daniele Miorandi[b], Alberto Coen-Porisini[a]

[a] Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, via Mazzini 5, 21100 Varese, Italy
[b] U-Hopper, via A. da Trento 8/2, 38122 Trento, Italy

ARTICLE INFO

Article history:
Received 26 May 2016
Accepted 27 May 2016
Recommended by: D. Shasha
Available online 4 June 2016

Keywords:
Internet of Things
Security
Publish & Subscribe
MQTT
Middleware
Prototype

ABSTRACT

The arising of the Internet of Things (IoT) is enabling new service provisioning paradigms, able to leverage heterogeneous devices and communication technologies. Efficient and secure communication mechanisms represent a key enabler for the wider adoption and diffusion of IoT systems. One of the most widely employed protocols in IoT and machine-to-machine communications is the Message Queue Telemetry Transport (MQTT), a lightweight publish/subscribe messaging protocol designed for working with constrained devices. In MQTT messages are assigned to a specific topic to which users can subscribe. MQTT presents limited security support. In this paper we present a secure publish/subscribe system extending MQTT by means of a key management framework and a policy enforcement one. In this way the flow of information in MQTT-powered IoT systems can be flexibly controlled by means of flexible policies. The solution presented is released as open source under Apache v.2 license.

1. Introduction

The Internet of Things (IoT) represents an emerging paradigm for networking and service provisioning, embracing heterogeneous devices (e.g., wireless sensor networks, RFIDs, actuators) and communication technologies in order to acquire data from the physical realm and process them in order to cooperatively provide useful services for the interested users [1]. Examples of IoT scenarios include health equipments for patients monitoring, connected cars in vehicular networks, surveillance devices, wearable sensors, smart home systems, and so on. In such contexts, it is fundamental to define how the involved "things" could efficiently communicate and exchange information among themselves and with remote servers. One key challenge relates to the amount of data generated, which poses scalability issues. Furthermore, some of such data may represent sensitive or personally identifiable information. What emerges is that there are significant issues to be addressed in order to efficiently and securely manage IoT systems. Such problems are related to: (i) the management of connections among the IoT system and the data sources (e.g., the devices which acquire information from the IoT environment), which could be affected by resources constraints in terms of energy and storage capacity (ii) the possibility, for the users, to control the distribution of their sensitive information through IoT connections as well as effective authentication and authorization mechanisms both for users and devices in order to prevent malicious access to resources [2].

* Corresponding author.
E-mail addresses: alessandra.rizzardi@uninsubria.it (A. Rizzardi), sabrina.sicari@uninsubria.it (S. Sicari), daniele.miorandi@u-hopper.com (D. Miorandi), alberto.coenporisini@uninsubria.it (A. Coen-Porisini).

http://dx.doi.org/10.1016/j.is.2016.05.004
0306-4379/© 2016 Elsevier Ltd. All rights reserved.

As regard the first issue, several existing application-level protocols for IoT and Machine-To-Machine (M2M) [3,4] systems have been designed. Such protocols are typically conceived to introduce little overhead and to minimize battery consumption, as well as to perform well in the presence of many short messages. The most widely adopted communication protocols for such fields are MQTT (Message Queue Telemetry Transport) [5] and CoAP (Constrained Application Protocol) [6], which are based on TCP and UDP, respectively. In our work, we focus on MQTT due to its maturity, stability and the fact that, after the recent adoption by the OASIS Consortium as official standard,[1] it is likely to become the de facto standard for IoT.

MQTT is a lightweight event- and message-oriented protocol, which allows the devices to asynchronously communicate across constrained networks to reach remote systems, as happens in a typical IoT/M2M scenario. MQTT is based on a publish/subscribe interaction pattern. In particular, MQTT has been implemented for easily connecting the "things" to the web and support unreliable networks with small bandwidth and high latency. This protocol employs a client–server pattern in which the server part is represented by a central broker that acts as intermediary among the clients (i.e., the entities that produce and consume the messages). All the communications among server and clients happen via a publish/subscribe mechanism, based on the topic concept. A topic is a mean for representing the resources (i.e., the information) exchanged within the system. Topics are used by clients for publishing messages and for subscribing to the updates from other clients. In Section 3 we analyze in depth MQTT features and functionalities and our motivations to employ such a protocol in the proposed IoT architecture. In particular, the actual version of MQTT (v 3.1.1) does not natively support neither mutual authentication mechanisms nor techniques able to guarantee the integrity and the confidentiality of the transmitted information.

Regarding the second issue, adequate mechanisms should be defined in order to control the flow of information and to enforce proper policies implementing specific rules for the management of resources and for handling users preferences. Such mechanisms should be expressive and flexible enough to support the wide range of technologies acting in IoT infrastructures and the various application domains where users and devices could operate. The aforementioned policies concern security requirements, in order to deal with different violation attempts, but also data quality aspects. More in detail, users should be aware of the levels of security and data quality of the information they receive by or transmit to the IoT system, in order to be able to filter them on the basis of personal (or, alternatively, application-dependent) preferences. As far as security levels are concerned, we consider four specific requirements: (i) data confidentiality; (ii) data integrity; (iii) privacy of the data sources; (iv) robustness of the authentication/authorization mechanisms adopted by the data sources. Concerning data quality requirements, we evaluate (i) data accuracy; (ii) data precision; (iii) information timeliness; (iv) information completeness [7].

Summarizing, in this paper we propose a new secure MQTT mechanism, named AUPS (AUthenticated Publish & Subscribe), which has been integrated in a flexible and cross-domain IoT architecture, starting from the Networked Smart Objects (NOS) middleware defined in [7,8]. NOSs are able to distributedly manage heterogeneous sources and evaluate the security and data quality of the information, in order to satisfy users' requirements and provide a lightweight and secure information exchange process. In such a system, AUPS has been further integrated with a policy enforcement mechanism, which guarantees the authentication and authorization of data sources via MQTT. AUPS is openly released under Apache v.2 license.[2]

The paper is organized as follows. Section 2 reviews the state of the art in terms of access control solutions and policy enforcement mechanisms for distributed networked systems. Section 3 describes the proposed IoT architecture, along with the adopted MQTT protocol and the proposed policy enforcement framework. Section 4 presents the integration between MQTT protocol and the enforcement mechanisms, in order to deal with security issues in the investigated context; its robustness is evaluated against possible violation attempts. Section 5 analyses a prototypical implementation of the proposed solution and presents performance evaluation results. Finally, Section 6 concludes the paper and discusses directions for future extensions and enhancements.

2. Related works

Before starting to design and develop our solution, a deep analysis of the state of the art has been carried out, with reference to both access control aspects in distributed systems as well as to the existing enforcement mechanisms. We remark that policies are operating rules which need to be enforced for the purpose of maintaining order, security, and consistency on data. A policy enforcement mechanism ensures that system operations can be performed only if they comply with the underlying security policies, typical operations be access (read or write) to resources. While security is widely acknowledged to be one of the major challenges for the wide adoption and diffusion of IoT systems [2], scientific literature on the topic is rather scarce.

For example, [9] focuses on the definition of a simulation environment supporting various policy languages, such as WS-Policy (Web Services-Policy) and XACML (eXtensible Access Control Markup Language), adopted in different systems. The final goal is to allow cross-domain policy enforcement. Note that, before applying policies across domain boundaries, it is desirable to know which policies can be supported by other domains, which are partially supported, and which are not supported. For this purpose a semantic model mapping and translation for policy enforcement across

[1] https://www.oasis-open.org/committees/mqtt.

