Dynamic Policies in Internet of Things: Enforcement and Synchronization

Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, and Alberto Coen-Porisini

***Abstract*—Security and privacy represent critical issues for a wide adoption of IoT technologies both by industries and people in their every-day life. Besides, the complexity of an IoT system's management resides in the presence of heterogeneous devices, which communicate by means of different protocols and provide information belonging to various application domains. Hence, adequate policies must be correctly distributed and applied to the information made available by the IoT network to secure the data themselves and to regulate the access to the managed resources over the whole IoT system. Policies mainly involve the access to resources and are usually established by system administrators in accordance with the rules of each specific domain. Since IoT concerns multiple application fields and often wide areas, a centralized solution which manages all the required policies would not be neither efficient nor scalable. Therefore, in this paper, a distributed middleware overlying the IoT network is proposed and integrated with a synchronization system for guaranteeing the correct distribution, update, and application of the policies across the entire IoT environment in real-time. Such a distribution and synchronization system has been developed within a policy enforcement framework. The presented solution has been validated by means of a simple yet real prototype; the analyzed metrics regard delay, overhead and robustness of the proposed enforcement and synchronization framework.**

***Index Terms*—Internet of Things, Security, Policy, Prototype, Middleware**

I. Introduction

In the last years, information sharing, obtained from different sources, is becoming a core requirement of the modern Internet of Things (IoT) application environments, in which users are provided with innovative services in real-time. Networking systems supporting such a kind of environments must manage the transmission of data across the boundaries of different application domains, in order to integrate heterogeneous information to fulfil users and vendors needs. The presented scenario involves devices interacting with different protocols over the network, thus emerging the need for the introduction of a distributed middleware layer able to manage the huge amount of available data and to cope with interoperability issues [1]. Also the rules that supervise the access to the IoT resources must be handled in a compliant manner within the whole IoT system by the middleware layer, thus requiring efficient and accurate methods for a correct distribution and synchronization with respect to the distributed nature of the environment.

S. Sicari, A. Rizzardi and A. Coen-Porisini are with the Department of Theoretical and Applied Science, University of Insubria v. G. Mazzini 5 21100, Varese (Italy), e-mail: sabrina.sicari@uninsubria.it, a.rizzardi@uninsubria.it, alberto.coenporisini@uninsubria.it.

D. Miorandi is with U-Hopper v. A. da Trento 8/2 38122 Trento (Italy), e-mail: daniele.miorandi@u-hopper.com.

To this end, the authors proposed, in [2] [3], a flexible and cross-domain IoT architecture, namely Networked Smart Objects (NOS) middleware. NOS aims to manage heterogeneous sources and evaluate the levels of security and quality of the information, in order to satisfy users requirements (e.g., in terms of data reliability or accuracy) and provide a lightweigth and secure information exchange process [4]. Such information concur to regulate the access to the resources, but it is not enough to provide a complete management of the data processing and sharing [5]. This is mainly due to the fact that NOSs are distributed over the wide IoT area and administered by one or more central entities that dictate the actual rules. As a consequence, such rules must be communicated to all the NOSs, possibly in real-time, in order to guarantee that the entire IoT system behaves in the same way all the time.

To cope with such an issue, adequate policies must be defined, distributed, and sychronized with the final aim of setting proper rules among the NOSs before releasing IoT data to users. The aforementioned policies, besides regulating the access to resources, have another important role, which consists in dealing with violation attempts. Moreover, due to the coexistence of hetereougeneous application realms, it is necessary to pay particular attention to control and regulate the flow of information from a domain to another, belonging to the IoT environment. A good solution is represented by the integration, into the NOS middleware, of a policy enforcement framework that acts as a sort of wrapper with respect to the tasks esecuted by NOSs. In this way, all the interactions among NOSs and the IoT entities (i.e., data sources and users) will be supervised following specific rules, established by the system's administrators. A partial solution to the just presented scenario is provided in [6], but too little effort has been done yet to define a comprehensive solution for the IoT data flow control. In fact, no mechanism has been provided to manage policy distribution and synchronization.

The definition of how information must be shared is a complex-prone task and must be treated with a distributed approach, as NOS does, mainly because of the high number and the heterogeneity of entities (e.g., devices and users) and information involved in the IoT scenario. In contrast, a centralized solution might not be sufficiently scalable [7]. More in detail, the main drawbacks of existing solutions are the followings: (i) they are not sufficiently flexible and scalable, intended as the capability of avoiding bottlenecks in the information flow; (ii) they do not consider temporary changes during the activity of the system; (iii) they do not adopt lightweight schemes for managing the various interactions.

Therefore, new mechanisms must be designed and they should be flexible enough to support the wide range of

technologies acting in the IoT infrastructures and the various application domains where users and devices could operate. As regards such a heterogeneity, policies'/rules' sychronization across multiple NOSs and hetereougeneous application realms must be investigated, and this represents the main goal of this paper. In fact, as just said, it is fundamental for guaranteeing that the IoT system behave in a compliant manner in all its parts.

Summarizing, in order to overcome the described issues, we propose a solution carrying out the following contributions:

- The integration, into the existing enforcement framework installed on the NOS middleware, of a mechanism for policies' distribution and sychronization, which considers the coexistence of different application realms
- Authorisation and behavioral rules are efficiently specified, distributed, and synchronized among multiple networked NOSs, in order to determine conditions in which particular actions are permitted to or must be executed by the different entities involved (i.e., NOSs themselves, data sources, users)
- The flow of information and policies within the IoT environment is properly supervised
- Violation attempts are prevented and blocked.

It is worth to remark that the adopted approach presents an important contribution, in particular in certain scenarios which require sychronization for guaranteeing a strict control on data and on the access to confidential resources, such as healthcare applications, vehicular and military systems.

The paper is organized as follows. Section II describes our contributions and motivation with respect to the actual state of the art. Section III presents the adopted moddleware architecture along with the integrated policy enforcement framework. Section IV describes our innovative contribution for policies' distribution and synchronization tailored to a distributed IoT environment, which is then validated in Section V by means of a real test-bed. Finally, Section VI ends the paper providing some hints for future works.

II. Related Works

The analysis of IoT secure distributed architecture and related enforcement mechanisms plays a fundamental role in order to allow the real diffusion and adoption of IoT paradigm.

[8] developed an information security policy process model, based on a methodology involving qualitative techniques, able to evaluate the key external and internal influences that can impact on organizational security against cyber threats. Such a model uses a data-centred approach, that lead to identify the primary policy processes, the key environmental and organization influences, and the relevant linkages among them. Note that such a process model represents a generalized framework rather than a specific model for a single company. Therefore, it does not consider the peculiar aspects of each organization in a way that the model may not equally apply to all organizations. Moreover, it does not address exceptional situations that may warrant a temporary violation of predefined policies. In our security policy schema, the aim is to go beyond such limitations by means of a distributed middleware able to manage the information of heterogeneous application domains and update the policies in real time in accordance with the organizational or users' actual requirements.