[2] https://bitbucket.org/alessandrarizzardi/nos.

domain boundaries is defined by means of the Web Ontology Language (OWL). A configurable middle-level component is provided for the mapping process among different domains. Our work does not only provide an uniform policy representation suitable for different application domains, but also integrate in a uniform platform the management of the heterogeneous entities (i.e., technologies and devices) involved in IoT, thus not limiting the target scenario to traditional infrastructures.

In [10] a novel access control framework, named Policy Machine (PM), is proposed. It is composed by the following basic entities: authorized users, objects, system operations, and processes. Objects specify system resources (e.g., records, files, e-mails) which are controlled under one or more policies; operations identify the actions that can be performed on the contents of objects (e.g., read, write, delete); finally, users submit access requests through processes. Policies are grouped in classes according to their attributes and, therefore, an object may be protected under more than one policy class, and, similarly, a user may belong to more than one policy class. In such a way, PM is a general purpose protection machine, since it is able to configure many types of access control policies, and it is independent from the different operating systems and applications; users need to login only to PM in order to interact with the secure framework. While relevant to our work in terms of conceptual design, the PM represents just an abstraction that cannot be implemented directly in a working framework.

Hence, [11,12] introduce a semantic web framework and a meta-control model to orchestrate policy reasoning with identification and access control of information sources. In fact, in open domains, enforcing context-sensitive policies requires the ability to opportunistically interleave policy reasoning with the dynamic identification, selection, and access of relevant sources of contextual information. Each entity (i.e., user, sensor, application or organization) relies on one or more Policy Enforcing Agents responsible for enforcing relevant policies in response to incoming requests. The framework is applicable to a number of domains where policy reasoning requires the automatic discovery and access of external sources. The limit of such an approach is partially the same of [10], even if an actual Java implementation exists, yet no specific tailoring to an IoT context is present.

Expressing security policies to manage distributed systems is a complex and error-prone task. Because of their complexity and of the different degrees of trust among devices in which code is deployed and executed, it is challenging to make these systems secure. Moreover, policies are hard to understand, often expressed with unfriendly syntax, making it difficult for security administrators and for business analysts to create intelligible specifications. In [13] a Hierarchical Policy Language for Distributed Systems (HiPoLDS) is introduced. HiPoLDS design focuses on decentralized execution environments under the control of multiple stakeholders. It represents policy enforcement through the use of distributed reference monitors, which control the flow of information among services and have the duty to put into action the directives generated by the decision engines. For example, an enforcement engine should be able to add or remove security metadata such as signatures or message authentication codes, encrypt confidential information, or decrypt it when required. Ref. [13] does not specify how the distributed system behaves and manages policy reconfigurations (e.g., if a reboot is required).

The authors of [14] consider that the application logic, embodied in the system components, should be separated from security execution policies. Therefore, they propose an infrastructure which can enable policy, representing high-level (i.e., user) or systems entities, able to drive the system functionalities in a distributed environment. To this end, a middleware is introduced, able to support a secure and dynamic reconfiguration process, and to provide a policy enforcement mechanism across system components. However, neither a case study nor a working real implementation is provided, which would allow to test its effectiveness and the performance in general.

The work in [15] describes a context-aware access control architecture for the provision of e-services based on an end-to-end web services infrastructure. The proposed architecture is able to control access permissions to distributed Web services through an intermediary server in a completely transparent way both to clients and resources. The access control mechanism is based on RBAC (Role-Based Access Control) model, in order to enable the enforcement of more complex rules and the inclusion of context information in the authorization decisions. Contexts are classified on the basis of the requirements imposed by the provision of e-services to the industrial domain. In particular, the presented security enforcement infrastructure implements the intermediary server by means of six modules: Policy Enforcement Point (PEP), Policy Decision Point (PDP), policy server, authentication, authorization, and context modules. Such an architecture is based entirely on open Web standards: HTTP, XML, SOAP and WSDL. What is missing is again the effective portability of such a mechanism in a more general IoT context, taking into account the data sources management and the users interactions with the system.

In [16] a category-based metamodel for access control in distributed (federated) environments is presented. A framework for the specification and the enforcement of global access control policies that take into account the local policies specified by each member of the federation is described. Such a framework provides mechanisms for specifying heterogeneous local access control policies, for defining policy composition operators, and for using them to define conflict-free access authorization decisions. In this framework, distributed access control policies can be easily specified and manipulated by means of local policy specification mechanisms and definitions of policy composition operators. However, no real application case-studies or implemented tools have been proposed by the authors, thus limiting the contribution to a theoretical approach.

In [17] an architecture for open networks is proposed, aiming to allow “things” with limited or no user interfaces to provide a high level of data security by delegating trust to a trusted third party (i.e., a provisioning server). Such third party helps the device to determine which users, devices or services are authorized to perform a given operation on the device itself in a secure manner. Such a solution uses an existing, open and standardized transport protocol for communication, named Extensible Messaging and Presence Protocol (XMPP), for guaranteeing interoperability and scalability. XMPP supports the most commonly communication patterns necessary for Internet of Things, such as request/response, asynchronous messaging and publish/subscribe. It is based on message brokers to solve the security issues concerning user identities, enforcing secure user authentication, and message authorization. The architecture aims for zero-configuration for operators and manufacturers, without compromising security or ease-of-use for end-users. The architecture is also scalable and can be used both in local environments such as cars, homes, offices, buildings, industry plants, with local provisioning servers and local message brokers, as well as in global environments, with global provisioning servers connected to global message brokers. An implementation of the provisioning server is also provided. Note that the author has chosen to use XMPP protocol rather than MQTT, as explained in [18]. However, the scalability claimed by the authors is not clear, since most of the operations are brought to external entities and devices seem unaware about what they are or are not allowed to do. Moreover, no performance analysis is provided to verify the overhead on devices of such a solution.

A scheme for the asynchronous discovering of topics in distributed publish/subscribe settings, based on Java Message Service, WS-Eventing and WS-Notification infrastructures, is presented in [19]. Every interaction within the system is secured and requires the presence of credentials before any actions can take place. The created topic advertisement is itself secured by encrypting the advertisement with a symmetric key and by securing this advertisement key with the creator's public key. In this work, no platform is defined, able to assess the behavior of the presented mechanism in a wide IoT scenario, with a proper threat model.

Ref. [20] aims to demonstrate that a standardized federated, dynamic, user-directed authentication and authorization model can be adapted from the web to be used in IoT, while preserving privacy for information and devices. The authors explore the use of OAuth, built on top of HTTPs, for IoT systems that instead use the lightweight MQTT 3.1 protocol. In particular, they use OAuth 2.0 to enable access control to information distributed via MQTT. What emerges from this work is that some issues still remain open, for example how to allow the re-use and the integration of such a mechanism with the IoT devices and the standard available protocols.

The enforcement solution presented in [21] is based on a Model-based Security Toolkit named SecKit, which is integrated with the MQTT protocol layer. In this work, authorizations and obligations are identified and a specific module (i.e., a Policy Enforcement Point) acts as a connector to intercept the messages exchanged in the broker with a publish/subscribe mechanism. Due to the similarity of such an approach with our AUPS solution, in Section 4.1 we provide a comparison between the two solutions, pointing out their crucial differences.

3. System architecture

Before presenting the proposed solution, the underlying IoT middleware, composed of multiple Networked Smart Objects (NOSs) [7,8], is introduced. In Fig. 1 its high-level layered structure is shown.

Starting from the bottom, the southbound interfaces of a NOS use HTTP as network protocol to communicate with IoT devices; such interfaces include the handling of the data transmissions by different sources (i.e., the nodes) as

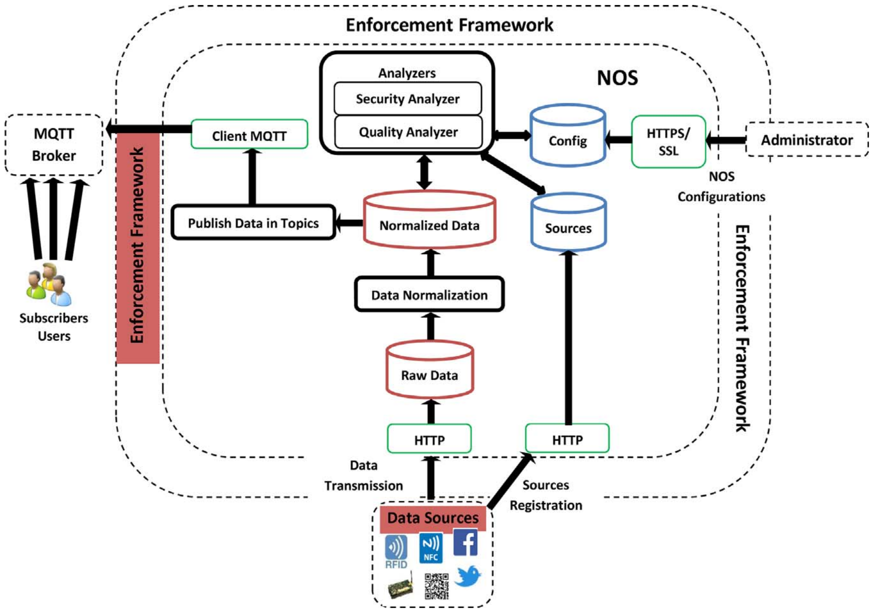

Fig. 1. System architecture.

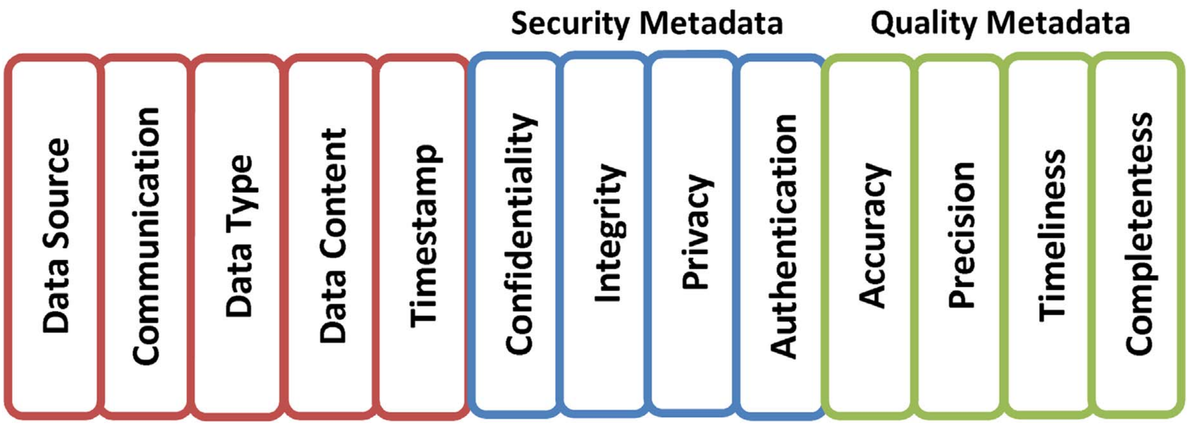

Fig. 2. NOS data format.

well as a service for source registration (each NOS deals both with registered and non-registered sources). The registration is not mandatory, but it provides various advantages in terms of security, since registered sources may specify an encryption scheme for their interactions with NOSs, thus protecting their communications. The information related to the registered sources are stored in the *Sources* collection. For each incoming data, both from registered and non-registered sources, the following information is extracted: (i) the data source type, which describes the kind of node (e.g., sensor node, actuator, RFID, NFC, social networks); (ii) the communication mode, that is, the way in which the data is collected (e.g., discrete or streaming communication); (iii) the data schema, which describes how the data content is structured and the format of the received data; (iv) the data itself; (v) the timestamp at which the data arrived to NOS. Since the received data are of different types and formats, NOS initially caches them in the *Raw Data* storage unit and periodically elaborates them according to the two-phase structure shown in Fig. 1. This includes two steps (*Data Normalization* and *Analyzers*), where results are put in a uniform data representation. First, the message stored in *Raw Data* is put in the format specified in Fig. 2 by the *Data Normalization* module, which stores them in another storage unit, named *Normalized Data*. This represents a sort of pre-processing phase in which the unnecessary information is removed from the data (it depends on the specific application domain); at this stage, security and quality metadata are still empty. Then, a second module, consisting of a set of *Analyzers*, periodically extracts the normalized data from the storage unit *Normalized Data* and elaborates them (in terms of security and data quality properties). Such an analysis implies that the data are annotated with a set of metadata (i.e., a score in the range $[0, 1]$ for each security and quality level). The rules, used for the assessment of security and quality scores, are stored in a proper format in another NOS storage unit, named *Config*. Note that such rules are not covered in this work; more details can be found in [8]. *Config* contains all the configuration parameters required for the correct management of the IoT system (e.g., how to calculate quality properties, which attacks or security countermeasures to consider); it can also be re-configured at run-time by an IoT system administrator through a secure connection (e.g., HTTPS, SSL) without the need to re-start the NOS application. The NOS *Analyzers* query the *Config* storage unit in order to know which actions they have to undertake on data. The semantic description of the data content itself is represented in Fig. 2. The data thus processed are used for providing services to the interested users (or external applications) by means of a publish/subscribe mechanism, as described in Section 3.2. Therefore, in order to achieve such a goal, the NOSs layer should be connected to IP-based networks (i.e., Internet, intranet). Note that multiple NOSs may co-exist, each of them serving a subset of the IoT devices present in the environment.

NOS northbound interfaces are instead based on the MQTT protocol, described in detail in Sections 3.1 and 3.2. Users and applications must previously register themselves to the IoT system; in such a phase, they are provided with credentials useful for accessing the system by means of a proper interface, as specified in Section 5. In particular, the key used by the user/application for accessing the IoT interface can be updated at any time, as happens in traditional authentication systems. Hence, the system allows both the registration of users and of external applications, which authenticate to NOS and then make requests to the services made available by the NOS itself. In case of an application registration, multiple users may register to such an application, instead of registering to NOS.

3.1. MQTT protocol – introduction

MQTT is a lightweight broker-based messaging protocol, designed and developed for constrained devices and bandwidth-limited communications by IBM/Eurotech in 1999 [5]. Such a publish–subscribe paradigm is developed following an event-based architecture, in which publishers publish structured events to an event service, usually called *broker*, and subscribers show their interest in a particular event through subscriptions. These subscriptions can be custom patterns over the structured events. Subscriptions notifications by publishers are sent to all the interested subscribers, in order to prevent the publishers needing to synchronize with subscribers.