Also, the work presented in [9] highlights that security mechanisms are often enforced in a separated way from each other (e.g., no interactions or integrations among different companies or organizations), thus limiting the kinds of policies that can be enforced in distributed and heterogeneous settings. To cope with such an issue, [9] introduces the concept of a Security Service Bus (SSB), representing a dedicated communication channel between the applications and the different security mechanisms. It allows the enforcement of advanced policies by providing uniform access to application-level information. However, such a security infrastructure is not enough flexible and scalable for a wide IoT scenario, since the dedicated channel may become a bottleneck for the interactions among security services and bounded applications. To cope with such an issue, in our solution, we employ a publish&subscribe mechanism to allow the policies propagation and updates to be decoupled from the involved services.

Two theoretical approaches are presented in [10] and [11], which address the secure data sharing among different applications domains. In particular, [10] proposes a knowledge access control policy language model able to identify the knowledge access control and sharing rules across enterprises members. Such a language is based on an ontology, which aims at solving knowledge heterogeneity issues within companies and workers belonging to different areas. This is achieved by establishing relationships among the topics of interest of the involved parties, thus allowing a timely response to access authorization request, also in case of changes in the business environment. Instead, in the IoT and Big Data fields, [11] proposes the policy-carrying data (PCD) mechanism, with the aim to establish access control rules for data consumers. Several drawbacks of such a work are the followings: (i) the introduction of a complex language for policy definition; (ii) the need of a centralized entity able to evaluate the behavior of producers and consumers (which may represent a bottleneck); (iii) the only theoretical presentation of the proposed solution.

[12] and [13] investigate the secure information sharing in the healthcare context, since medical data have a sensitive nature they must be protected and released only to authorized parties. The solution proposed by the authors involves a publish&subscribe middleware, which is in charge of handling the access control policies on the data transmitted by the healthcare structures by customizing policies depending on the actual events. The defined policy rules are local to each administrative domain and proper mechanisms are provided to assist administrators to maintain a consistent policy set. However, the data disclosure and policy management are centralized, thus compromising the scalability of the whole system. Moreover, automatic updates and revocation of policies are not considered, as we done in the solution presented in this paper. Note that our proposed mechanism also includes the management of independent domains and related policies.

Also, concerning access control in publish&subscribe environments, other previous works address such an issue

by means of: hierarchical role-based rules based on subscribers' privileges [14]; attribute-level policies concerning multi-domain environments and involving the use of shared keys for the interaction with untrusted brokers [15]; policy integration and distribution along with performance implications [16]. We aim to extend and overcome such approaches by proposing a more flexible solution specifically tailored to the heterogeneous IoT context ([14] and [16] are only general theoretical approaches), able to manage not only policy over multiple domains, as [15] does, but also their synchronization in real time and in an efficient way.

Looking at the available literature, other works focus on the definition of centralized enforcement frameworks addressing the orchestration of policy across different domain boundaries, with no reference to IoT environments. For example, [17] provides a configurable middle-level component for the policy mapping process among interconnected systems; [18] presents an access control framework, named Policy Machine (PM), to which users may login and access to resources on the basis of the attributes assigned by the policies themselves; [19] and [20] enforce context-sensitive policies, which require the ability to derive the policy to be applied at each time on the basis of contextual information; [21] uses an intermediary server to control the access permissions (based on the RBAC model) to distributed web services; [22] comprises the concept of federation in order to apply global and local policies to each member. The last solution is the only targeted to a distributed environment. Taking in mind the available solutions, our work is focused on the capability to control the access to resources in a more efficient and general-purpose way. In particular, the integration of a lightweight policy distribution and synchronization schema with the existing NOS distributed middleware [23] is carried out.

III. NOS Middleware Architecture

Architectural components for the previous and the new version of NOS are sketched in Figures 1(a) and 1(b), and detailed in Sections III-A and III-B, respectively.

A. Previous NOS architectural components

We divide NOS interactions into three main groups, detailed in the next sections: (i) the southbound interfaces, in charge of managing the communications with the data sources, which are also generically referred as nodes (e.g., wireless sensor nodes, actuator, RFID, NFC, social networks); (ii) the northbound interfaces, consisting of a publish&subscribe mechanism able to made processed information available to interested users or applications, in the form of services; (iii) the enforcement framework, that acts as a sort of wrapper, responsible for properly managing the access to the available resources and handling possible violation attempts, by means of well-defined policies.

1) NOS southbound interfaces: The southbound NOS interfaces have been defined in [4]. They use HTTP as network protocol and include: (i) the handling of the data transmissions by different sources; (ii) a service for source registration. In fact, 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 NOS, thus protecting their communications. The information related to the registered sources are put in the storage unit, named *Sources*. While, for each incoming data, both from registered and non-registered sources, the following information are extracted: (i) the data source, which describes the kind of node; (ii) the communication mode, that is, the way in which the data are collected (e.g., discrete or streaming communication); (iii) the data schema, which represents the type (e.g., number, text) and the format of the received data; (iv) the data itself; (v) the reception timestamp. Such a behavior towards sources and incoming data is regulated by the rules established by the policies.

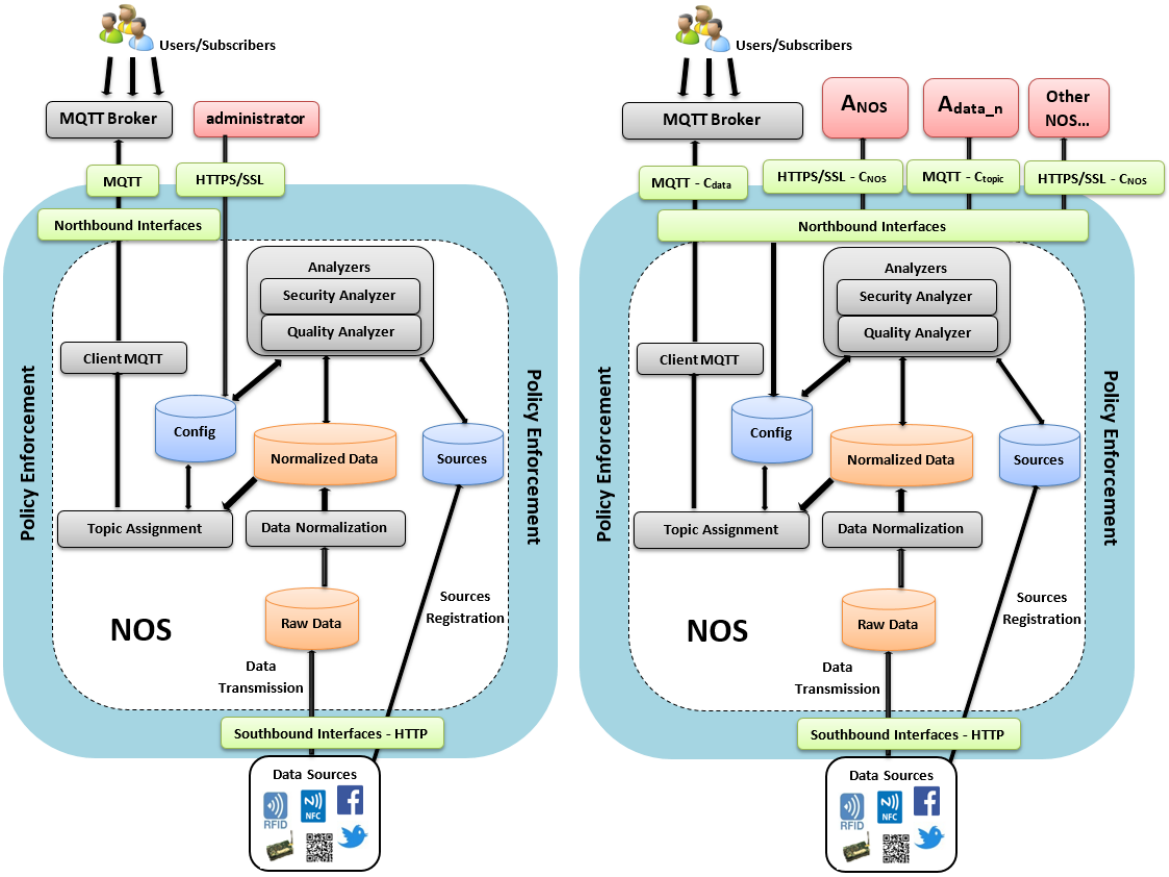
NOS initially puts the received data in the storage unit, named *Raw Data*; then they are periodically elaborated by the *Data Normalization* and *Analyzers* phases, in order to obtain an uniform representation and add useful metadata. More in detail, firstly data are pre-processed by the *Data Normalization* module, which put them in another storage unit, named *Normalized Data*, in the format specified in Figure 2. Then, a second module, consisting of a set of *Analyzers*, periodically extracts the normalized data from the storage unit *Normalized Data* and annotated them with a set of metadata (i.e., a score in the range [0:1]) for each security and data quality property considered. In particular, as regards the security, four requirements are evaluated: (i) data confidentiality; (ii) data integrity; (iii) privacy of the data sources; (iv) robustness of the authentication/authorization mechanisms adopted by the data sources. While, concerning data quality requirements, the following properties are investigated: (i) data accuracy; (ii) data precision; (iii) information timeliness; (iv) information completeness.

Note that the rules used for the assessment of such security and quality levels are dynamically configured at runtime by system administrators connecting remotely to NOS without the need to re-start NOS services. These rules are stored in a proper format in another NOS storage unit, named *Config*, and really represent a kind of policy, therefore they are treated as described in Section IV.

What generally happens is that *Analyzers* queries the *Config* storage unit in order to know which actions they have to undertake on data. We remark that the assignment of security and quality scores allows the users or applications, interested in receiving the data provided by NOS, to filter them according to personal or business preferences. This makes our approach extremely flexible and able to adapt to very different application scenarios where users may require diverse purposes in using the IoT information.

As regards the algorithms used for security and data quality assessment, they are not object of this work, but are deeply discussed in another work by the authors, to which we refer for further details [4]. They are also applied under specific policies and, as a consequence, properly enforced.

2) NOS northbound interfaces: As already presented in [4], once data are processed by *Analyzers*, they are ready to be sent to users/applications via the publish&subscribe system, made



(a) (b)

Fig. 1: NOS architecture (a) previous (b) new.

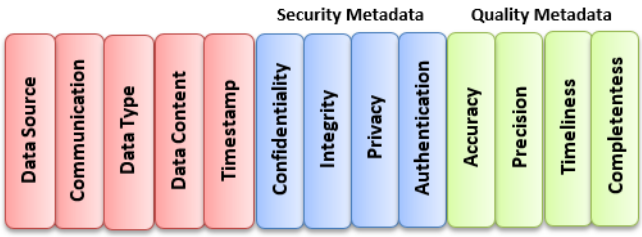


Fig. 2: NOS data format

available by NOS northbound interfaces, which are based on the well-known MQTT protocol [24], recently standardized by OASIS foundation. Note that MQTT is a simple and lightweight protocol with low overhead; such features make it suitable for constrained environments, as often happens in IoT application domains.

Users and applications must previously register themselves to NOS and they are provided with credentials useful for access the system by means of a proper interface, as specified in Section V.

Each NOS has a module in charge of assigning the corresponding topics to processed data (module *Topic Assignment* in Figure 1). Such an assignment depends on the policies applied for each specific application domain. In general, topics consist of one or more level separated by a forward slash (e.g., the temperature information of a sensor with identifier *sensorId* could be represented by the topic *sensor/temperature/sensorId*), thus generating a logical information structure, like in any file system. Then, the MQTT client (*Client MQTT* in Figure 1) publishes messages under the specified topics to an MQTT broker (*MQTT Broker* in Figure 1). Interested subscribers (i.e., users or applications) can register for specific topics at runtime by means of proper requests to NOSs; they can also dynamically subscribe and/or unsubscribe to the topics themselves during the time. All notifications are mediated by the broker, which is responsible to dispatch the events from the publishers to all the interested subscribers, in order to prevent the publishers (i.e., NOSs) needing to synchronize with subscribers, thus avoiding a continuous polling.

3) Enforcement Framework: NOS has been integrated with a basic policy enforcement framework in [6]. It includes: a Policy Enforcement Point (PEP), which is the point that intercepts the requests of access to resources from users/applications, 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.

Note that, in our case, where communication is based on the MQTT protocol, all requests by users or applications are handled via the MQTT broker, as sketched in Figure 3. The architecture underlying the enforcement framework may comprise one ore more NOS and a huge amount of nodes, which act as data sources, and users, which act as data consumers (either directly or mediated by other registered applications). Each NOS includes a PEP, a PDP and a PAP, while each device has an application representing an interface between the device itself and NOSs.

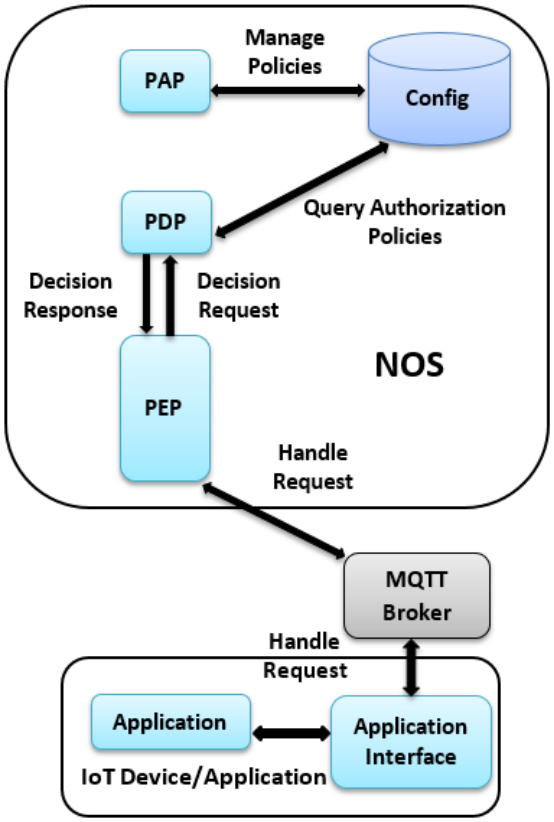


Fig. 3: NOS enforcement framework

With this regard, policies should be expressed with a proper interoperable specification language, able to be flexible enough to represent the heterogeneous analyzed contexts both in a general-purpose and in a customizable way. To this end, the syntax of the policies managed by the enforcement framework is structured hereby in three main components, representing, respectively: (i) the input attributes, owned by the entity to which the policy should be applied and used for evaluating the policy itself; (ii) the tasks to be executed on the provided inputs to assess the policy; (iii) the returning values of the policy assessment, which define whether the request shall be accepted or not. Once pointed out such a representation, a formalism for the practical deployment of the established policies have to be chosen. In this work, as in [6], policies are represented using JSON format, since it is perfectly compatible for the integration with the used database management system (i.e., *MongoDB*), specified in Section V.