Due to its simplicity and low overhead, MQTT is suitable for resource-constrained environments and has found application in several domains, including monitoring applications, applications with live feeds of real-time data (e.g., RSS feeds), dissemination of events related to advertisements, support for cooperative working where users/applications need to be informed about events of interest, support for ubiquitous computing, etc. MQTT specifications are open. Open source MQTT implementations are available for all major IoT development platforms, for the two major mobile platforms (i.e. Android and iOS), and for several programming languages (Java, C, PHP, Python, Ruby, Javascript). Moreover, the MQTT protocol has recently been adopted as a standard by the Advancing open standard for the information society (OASIS) consortium.

We remark that we chose MQTT since it represents the de facto standard protocol used by a variety of IoT and M2M systems. However, MQTT natively provides a very simple security model. For authentication of clients by server, the currently MQTT v.3.1.1 Protocol Specification only allows the use of a username and, optionally, a password. Developers can define customized authentication mechanisms using AES and DES as cryptographic primitives. Nevertheless, this is not sufficient for guaranteeing a mutual authentication among clients and servers; the integrity and the confidentiality of the transmitted information is also left open. More in detail, for encryption and transport-level security, the Transport Layer Security (TLS) standard is recommended, although this is not always a adequate choice for resource-constrained devices. Some implementations also support the use of a Pre-Shared Key (PSK) with TLS for authentication as well as encryption. Such a solution is definitely not suitable for highly dynamic IoT environments, since this strategy would require frequent session re-negotiations to establish new cryptographic parameters (i.e., change authentication credentials) among NOS and the registered sources/users/applications. Therefore, as described in Section 4, we propose a more lightweight credential management solution, also concerning temporary keys with the aim to improve the system's resilience to malicious attacks.

3.2. MQTT protocol – technical aspects

Some key features of MQTT are the followings:

- It provides one-to-many message distribution and decoupling of information sources and consumers.
- It is agnostic about the content of the payload.
- It is built over TCP/IP protocols.
- It has a small transport overhead.

MQTT, as in general all the publish/subscribe models, consists of a small set of operations, including the primitives pointed out in Table 1. As just said, all these operations are mediated by a broker, which is responsible to dispatch the events from the publishers to the interested subscribers. Both centralized and distributed architecture implementations are available. Obviously, the simplest approach is the centralized one: in this case, every communication (i.e., from publishers to broker or from broker to subscribers) takes place through a series of point to point messages. However, the broker could become a bottleneck. To prevent such a situation, the broker can be replaced with a network of brokers that cooperate to offer the services. In the solution proposed in this paper, we consider for the sake of simplicity multiple NOSs connected to one broker, albeit this can be easily replaced by a plurality of brokers in larger scenarios.

As far as MQTT topics are concerned, they present the following features:

- They are represented as UTF-8 strings used by the broker to filter messages for each connected client.
- They consist of one or more topic levels separated by a forward slash, forming a logical tree structure (e.g., a topic for publishing the temperature information of a sensor with identifier *sensorId* could be *sensor/sensorId/temperature*).
- They are used by clients for publishing messages and for subscribing to the updates from other clients, thus avoiding a continuous polling between producers and consumers.
- It is possible to subscribe to an exact topic or to multiple topics at once by using the wildcards, which are represented by the following symbols:
  - +, for a single-level wildcard (i.e., exactly one topic level).
  - #, for a multi-level wildcard (i.e., an arbitrary number of topic levels).

Table 2 shows the MQTT message format, consisting of three parts: a fixed and a variable header, and a payload. For further details we refer to [22].

Concerning reliability, MQTT is based on TCP, so it provides standard TCP delivery reliability. Furthermore, three levels of application-level QoS are supported, as summarized in Table 3.

MQTT also supports persistence of messages to be delivered to future clients that subscribe to a topic, and may be configured to send messages of specific topics when the subscriber connection is abruptly closed. Such a kind of configuration is established by the IoT system administrator on the basis of the IoT application requirements, and, regarding the NOS architecture, it is specified in the *Config* storage unit.

Regarding NOSs, they have a module in charge of assigning the corresponding topics to data and then publishing them to a MQTT client (module *Publish Data in Topics* in Fig. 1). A MQTT client, as that contained in each NOS, exchanges messages with a MQTT broker by means of publish and subscribe operations. The MQTT client running in a NOS system (*Client MQTT* in Fig. 1) publishes messages under the specific topics. The assignment of the topics depends on the application domain and is out of the scope of this work, but it is important to remark that the assignment should be linked to the definition of an ontology able to represent the semantics of the managed resources. Note that subscribers may register for specific topics at runtime and NOSs provide a mechanism for dynamic subscription e unsubscription to the topics.

Table 1
MQTT operations set.

Primitive	Description
publish	Used by a publisher to disseminate an event
notify	Used by a subscriber to receive an event notification (a topic update)
subscribe	Used by a client to subscribe to a specific topic
unsubscribe	Used by a client to unsubscribe from a specific topic

Table 2
MQTT message format.

Component	Information
fixed header	Message type, Quality of Service (QoS) level, some flags, message length
variable header	It depends on the application domain (information type)
payload	The data value referred to the topic

Table 3
MQTT QoS.

Component	Information
at most once	Messages are delivered according to the best effort of TCP/IP networks, so message loss and duplication can occur
at least once	Messages are assured to arrive, but duplicates may occur
exactly once	Messages are assured to arrive exactly one time

3.3. Enforcement framework

NOS is integrated with a policy enforcement framework, as sketched in Fig. 1; here it is represented around the core of NOS functionalities, since it has to manage the access to the resources against violation attempts. The enforcement components include [23] a Policy Enforcement Point (PEP), which is the point that intercepts the requests of access to resources from users/devices, and makes the decision requests to PDP in order to obtain the access decision (i.e., approved or rejected); a Policy Decision Point (PDP), which evaluates the access requests against the authorization policies before taking the authorization decisions; a Policy Administration Point (PAP), which contains the authorization policies established by the system administrators. In this paper, the focus is on the policies related to the access to resources and, in particular, to those concerning the publishing of the information and their notification to the subscribers, which are detailed in Section 4; however, the system may include also other kinds of policy, beyond the MQTT scope (e.g., regarding the data processing).

The access control model considered in this paper is the Attribute Based Access Control (ABAC) [24]. In such a mechanism, both the subjects, who want to access or to provide the resources, and the objects (i.e., data), which represent the resources themselves, are described by means of specific attributes, which are used for the policies definition. Attributes can be based on the metadata fields natively supported in our data representation and control rules can be defined according to the specific needs of the application domain. This feature is fundamental for guaranteeing the proper flexibility to ensure applicability to heterogeneous IoT environment.

Although so far we referred to only one NOS, the IoT system can comprise one or more NOS and a huge amount of nodes (acting as data producers), and users/applications (acting as data consumers). Each NOS is provided with a set of policies, therefore, their distribution, update and synchronization have to be considered. These operations are carried out through the *Config* storage unit, which is in charge of update the PAP. In case of multiple NOSs interacting with each other, all NOSs share the same security policies, in case of the same IoT context, and each of them has its own policy enforcement component. An important feature of the proposed policy environment is that it supports the loading of new policies at runtime, without the need of off-line updates, since the enforcement framework is kept decoupled from other management components belonging to the core NOS functionalities.

4. MQTT policy enforcement

In order to provide a secure system able to enforce the policies defined in a specific application domain, it is fundamental to integrate a secure MQTT protocol and an enforcement framework within the overall NOS architecture. This is what we term AUPS (AUthenticated Publish/Subscribe system for the Internet of Things).