Concerning policy inputs, we just mentioned the use of attributes. This is due to the fact that he access control model considered in this paper, as in [6], is the Attribute Based Access Control (ABAC) [25]. It concerns the presence of both the subjects, who want to access or to provide the resources, and the objects (i.e., data), which represent the resources, described by means of specific attributes, just used for policies definition. Attributes can be based on the metadata fields natively supported in our data representation and control rules can be defined and dynamically configured according to the specific needs of the application domains. As previously introduced, each user/application has to complete a registration phase before interacting with NOSs; during this phase, a set of attributes is assigned to each user on the basis of his/her role in a certain context (e.g., a manager or an employee of a financial company should have different attributes for accessing the resources of the company itself). The active policies will be based on such attributes. In particular, sub-sets of policies, related to the data of the same company/organization or disjoint sets of policy of data belonging to different companies/organizations, are both allowed.

B. New NOS architectural components

With respect to the previous NOS architecture just presented [6] [4], relevant changes mainly concern the introduction of the following functionalities and interfaces:

- The definition of new functionalities for an efficient policies' management, also considering policies' changes during the time. More in detail, new mechanisms are provided in order to ensure that all NOSs share the same policies, established on the basis of the involved application domains, at each time. In fact, in the previous version of NOS, such an aspect is treated in a trivial manner: an HTTPS connection is used by an external administrator to add/update/remove the policies, limiting the action of the enforcement framework to a single NOS. Hence, no attention is paid on efficient distribution, synchronization and compliance among different NOSs. Instead, such an important functionality is provided in the new version of NOS middleware, proposed in this paper, where NOS' capabilities are expanded in order to supervise the interactions among a NOS and other NOSs, or the IoT system's administrator, or the administrators of the resources provided by data sources. Hence, the new extension provides efficient mechanisms for synchronization among NOSs regarding the active rules.
- The introduction of new communication northbound interfaces. As emerged in Section III-A2, the problem of synchronization within the MQTT publish/subscribe system is solved by the used technology itself, which is able to handle the notifications in an efficient way. Instead, with regards to NOSs' policy configuration, new interfaces are introduced in the new version, as shown in Figure 1(b):
  - A_{NOS}: northbound interface is used for communications with the NOSs' administrator A_{NOS}, via HTTPS/SSL

- A_{data_n}: northbound interface is used for communications with the administrators A_{data_n} of the resources, provided to NOSs by the data sources, via MQTT
- *Other NOS*: northbound interface is finally used for commucations with other NOSs. In fact, different NOSs must be able to securely exchange useful information about adding/removing/updating the active policies enforced by the framework installed on each NOS by means of HTTPS/SSL connection.

Note that, in the previous version of NOS, a simple HTTP/SSL connection were provided for basic interactions with the administrator. More accurate details will be provided in Section IV.

IV. NOS Policy Synchronization

Users and applications adopts the MQTT communication protocol mediated by a broker in order to obtain data from NOSs system, as introduced in Section III. In particular, a user/application logs on a service provided by IoT system itself (e.g., an application running on user devices - smartphone, tablet, pc, etc.) and interacts through a proper GUI; as a consequence, a session is opened, during which the user/application can request for the services provided by NOS on the basis of the accessible resources. The resources can be accessed on the basis of the policies defined within NOS enforcement framework, in the format specified in Section III. Moreover, the MQTT broker has to interact with the underlying PEP on NOS in order to establish which subscriptions to accept or deny, and enforce the current subscriptions themselves.

In NOS system, policies distribution, update and synchronization tasks may be done by runtime configurations through the *Config* storage unit, which is in charge of dynamically update the local PAP of each NOS. As a consequence, each NOS applies the policies currently specified in *Config* on the incoming data and on the user/application requests, via the MQTT broker (as shown in Figure 3). Hence, it is essential that all NOSs always share the same set of policies.

Two principal aspects have to be clarified: (i) which kinds of policies can be defined within NOS system and (ii) which entities decide how and when such policies have to be applied on data/requests. Firstly, as regards the set of policies, we distinguish them into two groups:

- P_{NOS}: policies related to NOS behavior, which includes, for example, the methods used for assessing the security and data quality properties (i.e., the behavior of the *Analyzers*), the format established for the normalized data (i.e., the behavior of the *Data Normalization* phase), the rate of data processing, etc
- P_{data_n}: policies concerning the access to the data themselves, which are strictly related to the topics' assignment (i.e., which users or applications are allowed to access to some topics on the basis of their attributes assigned at NOS registration phase); such policies can be divided in *n* groups, where *n* is the number of organizations or companies in charge of managing (or that own) the data provided by the sources. The number *n* may vary over the time, since organizations or companies may join or leave the IoT network (i.e., they can start or stop to send data to NOSs at every time by means of their available data sources). It is worth to remark that the scope of the proposed policy distribution and synchronization schema is tailored not only to a single scenario, but aims to be adopted by many companies, each one requiring customized and shared policies e.g., a vehicle traffic notification system, a weather service, and so on. For these reasons we targeted our solution to more general organizations and we do not refer to user roles concerning a single application domain.

Therefore, multiple entities are will be associated with different and independent policies. All of them are conceived as system administrators, that are mainly supposed to be independent from each other; for example they may belong to different companies or organizations. More in detail, we have:

- A_{NOS}, which represents the administrator of the set of policies P_{NOS}; such an administrator is unique and is responsible for the entire IoT system administration, since NOS behavior cannot be managed by parties involved in data provision (e.g., external companies could foster their business interests)
- A_{data_n}, which represents the *n* administrators of the resources; they define the policies to be applied to filter the access to the data processed and published by NOSs. Note that each A_{data_n} may belong to a diverse company or organization and is independent in terms of provided resources and required policies. Also a sort of hierarchy (e.g., sub-sets of policies) is admitted for the policies belonging to the same company or organization.

Once made such distinctions among policies and administrators, the main challenge is how to manage in an efficient way the policies distribution to each NOS. The principal issues to be faced regard the following aspects: (i) each NOS belonging to the IoT system has to be synchronized with each other in order to behave in the same way when treating a given type of data; (ii) policies may need to be updated over the time due to the provision of new data to NOSs or to changes within the companies/organizations resources' disclosure.

The simplest solution would be the adoption of a central authority at the head of both A_{NOS} and A_{data_n}, able to intercept all the requests to add, update or remove policies and, then, propagate such changes to all NOSs, for example via a secure HTTPS or SSL channel. However, this may represent a bottleneck and does not ensure the real synchronization of all NOSs. In particular, a system of acknowledgments should be integrated, in order to let the central authority aware of the correct reception of the changes by all NOSs. Obviously, such a solution is not suitable for the wide IoT scenario, due to the huge number of information and policies involved. Note that the introduction of a central authority would prejudice the principal function of NOS, which is that of moving the data processing closer to the data sources without delegating the activities to a central entity, in charge of collecting and manage all the information transmitted within the IoT system.

Hence, a totally distributed approach is adopted, which

TABLE I: Adopted acronyms

Acronym	Meaning
P_{NOS}	set of policies about NOS behavior
P_{data_n}	set of policies about the access to data
A_{NOS}	administrator of P_{NOS}
A_{data_n}	administrators of P_{data_n}
C_{data}	MQTT channel for pub/sub to/for users/applications
C_{topic}	MQTT channel for communications with A_{data_n}
C_{NOS}	HTTPS/SSL channel for communications among NOSs and A_{NOS}

exploits the MQTT connection protocol already integrated into NOS. The scope is to create different virtual channels in order to separate the management of the sets of policies P_{NOS} and P_{data_n} and to allow efficient transmission among NOSs and administrators. Such a separation is made at the level of northbound NOS interfaces, presented in Section III. In particular, looking inside the northbound NOS interfaces, as depicted in Figure 1, the following virtual channels are introduced:

- C_{data}, which is, in the previous version of NOS [4], the only MQTT channel used for publishing and notifying the processed data to interested users/applications; this means that it is the only channel inherited by the previous NOS version
- C_{topic}, which aims to allow the administrators of the group A_{data_n} to add/remove/update the policies related to the information belonging to the topics under their authority. This channel is based on MQTT as the previous one (i.e., C_{data})
- C_{NOS}, which forms a sort of private channel among NOSs themselves and A_{NOS} used for security purposes, as clarified in the following. In fact, such a channel is used for checking the policies synchronization and is based on a secure HTTPS/SSL protocol.

Two aspects have to be remarked: (i) C_{data} and C_{topic} overlap on a practical level, since both exploit the existing publish&subscribe mechanism, but they are separated from a conceptual point of view, as we will further detail; (ii) all the interactions are mediated by the broker; in our case we assume to have one broker to which a various number of networked NOSs is connected.

Table I summarized the adopted acronyms along with their meaning. Instead, Figure 4 goes inside the northbound interfaces presented in Figure 1 and outlines the proposed schema including all the involved entities. Blue colour indicates the communications which happen within the virtual channel C_{data}; in fact the blue uni-directional arrows represent the route of the information processed by NOSs to their publication to the MQTT broker towards the final notification to the end users/applications. The access to such data is regulated by the policies applied to the topics associated to the information themselves, which are managed by the enforcement framework running on each NOS (see Section III-A3).

How the policies are propagated to NOSs is handled by the C_{topic} virtual channel. In detail, all NOSs agree, in a preliminary phase (i.e., before NOSs deployment), with the A_{NOS} administrator on a particular topic t (e.g., *NOS/policy*); this is used for publishing the policies related to the adding/removing/updating operations, both by A_{NOS} itself or

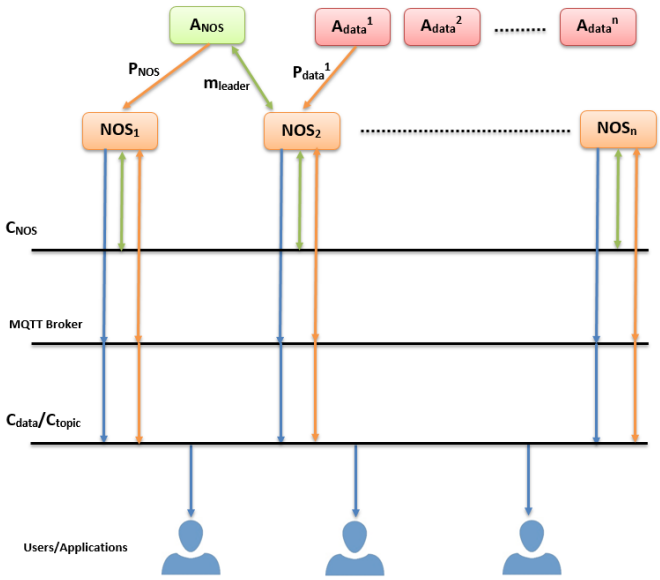


Fig. 4: Policy management schema

by the A_{data_n} administrators. The access to t, which may be structured in a proper hierarchy in order to obtain full expressiveness and flexibility, is restricted to NOSs, which are the only entities notified of such a kind of information and able to know its content (which is transmitted in an encrypted way, by means of a proper algorithm agreed by NOSs and A_{NOS}). Therefore, users and applications are prevented from access to the policies, despite the same "physical" channel (C_{data} corresponds to C_{topic}) is used for the notification of the data to which they are subscribed. More in detail, in Figure 4, the orange arrows show the flow of policies propagation, which consists of the following steps:

1) A_{NOS} for the P_{NOS} policies and A_{data_n} for the P_{data_n} policies communicate to one NOS (chosen randomly at each time or on the basis of information about NOS locations), via a secure HTTPS/SSL channel, the existence of a new policy for a certain resource or the update of an existing one. Note that such administrators may also expose RESTful services to NOS for policy transmission
2) NOS stores or updates such a policy in the *Config* storage unit. As a consequence, the enforcement framework automatically starts to modify its behavior towards the involved resource accordingly. Note that this is an important feature of the proposed NOS modular architecture, which allows to save time and computational effort, since the system is able to be re-configured without restarting or modifying the modules themselves
3) NOS has to publish the new policy or the update to the MQTT broker under the agreed topic *t*
4) All other NOSs are allowed to access the information under the topic *t* and, then, are able to store or update the policy.

From a practical point of view, all NOSs are subscribed to the topic *t*; therefore they are notified at each new incoming event related to *t*. In fact, the orange arrows are bi-directional,

since a NOS may either publish an information to the MQTT broker or be notified about a new one. Obviously, such data are not transmitted in clear over the network, but are ciphered by means of a proper encryption schema (e.g., RSA, PKI, etc.) shared by all NOSs and established, for example, before NOS deployment, as said above. Figure 5 shows the transactions taking part during such a phase, for the NOS informed of the adding/removing/updating of a P_{data_n} policy. The figure also specifies the activities taking place before the deployment and the communication channels used.

Summarizing, exploiting the MQTT publish&subscribe protocol potentialities, NOSs are able to manage and update the policies of heterogeneous companies or organizations in a distributed and lightweight way, without the need of a central coordinator, which may represent a single point of failure. Moreover, note that no additional modules have been added to the previous NOS architecture [4], since, in order to provide the new functionalities just described, the existing MQTT mechanism has been directly used. Furthermore, such a mechanism is transparent with respect to the different policies applied to data belonging to heterogeneous domains.

The last feature to be considered strictly regards the policy synchronization; in fact, the IoT system, and, in particular, the A_{NOS} administrator, has to be able to know if all NOSs share the same policies at a certain interval time. To this end, the C_{NOS} channel is introduced. Periodically, A_{NOS} selects a NOS as a leader. This choice can be made by means of one of the well-known algorithms available in literature for leader selection [26], which represent a well-investigated subject in distributed systems. For our work, we consider the case in which A_{NOS} directly selects the leader (i.e., as a central server), and we leave the case of NOS self-leader election as a future extension. Note that we assume that NOSs are trusty.