As described in Section 3, each NOS has southbound interfaces towards the data sources and northbound interfaces towards the users/applications. Concerning the southbound interfaces (based on HTTP protocol), the following endpoints are provided:

- *POST/data*: Used for handling transmission of data from the nodes to NOSs. The data format is just required to be a valid JSON node and is completely domain-dependent.
- *POST/registration*: Used by nodes for registering to NOS. The body of request is required to be a valid JSON node. The following fields are mandatory: `NodeId`, `NodeType`, `CommunicationMode`. Optional field: `EncryptionScheme`. The response includes node credentials in the case of a valid registration request.

NOSs deal both with registered and non-registered sources, so data from unknown nodes are also accepted; moreover, HTTP offers in-built authorization and authentication functionalities, thus generating no noticeable security issues.

The problems arise with the northbound interfaces, which are based on MQTT protocol. An application/user wanting to use data from NOS can access it through subscription to the relevant topic(s), handled by the MQTT broker. Yet, the resources are accessible on the basis of the policies defined within the NOS enforcement framework. Therefore, the MQTT broker has to interact with the underlying PEP on NOS in order to accept or deny subscription requests.

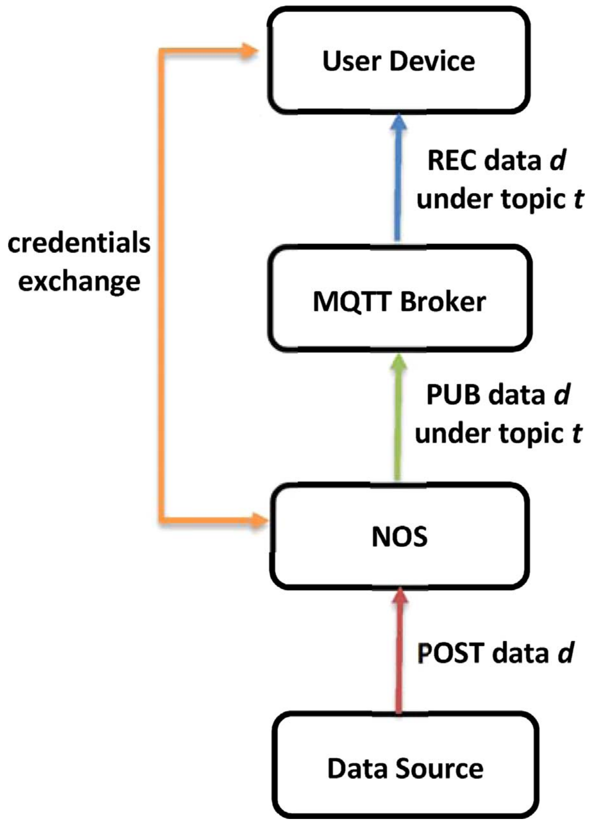

Fig. 3. High-level information flow.

In such a context, our AUPS solution aims to propose a new system for the enforcement management of users/applications authentication and authorization policies through integration with the MQTT mechanisms. The involved entities and the flow of information are shown in Fig. 3. Four relevant actors are present in the system:

- The *data source*, which communicates data to NOS using the HTTP protocol.
- The *NOS*, which processes the data according to the procedures described in Section 3 and publishes them under the relevant topic(s) to the MQTT broker.
- The *MQTT broker*, which notifies the interested subscribers of the new incoming data.
- The *user* (*or*, *equivalently*, *service*), which accesses IoT-generated data through subscriptions to the MQTT broker.

In particular, two transactions, strictly related to MQTT, have to be handled, for simplicity named *PUB* and *REC*: *PUB* represents the publication of new data to a topic by NOS; while *REC* represents the notification and reception of new published data to the subscribers. Before detailing such operations, the Keys Topics Manager (KTM) has to be introduced. It is added as an external NOS component, which is able to interact with NOS by means of a secure HTTPS/SSL connection, in the same way as for the configurations contained in *Config* collection (i.e., managed by an external administrator). KTM is in charge of

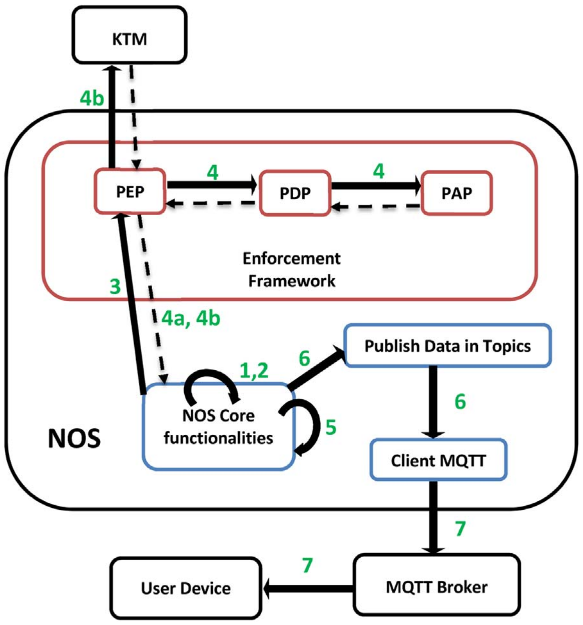

Fig. 4. *PUB* transaction.

managing temporary keys for topics access control. In particular, it handles a table structure with the following fields:

- *keyId*: the identifier of the corresponding key.
- *keyT*: the actual key.
- *val*: the expiration date of the key.[3]
- *atb*: the attribute(s) owned by the users/applications allowed to access the resource. The key is associated to the correct users/applications on the basis of the attributes by the corresponding policies, expressed in PAP component and defined by system administrator. In this way, the KTM is decoupled from the enforcement framework.

We remark that the keys are not fixed, but they come with an expiration date, which is expressed in the field *val*. Moreover, the expiration times can be out-of-phase from each others, thus without affecting the system load. Since, as explained in Section 3.3, the policies are global for all the NOSs belonging to an IoT system, also the keys topics management has to be global due to the attributes configurations. In this way, we also guarantee the NOSs synchronization in terms of active policies. Moreover, as just introduced in Section 3.3, the access control is based on ABAC; in fact, each user/application has to pass a registration phase before interacting with the IoT system, in which a set of attributes is assigned on the basis of the specific application domain (e.g., a manager and an employee of a financial company should have different attributes for accessing the resources of the company itself).

As regard the operations, *PUB* transaction is composed by the following steps, also represented in Fig. 4. Note that, for the sake of simplicity, in rest of the section we refer to

[3] This clearly requires synchronization among hosts.

NOS meaning *NOS Core functionalities*, as shown in Fig. 4. This is due to the fact that the enforcement framework belongs to the NOS as a running module, but, at the same time, it is queried by proper NOS core functionalities when needed.