Going inside the synchronization process, it consists of the following steps, highlighted in Figure 4 with green arrows:

1) A_{NOS} elects a leader L_{NOS} among the networked NOS by sending a message m_{leader} only to the selected NOS via the secure HTTPS/SSL channel, just previously presented; the leader should change periodically in order to increase the robustness of the IoT system in case of link failure
2) When the leader L_{NOS} receives a policy notification, besides performing the publication to the MQTT broker, it sends to the other NOSs an advertising message m_{id} through the secure C_{NOS} channel, where *id* represents a progressive identifier chosen by L_{NOS} for identifying the policy currently under the synchronization process:

$$\begin{aligned} &\{ \\ &\quad \text{“}NOS\text{”} : L_{NOS}, \\ &\quad \text{“}policyId\text{”} : id \\ &\} \end{aligned} \tag{1}$$

3) All NOSs has to reply to such an advertise with a simple acknowledge message r_{id}, using the secure C_{NOS} channel:

$$\begin{aligned} &\{ \\ &\quad \text{“}NOS\text{”} : NOS_{id}, \\ &\quad \text{“}policyId\text{”} : id \\ &\} \end{aligned} \tag{2}$$

4) L_{NOS} verifies the reception of r_{id} from all NOSs; if one or more responses lack, then L_{NOS} sends a report to A_{NOS}, which could take some countermeasures (e.g., do not consider the no-synchronized NOS for leader selection).

Figure 6 shows the transactions taking part during the synchronization phase from the perspective of the NOS that acts as a leader. Figure 6 also specifies the communication channels used.

Note that some important parameters of the synchronization process have to be considered and customized depending on the number of NOS and connected sources and on the data load. Such parameters include: (i) the rate of leader selection; (ii) the time waited by the leader for the responses by NOSs. These fall into the issue of determining the global state of a system in a certain interval time, which is also a well-investigated field in distributed environments [27].

Finally, it is important to remark that the proposed schema is independent from the kind of data managed by NOSs as well as from the specific application domains. In fact, as confirmed by the presence of multiple A_{data_n}, policies related to different realms may coexist.

V. Experimental Evaluation

The solution, presented in Section IV, has been implemented in simple, but yet real prototype, whose code is released as open source under a permissive license[1]). The following technologies have been adopted: (i) *Node.JS* platform[2] for NOS development; (ii) *MongoDB*[3] for database management; (iii) Mosquitto[4] for the publish/subscribe mechanism. All the modules, including the enforcement framework, interact among themselves through *RESTful* services. In such a way, it is possible to add new modules or update/remove the existing ones without stopping and re-starting the entire NOS, since they work in a parallel (non-blocking) manner. In the remainder of this section, the experimental setup along with the performance evaluation of the proposed solution are detailed.

In our test environment, shown in Figure 7, four NOSs run on Raspberry Pi platforms and receive in real time the data provided by six sensors at the meteorological station of the city of Campodenno (Trentino, Italy). The following information are sent as open data in *JSON* format: temperature, humidity, wind, energy consumption, air quality. These are mapped to proper topics, such as *campodenno/sensor1/temperature*, *campodenno/sensor2/humidity*, and so on. Also, a set of policies is defined accordingly. A laptop is employed to emulate the behaviour of this set of sources, basically reading data from the aforementioned feeds and sending them to NOSs as if they came from six different nodes. The data rate (that is: how often NOSs queries the data sources to fetch data) is fixed to 10 and 20 packets per second. Laptop and Raspberries Pi communicate via a WiFi network. Users can connect to

[1] https://bitbucket.org/alessandrarizzardi/nos
[2] http://nodejs.org/
[3] http://www.mongodb.org/
[4] http://mosquitto.org

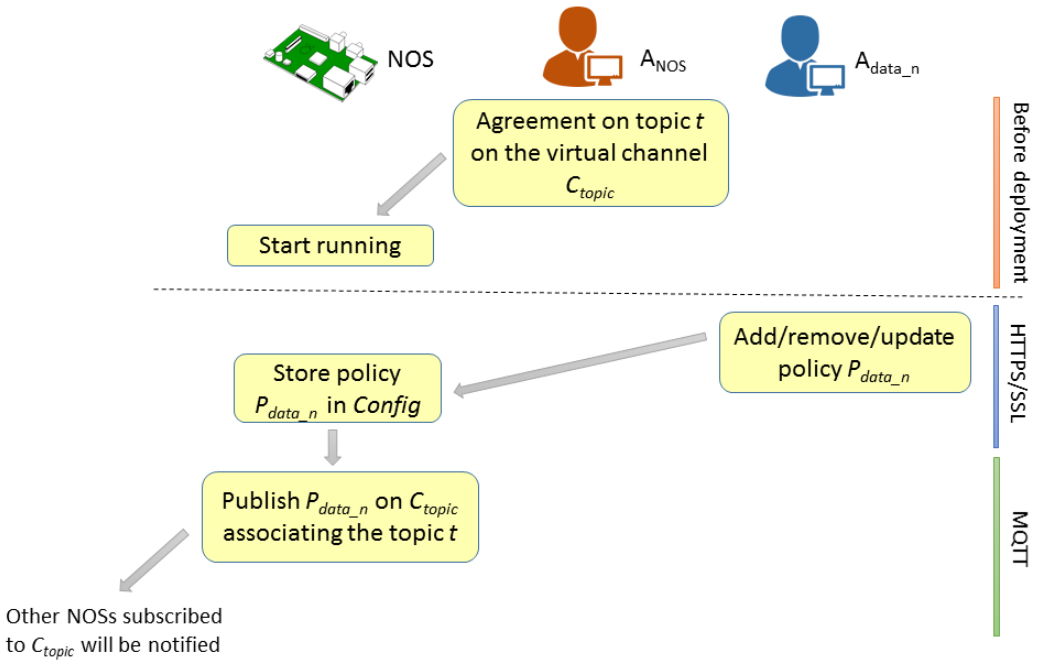


Fig. 5: Add/remove/update of a policy: schema

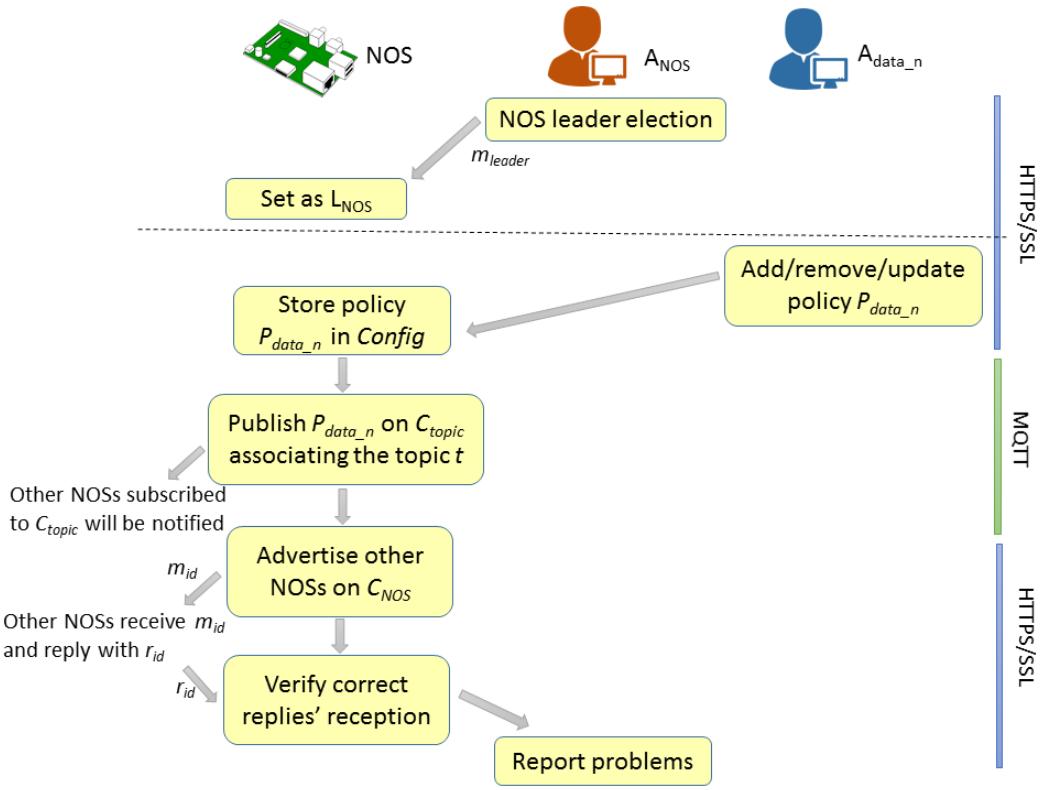


Fig. 6: Policy synchronization: schema

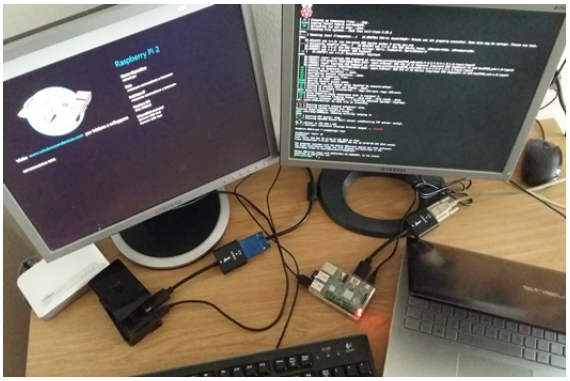


Fig. 7: Simulation setup

TABLE II: Test-bed parameters

Parameter	Value
Number of data sources	6
Number of NOSs	4
Number of A_{NOS} administrators	1
Number of A_{data} administrators	1
Data rate	10 and 20 pck/sec
Policy update rate	from 1 to 5 policy/min
Observation time	6 h

a public IP address by means of their computers, tablets or smartphones, in order to interact with the IoT system and visualize on their browser the gathered data (to which they are subscribed), through a proper application interface. Our test-bed also simulates the behavior of a A_{NOS} administrator; while one A_{data_1} administrator is considered, since the data belong to a unique application context. Finally, the rate of policy update varies in a random range from one 1 to 5 minutes. Table II summarizes the parameters used in the test-bed, for an observation time of 6 hours.

A. Analysis

The execution time and CPU load, spent for the execution of the mechanism proposed in Section IV (i.e., policies' update and synchronization), have been evaluated by means of a six-hours running. Figures 8 and 9 show, through the box-whiskers representation, the mean results obtained from the four NOSs used for our test-bed. Both Figures 8 and 9 reveal promising outcomes with respect to the dimension of the test-bed, since both execution time and computational effort are low and stable over the time. Note that the results are not affected in a relevant way by the change in the data rate.

Another interesting metric concerns the update time required by NOSs to activate/modify a policy. During the observation time of 6 hours. As stated at the beginning of the section, the rate of policies' update is set in the range from 1 to 5 minutes. The box-whiskers representation of Figures 10 reveals that also such results are acceptable in relation to the test-bed dimension. It also emerges that the time required for policies' update is influenced by the arrival time of the data; it is reasonable because NOSs' computational resources are also employed for data processing activities.

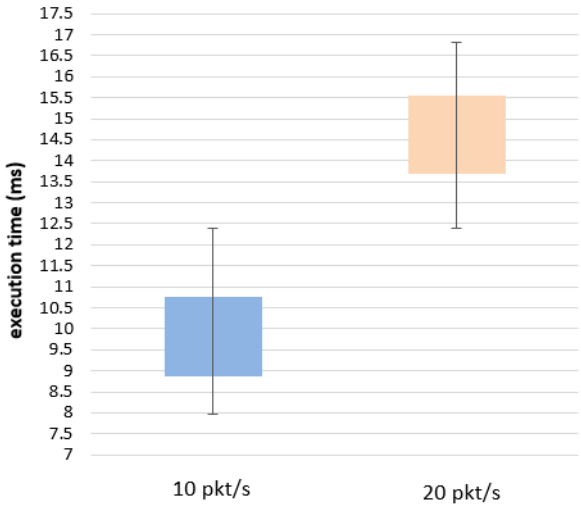


Fig. 8: Whiskers-box diagram: execution time (in ms) for two different traffic configurations, expressed in packets/s

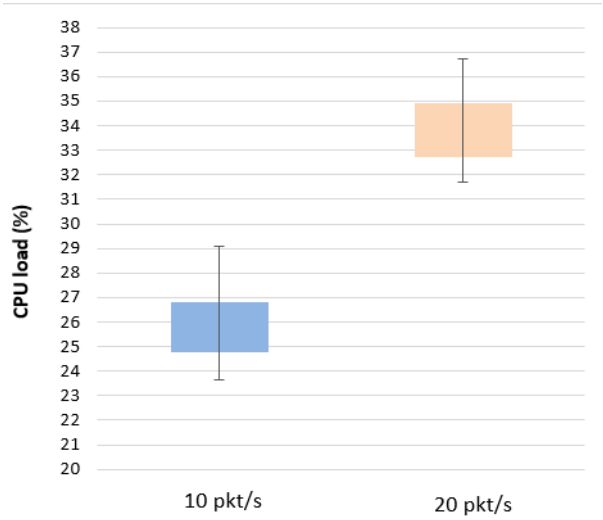


Fig. 9: Whiskers-box diagram: computational load (%) for two different traffic configurations, expressed in packets/s

Certainly, the presented example is a very simple application involving four NOSs in a context characterised by the analysis of real-time data, but it demonstrated the viability of the proposed schema to meet its requirements (i.e., enforcement of the defined policies, secure and efficient synchronization among multiple NOSs) with acceptable performance indices. As a future development, we aim to test our solution in a wide IoT context (i.e., more NOSs and data sources), possibly including different application domains (i.e., more A_{data_n} administrators) and, consequently, more complex policies (e.g., hierarchical structures). Only in this way it will be possible to definitely demonstrate the viability of the proposed approach.