1. NOS produces a new data item d, which has passed both the normalization and the analyzers phases (Section 3).
2. The data d is assigned to a specific topic t.
3. NOS queries the PEP in order to obtain the information useful for protecting the access to the resource represented by the data d relative to the topic t.
4. PEP queries the PDP in order to know which attributes *atb* a user/application has to own in order to access the resources represented by the topic t. PDP returns to PEP the information related to the attributes *atb* in accordance with the policies specified by PAP. Then, PEP can perform two actions:
   (a) If it owns a valid key for the access of users/applications with attributes *atb* to data published under topic t, then it sends *keyT*, along with the corresponding *keyId* and *val* to the NOS.
   (b) If it does not own a valid key for the access of users/applications with attributes *atb* to data published under topic t, it asks a new valid key to the KTM before sending to the NOS *keyT*, *keyId* and *val*.
   Note that, once a PEP of a NOS has asked a new valid key, it caches it locally. When PEP finds out that a key is not valid any longer, then a request to get a new one is triggered.
5. NOS encrypts the data d with the obtained *keyT*, thus obtaining the encrypted data d_{enc}:

$$d_{enc} = enc(d, keyT, keyId), \tag{1}$$

   where any existing encryption mechanism $enc(\cdot)$ can be used.
6. NOS prepares the message to send for being published in the format specified by the following JSON syntax:

$$\begin{array}{l} \{ \\ \quad \text{“}keyId\text{”}: keyId, \\ \quad \text{“}data\text{”}: d_{enc} \\ \} \end{array} \tag{2}$$

7. The NOS MQTT client sends such information to the MQTT broker to be published under the specific topic t and to notify the interested subscribers.

After the *PUB* transaction, the *REC* transaction takes place, since the data d has been published, in an encrypted way (i.e., d_{enc}), under the proper topic t. As a consequence, all the subscribers interested in the topic t have been notified about the new published data. Supposing that a user device u receives a new notification (the case of application is omitted, but analogous), the steps to be performed are the following ones, as also represented in Fig. 5:

1. The user device u can do two actions:
   (a) If it already owns a valid key for the access to the resources specified by the topic t, then it can go to the final step.
   (b) If it does not own a valid key for the access to the resources specified by the topic t:
      (i) u issues a request to NOS to access the information needed for decrypting the data d_{enc}.
      (ii) NOS queries the PEP, which in turn queries the PDP, in order to establish if the requesting user u with its attributed *atb*, is allowed to access to the resources of topic t; the response depends on the policies activated within PAP (and established on the basis of the attributes owned by the user as established during the preliminary registration phase).
      (iii) Then:
         (A) If the response of the PEP is positive, then the PEP sends to NOS the information related to the key *keyT*, the identifier *keyId* and the validity *val*, which are then sent to u, encrypted with the key obtained by the user during the preliminary registration phase.
         (B) If the response is negative, an error message is sent by PEP to NOS and, then, by NOS to u.[4]
2. Once in possession of the key *keyT* and the identifier *keyId*, the user (or: service) u is able to decrypt the data d_{enc}. We remark that a user does not have to require the key at each notification, since the key has a validity indicated by the timestamp *val*.

Summarizing, the adopted approach is able to effectively decouple aspects related to security features from the usage of the MQTT pub/sub protocol; at the same time, network and computing resources are used effectively. Furthermore, the use of temporary keys and of user/application registration improves the system's resilience to malicious attacks (e.g., man in the middle attacks, replay attacks, password discovery).

4.1. Robustness evaluation

In order to further clarify the innovative contribution of the presented AUPS solution, we compare it with another existing approach integrated in SecKit [21] and briefly described in Section 2. SecKit also aims to address the lack of security policy enforcement capabilities in existing MQTT implementations, but it follows a different approach.

SecKit extends the implementation of the open source Mosquitto MQTT broker with a security plugin, in a way that the PEP directly resides on the broker. Instead, in AUPS, PEP and the other enforcement components reside on each NOS. Note that NOSs represent a real IoT middleware with data processing and source management capabilities (as described in Section 3); while in [21] the deployment of a large scale IoT middleware is left out of the scope. Hence, [21] limits the presentation of the results to a restricted scenario, in which all the load of policy

[4] Note that if the PEP has not a valid key for the data belonging to topic t, it has to ask them to KTM, as described for *PUB* transaction. Moreover, the fields *keyT*, *keyId* and *val* are not sent in clear by NOS to u, but they are also encrypted with the key of the device u itself, established during the initial registration phase.

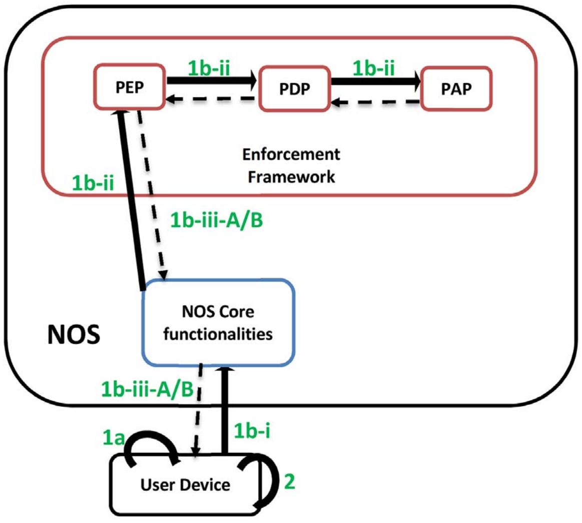

Fig. 5. *REC* transaction.

enforcement is moved on the MQTT broker, thus increasing its computational overhead. The major implication of porting PEP on the broker, as SecKit does, is that the native lightweight efficiency of Mosquitto in terms of processing and memory capabilities may be compromised, in particular in a large-scale application setting, with huge amount of topics and adopted policies. For this reason, we decided to keep the security enforcement management on NOSs, which have more processing and storage resources. Moreover, the usage of a uniform language for implementing PEP and PDP (in our case: javascript, using the *Node.JS* platform[5]) ensures easier integration and maintenance than the mixed approach (PEP in C and PDP in Java) used in SecKit. Furthermore, if PEP resides on the MQTT broker, then the violation of the broker itself by malicious entities may compromise the whole network, breaking access decision allowing non-authorized users/applications to access restricted information. Otherwise, in AUPS, NOS itself would be violated, which is a more complex task and, in addition, the key management is controlled by another external entity, the KTM, which is decoupled with respect to NOSs, thus increasing the robustness of the entire system.

A further crucial difference between AUPS and SecKit is that our access control acts at the topic level, while, in [21], the authors propose to consider the delivery of individual messages and filter the access on the basis of the content of the payload. To do this, SecKit has to include in the model the concepts of context and situation, by means of which PDP is able to assign different policies to the same information depending on the actual conditions. As a consequence, authorizations and obligations are specified using an Event-Condition-Action (ECA) structure and, at any time, before allowing the access to a resource, the system has to evaluate the actual events, which take part within the system. In our opinion, such an approach may seriously affect the system performance; in fact, not only an ECA hierarchy has to be defined for each new resource, but also the related events, to be executed in response to particular scenarios, need to be specified, thus requiring a considerable computational effort. Therefore, we decided to perform access control at a coarser level of granularity, using ABAC (instead of ECA) and without requiring the access control operations at each notification event by introducing the concept of temporary keys, thus reducing the computational and storage overhead. Another advantage of the adoption of KTM and temporary keys on NOSs is represented by robustness against violation attempts. In fact, our mechanism allows us to preserve the system from credential discovery and replay attacks, since the keys, used for decrypting the information, have a limited temporal scope. Also as regard man in the middle attacks, even if a malicious entity listens to the communications between broker and user devices (or external application), it will not be able to intercept the clear content, since the encryption varies over time, without a fixed or predictable pattern. Furthermore, if a malicious device obtains one or more *keyT*, then it may perform a Denial of Service (DoS) attack; also this action is mitigated since, in AUPS, when the key will expire, the device will not own the correct credentials for authenticating itself to NOS, therefore it will not obtain any new valid key and it will not continue to execute the attack. Is is important to remark that SecKit only refers to DoS, which is recognized by monitoring the flood of message, but no other kind of attacks is handled; moreover, as regard DoS attack, no possible countermeasures are specified by the authors in order to face such violations.