Final considerations regard the robustness of the proposed solution with respect to violation attempts. In order to clarify the behavior of the IoT system considered in this paper, we

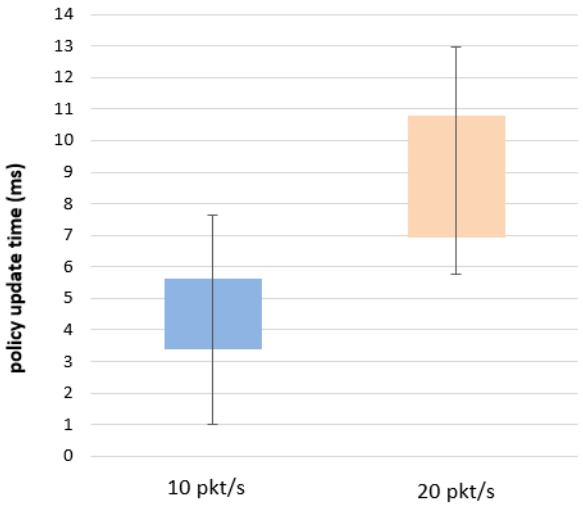


Fig. 10: Whiskers-box diagram: policies' update time (in ms) for two different traffic configurations, expressed in packets/s

point out the possible situations that can verify:

- Discovering of the topic *t* on the C_{topic} virtual channel by an external malicious entity: such information is agreed among NOSs before deployment, as shown in Figure 5, therefore they cannot be sniffed or eavesdropped during network communications
- Discovering of the policy P_{NOS} and P_{data_n} updates by an external malicious entity: they are transmitted to NOSs via a secure HTTPS/SSL channel and then retransmitted over the C_{topic} virtual channel to notify other NOSs in an encrypted way. It is worth to remark that credentials are installed on NOSs before deployment
- Recognition of the NOS leader: the leader election is done via a secure HTTPS/SSL channel and the leader periodically changes.

Nevertheless, sophisticated attacks could still make the system vulnerable from other perspectives (i.e., Denial of Service attacks, brute force attacks for credentials discovery, and so on). We are also working on solutions to protect the IoT system from such a kind of attacks [28].

VI. Conclusions

In heterogeneous and fully distributed environments, information flow must be controlled under well-defined policies, in order to manage the access to resources, which may be confidential, and prevent possible violation attempts. In order to address such an issue, in this paper, a policy distribution and synchronization schema has been proposed. It is based on an efficient and lightweight technique for the propagation and synchronization of policies across different domains, tailored to an IoT context application. The presented solution has been integrated within a distributed IoT middleware platform, named NOS, which uses the MQTT protocol for information exchange, fitting the needs of flexibility of both large-scale and constrained environments. The feasibility and the performance of the proposed approach have been evaluated by means of a simple yet real prototypical implementation. In particular, we tested the mean execution time, the CPU load of NOSs, and the time spent for policies' update. We also conducted a brief robustness analysis of the proposed solution. In the next future, we will focus on the deployment of the middleware along with the policy management and synchronization mechanism in a wider IoT context involving different application domains, such as building automation, healthcare structures, and environmental monitoring, in order to test the effective scalability of the proposed solution.

References

[1] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: Perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, 2014.

[2] S. Sicari, C. Cappiello, F. D. Pellegrini, D. Miorandi, and A. Coen-Porisini, "A security-and quality-aware system architecture for internet of things," *Information Systems Frontiers*, pp. 1–13, 2014.

[3] S. Sicari, A. Rizzardi, A. Coen-Porisini, and C. Cappiello, "A NFP model for internet of things applications," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*. Larnaca, Cyprus: IEEE, 2014, pp. 265–272.

[4] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini, "A secure and quality-aware prototypical architecture for the internet of things," *Information Systems*, vol. 58, pp. 43–55, 2016.

[5] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[6] S. Sicari, A. Rizzardi, D. Miorandi, C. Cappiello, and A. Coen-Porisini, "Security policy enforcement for networked smart objects," *Computer Networks*, vol. 108, pp. 133 – 147, 2016.

[7] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, 2013.

[8] K. J. Knapp, R. F. M. Jr., T. E. Marshall, and T. A. Byrd, "Information security policy: An organizational-level process model," *Computers & Security*, vol. 28, no. 7, pp. 493 – 508, 2009.

[9] T. Goovaerts, B. D. Win, and W. Joosen, "Infrastructural support for enforcing and managing distributed application-level policies," *Electronic Notes in Theoretical Computer Science*, vol. 197, no. 1, pp. 31 – 43, 2008.

[10] T.-Y. Chen, "Knowledge sharing in virtual enterprises via an ontology-based access control approach," *Computers in Industry*, vol. 59, no. 5, pp. 502–519, 2008.

[11] J. Padget and W. W. Vasconcelos, "Policy-carrying data: A step towards transparent data sharing," *Procedia Computer Science*, vol. 52, pp. 59–66, 2015.

[12] J. Singh, L. Vargas, J. Bacon, and K. Moody, "Policy-based information sharing in publish/subscribe middleware," in *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*. IEEE, 2008, pp. 137–144.

[13] J. Singh, D. M. Eyers, and J. Bacon, "Disclosure control in multi-domain publish/subscribe systems," in *Proceedings of the 5th ACM international conference on Distributed event-based system*. ACM, 2011, pp. 159–170.

[14] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody, "Role-based access control for publish/subscribe middleware architectures," in *Proceedings of the 2nd international workshop on Distributed event-based systems*. ACM, 2003, pp. 1–8.

[15] L. I. Pesonen, "A capability-based access control architecture for multi-domain publish/subscribe systems," *University of Cambridge, Computer Laboratory, Technical Report*, no. UCAM-CL-TR-720, 2008.

[16] A. Wun and H.-A. Jacobsen, "A policy management framework for content-based publish/subscribe middleware," in *Middleware 2007*. Springer, 2007, pp. 368–388.

[17] Z. Wu and L. Wang, "An innovative simulation environment for cross-domain policy enforcement," *Simulation Modelling Practice and Theory*, vol. 19, no. 7, pp. 1558–1583, August 2011.

[18] D. Ferraiolo and V. A. ans S. Gavrila, "The policy machine: A novel architecture and framework for access control policy specification and enforcement," *Journal of Systems Architecture*, vol. 57, no. 4, pp. 412–424, April 2011.

[19] J. Rao, A. Sardinha, and N. Sadeh, "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 40 – 56, 2009.

[20] ——, "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 40–56, January 2009.

[21] V. Kapsalis, D. Karelis, L. Hadellis, and G. Papadopoulos, "A context-aware access control framework for e-service provision," in *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*, Dec 2005, pp. 932–937.

[22] C. Bertolissi and M. Fernndez, "A metamodel of access control for distributed environments: Applications and properties," *Information and Computation*, vol. 238, pp. 187–207, 2014.

[23] A. Rizzardi, D. Miorandi, S. Sicari, C. Cappiello, and A. Coen-Porisini, "Networked smart objects: Moving data processing closer to the source," in *2nd EAI International Conference on IoT as a Service*, Oct 2015.

[24] "Ibm and eurotech, "mqtt v3.1 protocol specification"," http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/ mqtt-v3r1.html.

[25] V. Goyal, O. Pandey, A. Sahai, and 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.

[26] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[27] K. M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[28] S. Sicari, A. Rizzardi, D. Miorandi, and A. Coen-Porisini, "Reacting to denial of service attacks in the internet of things," *Technical Report*, 2017.