5. Evaluation scenario

The AUPS solution presented in the previous section has been implemented and the code is freely released under an Apache v.2 license.[6] The implementation was carried out using the following components/technologies/libraries: (i) NOS is developed over a *Node.JS* platform,[7] (ii) *MongoDB*[8] is used for database management; (iii) the Mosquitto[9] MQTT broker is used for the publish/subscribe mechanism. All the modules, including the enforcement framework, interact among themselves through *RESTful* services. In this way it is possible to add new modules or duplicate the existing ones, since they are able to work in a parallel (non-blocking) manner or to define new functionalities or removing the active ones. Moreover, *MongoDB*, as a non-relational database (i.e., NoSQL), allows the data model to evolve dynamically. Indeed, the data are handled in *JSON* format, which is also a lightweight data-interchange format.

In our experimental setup, a NOS runs on a Raspberry Pi, which, in our test environment, receives real-time data feeds from six sensors located in the meteorological station of the city of Campodenno (Trentino, Italy). Data are

[5] http://nodejs.org/.

[6] https://bitbucket.org/alessandrarizzardi/nos.

[7] http://nodejs.org/.

[8] http://www.mongodb.org/.

[9] http://mosquitto.org.

formatted in *JSON*, and include information on temperature, humidity, wind, energy consumption and air quality. In the following example, NOS fetches the data at two different rate: 10 packets per second and 20 packets per second. This frequency obviously influences the memory occupancy as well as the computational effort. Others parameters which may affect the performance of the whole system are represented by the number of sources and of interacting users. As just said, we consider six data sources, whereas the number of users is fixed to two. With the term “fixed”, we mean that, in a real application scenario, users may join (i.e., by means of NOS registration) or leave the IoT network at any time. In the remainder of this section, we present the results, in terms of performance, achieved experimentally by our implementation.

5.1. Overhead analysis

The overhead in terms of storage capacity and computational effort has been analyzed. In our test-bed, NOS runs on a Raspberry Pi with 512 MB of RAM and a variable physical memory, which depends on the capacity of the adopted microSD; in the present scenario, the microSD has 8 GB of memory. An important premise is that NOS does not support persistent storage of IoT-generated data. In fact, data are only temporarily cached on the NOS while being processed before being submitted to the MQTT broker. Once data are pushed to or pulled from the server which handles the topics notification to subscribers, data can be safely flushed from the NOS itself. In our prototypical implementation, the in-memory capability of *MongoDB* has been used for *Raw Data* and *Normalized Data* collections, while *Config* and *Sources* databases are persistently stored on the local hard disk. Hence, the memory occupancy resulted on average slightly less than 7.5 megabyte, where the 8% is used for AUPS. More in detail, NOS has always to maintain the set of tuples *T* in the form *keyId, keyT, val, atb* for each combination of attributes associated to the topics and established by the policies themselves. In our example, the considered topics are six (i.e., one for each kind of data provided by the sources). At this stage, the number of possible combination for the access to resources depends on the granularity of the policies required by the system administrator. In fact, the system administrator may decide to associate a different attribute for accessing each topic or, instead, group them, so that access is allowed to a subset of the data. Note that we adopt ABAC, as explained in Section 3.3.

Two users are considered for our validation: the former with access to topics 1 *campodenno/sensor1/temperature* and 2 *campodenno/sensor2/humidity*; the latter with only access to topic 2. Such policies may be replicated for each of the six kinds of data provided by the sources. As a consequence, the amount of memory required on NOS is strictly related to the structure and granularity of access permissions and, more in general, to the specific application domain. In the presented case study, the physical memory needed is very low (the 8% of the memory occupancy, as just said), since few kinds of information are provided to the system by the meteorological sensors. Moreover, note that the dimension of each tuple *T* varies on the basis of the length of the fields *keyT* and *atb*. Hence, the application of the proposed approach in a wide real scenario has to take into account such aspects related to memory occupancy in the definition of attributes and access permissions.

Instead, as regard the computational overhead with respect to the original NOS MQTT version, we consider that, for each incoming data, the following operations are executed (in brackets, the measured average execution time):

- Three database queries for obtaining the access permissions to be associated to the new data (2.4 ms).
- One operation of encryption (0.5 ms).
- One HTTP call to the KTM, only in case of expired key (3.2 ms).

While, when a new data is notified to a user, then it is the user device which is in charge of executing the operation of decryption; NOS is asked a new valid key only in case of an expired one on the user device. Such an activity requires:

- Three database queries for obtaining the access permissions to be associated to the user (2.4 ms).
- One HTTP call to the KTM (3.2 ms).

Finally, an important remark has to be done about the KTM. In fact, being an external entity, KTM does not delegate the keys and attributes management to NOS, thus saving memory and processing capacity. Moreover, KTM cannot be considered a bottleneck, besides it serves all NOSs, since its operations (i.e., the temporary keys generation) are asynchronous with respect to NOSs activity. In fact, we suppose that the key expiration time is not the same for all the generated keys, thus allowing a balanced load of HTTP calls made by NOSs to the KTM during the system running. Obviously, a large-scale evaluation should be done in order to better assess the KTM performance.

5.2. Latency

The next performance analysis of AUPS investigates the end-to-end latency introduced by the adoption of the proposed secure mechanism to the native MQTT transactions, executed in the previous version of NOS [7] (i.e., with the secure extension disabled). Latency is computed as the distribution of the elapsed time from the data reception to NOS until it is sent to the broker. Latency is computed for two different settings, with and without AUPS. For AUPS, the key expiration timeout was set to 5 min. Fig. 6 shows the distribution of the latency for one NOS and the six data sources, measured over a period of 24 h and with two different data fetch rate: 10 packets per second and 20 packets per second. In the original configuration, i.e., without AUPS, the average measured latency is approximately 3.5 ms and 4 ms for 10 and 20 packets per second, respectively. Instead, once enabling the secure extension, such an average time increases by 2 ms. What emerges is that the new operations add a

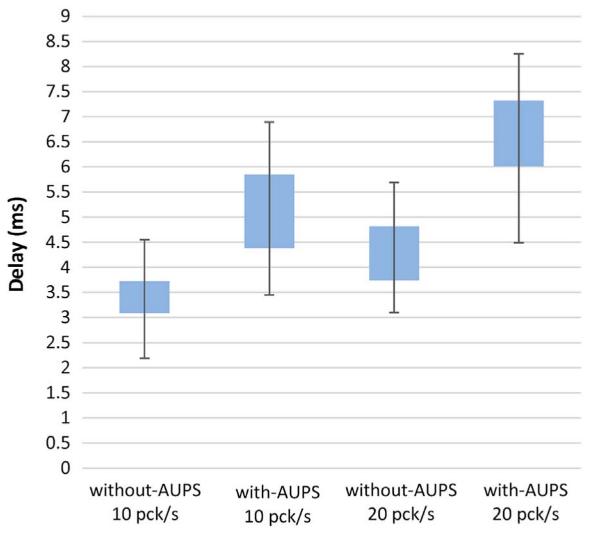

Fig. 6. Latency: whiskers-box diagram for a system with and without AUPS with two different data fetch rate (10 and 20 pkt/s).

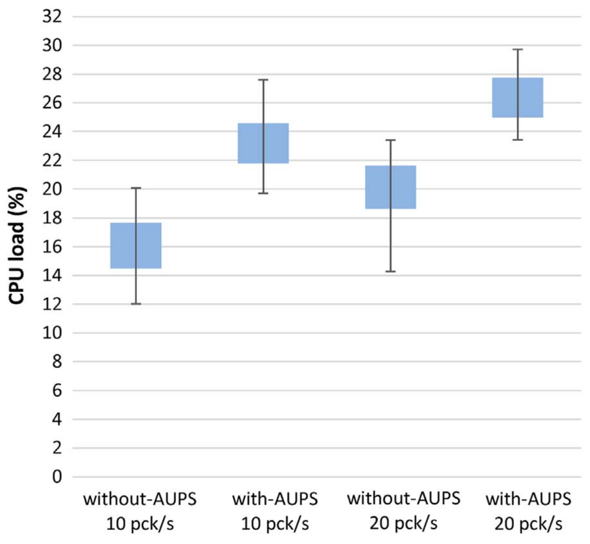

Fig. 7. CPU load.

stable increase of the delay, which remains under an acceptable threshold.

5.3. *Computing*

Taking in mind the same parameters used for the latency performance evaluation, Fig. 7 shows the distribution of the CPU load on the NOS with and without AUPS. Also in this case the computational efforts is stable during system running and results are very promising for encouraging the adoption of AUPS solution in a large-scale environment. It is important to note that both latency and computing performances are affected by the number of users, subscribed to the topics provided by NOS. In fact, such parameters influence the number of keys to be generated and, therefore, the overall resources required by AUPS.

Furthermore, we remark that this represents only an example of a very simple NOS application in a context characterised by the analysis of real-time data. One aspect which deserves some further clarifications refers to the fact that in our example we considered one single NOS. While indeed we aim to deploy the presented middleware in a distributed environment, from an analysis of NOS functionality it is not difficult to see that no NOS-to-NOS coordination is strictly required. In fact, NOSs are able to (i) independently handle the data sources, without the need to inform the other NOSs of their active and past interactions; (ii) be independently re-configured by IoT system administrators; (iii) independently assign topics and publish data on the basis of the defined rules; (iv) enforce the application of the policies defined for the IoT system. Therefore, we can safely conclude that considering a single NOS-scenario for validation purposes does not represent a limiting factor.

6. Conclusions

The lack of a comprehensive security solution represents a major threat to the growth and market take-up of Internet-of-Things systems across a variety of vertical application domains. MQTT is emerging as a de facto standard messaging protocols for IoT applications, yet in its current version it provides little security support. In this work we presented AUPS, a system and methods for adding security to MQTT-based IoT systems. AUPS includes a policy enforcement framework, coupled with a key management one, and is able to effectively manage publications and subscriptions through MQTT interactions. The proposed solution has been implemented and a working implementation is released, under an open source license, to the research community at large. Future extensions include testing in a larger, more complex setting, possibly characterised by the presence of a plurality of networked brokers and NOSs, where issues related to synchronization of policies among hosts may arise.

References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, I. Chlamtac, Internet of Things: vision, applications and research challenges, Ad Hoc Netw. 10 (7) (2012) 1497–1516.
[2] S. Sicari, A. Rizzardi, L.A. Grieco, A. Coen-Porisini, Security, privacy and trust in Internet of Things: the road ahead, Comput. Netw. 76 (2015) 146–164.
[3] D. Boswarthick, O. Elloumi, O. Hersent, M2M Communications: A Systems Approach, 1st ed. Wiley Publishing, USA, 2012.
[4] L.A. Grieco, M.B. Alaya, T. Monteil, K.K. Drira, Architecting information centric ETSI-M2M systems, IEEE PerCom (2014).
[5] IBM and Eurotech "MQTT v3.1 Protocol Specification", ⟨http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html⟩.
[6] S. Raza, H. Shafagh, K. Hewage, R. Hummen, T. Voigt, Lithe: lightweight secure CoAP for the Internet of Things, IEEE Sensors J. 13 (10) (2013) 3711–3720.
[7] A. Rizzardi, D. Miorandi, S. Sicari, C. Cappiello, A. Coen-Porisini, Networked smart objects: Moving data processing closer to the

source, in 2nd EAI International Conference on IoT as a Service, October 2015.
[8] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, A. Coen-Porisini, A secure and quality-aware prototypical architecture for the Internet of Things, Inf. Syst. 58 (2016) 43–55.
[9] Z. Wu, L. Wang, An innovative simulation environment for cross-domain policy enforcement, Simul. Model. Pract. Theory 19 (August (7)) (2011) 1558–1583.
[10] D. Ferraiolo, V. Atluri, S. Gavrila, The policy machine: a novel architecture and framework for access control policy specification and enforcement, J. Syst. Archit. 57(April (4)) (2011) 412–424.
[11] J. Rao, A. Sardinha, N. Sadeh, A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources, Web Semantics: science, Services Agents World Wide Web 7 (1) (2009) 40–56.
[12] J. Rao, A. Sardinha, N. Sadeh, A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources, Web Semantics: science, Services Agents World Wide Web 7 (January (1)) (2009) 40–56.
[13] M. Dell'Amico, M.S.I.G. Serme, A.S. de Oliveira, Y. Roudier, Hipolds: a hierarchical security policy language for distributed systems, Inf. Secur. Techn. Rep. 17 (February (3)) (2013) 81–92.
[14] J. Singh, J. Bacon, D. Eyers, Policy enforcement within emerging distributed, event-based systems, in: DEBS 2014—Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, 2014, pp. 246–255.
[15] V. Kapsalis, D. Karelis, L. Hadellis, G. Papadopoulos, A context-aware access control framework for e-service provision, in: IEEE International Conference on Industrial Technology, 2005. ICIT 2005, December 2005, pp. 932–937.
[16] C. Bertolissi, M. Fernández, A metamodel of access control for distributed environments: applications and properties, Inf. Comput. 238 (2014) 187–207.
[17] P. Waher, Security in Internet of Things using delegation of trust to a provisioning server, 2014.
[18] M. Kirsche, R. Klauck, Unify to bridge gaps: Bringing XMPP into the Internet of Things, in: 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), IEEE, USA, 2012, pp. 455–458.
[19] S. Pallickara, G. Fox, H. Gadgil, On the creation & discovery of topics in distributed publish/subscribe systems, in: The 6th IEEE/ACM International Workshop on Grid Computing, 2005, IEEE, USA, 2005, p. 8.
[20] P. Fremantle, B. Aziz, J. Kopecky, P. Scott, Federated identity and access management for the Internet of Things, in: 2014 International Workshop on Secure Internet of Things (SIoT), IEEE, USA, 2014, pp. 10–17.
[21] R. Neisse, G. Steri, G. Baldini, Enforcement of security policy rules for the Internet of Things, in: 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), October 2014, pp. 165–172.
[22] A. Banks, R. Gupta, qtt version 3.1. 1, OASIS Standard, 2014.
[23] N. Ulltveit-Moe, V. Oleshchuk, Decision-cache based XACML authorisation and anonymisation for XML documents, Comput. Stand. Interfaces 34 (6) (2012) 527–534.
[24] V. Goyal, O. Pandey, A. Sahai, B. Waters, Attribute-based encryption for fine-grained access control of encrypted data, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, 2006, pp. 89–98.