REATO: REActing TO denial of service attacks in the Internet of Things

Sabrina Sicari*‡, Alessandra Rizzardi*, Daniele Miorandi§, Alberto Coen-Porisini*
*Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, via G. Mazzini 5 - 21100 Varese (Italy)
§U-Hopper, via R. da Sanseverino 95, 38121 Trento, Italy
‡Corresponding author
Email: {sabrina.sicari; alessandra.rizzardi; alberto.coenporisini}@uninsubria.it, daniele.miorandi@u-hopper.com

***Abstract*—Denial of Service (DoS) attack represents until now a relevant problem in Internet-based contexts. In fact, it is both difficult to recognize and to counteract. Along with the adoption and diffusion of Internet of Things (IoT) applications, such an issue has become more urgent to solve, due to the presence of heterogeneous data sources and to the wireless nature of most communications. A DoS attack is even more serious if not only the data sources are under attack, but also the IoT platform itself, which is in charge of acquiring data from multiple data sources and, after data processing, provide useful services to the interested users. In this paper, we present a solution, named *REATO*, for actively and dynamically detecting and facing DoS attacks within a running IoT middleware. A real prototype has been realized in order to validate the proposed method, by assessing different relevant parameters.**

Keywords—Internet of Things, Security, Denial of Service, Attacks, Middleware, Prototype

I. Introduction

Many vulnerabilities affect applications based on the Internet of Things (IoT) paradigm. This is mainly due to the wireless nature of the communications taking place among the involved entities, which can be represented by heterogeneous devices, such as sensors, actuators, RFID, NFC, and so on. As a consequence, an IoT-based system can be object of various kinds of attacks to the network resources, to the integrity and confidentiality of the transmitted data, or to the privacy of the users or companies that exploit IoT services.

Note that security is one of the main obstacles towards the wide adoption and diffusion of IoT applications [1]. Until now, little attention has been paid towards the identification and counteraction of specific kinds of attacks pursued in IoT contexts, but several solutions are available for wireless sensor networks (WSNs) [2] and mobile ad hoc networks (MANETs) [3], as presented in the state of the art section. However, such approaches should be revised with respect to the dynamicity and heterogeneity of the IoT environment.

In order to guarantee proper levels of security, an IoT system should provide authentication systems for: (i) the entities that send data to the IoT platform; (ii) the entities that require information to the IoT platform in the form of services; then it should also provide encryption mechanisms for ciphering the transmitted data and, to do this, adopt an effective key management system. Finally, policies able to regulate the access to the available resources should be defined and enforced.

However, besides such countermeasures would improve the resilience of the IoT system, particular attention must be directed against Denial of Service (DoS) attack, which is widely considered to be an important security issue [4]. There are various kinds of DoS attack, such as flooding the network with useless traffic, resource power exhaustion, and jamming the signal [5]. In fact, they aim at making devices or network resources unavailable by preventing the communications and/or compromising the traffic or wasting resources and, thus, blocking the access to the services provided by the IoT platform. Note that generally no variation is made on the transmitted data by such a kind of attack.

DoS attacks can be initialized from a remote place by means of proper tools and commands. In the IoT, the situation worsens if not only data sources are attacked but if also the IoT platform itself is picked on. If the IoT platform, that acts as a distributed server, is saturated, for example, with external requests, then it cannot respond to legitimate ones in a reasonable time (i.e., because communications are slowed by the attack). On the other side, storage and computational resources of both devices and IoT platform will be consumed, thus generating further damages [6].

Hence, it is fundamental both to promptly detect DoS attacks and to be able to recover the network's functionalities in a short time. In this paper, we propose a technique, named *REATO*, for recognizing and counteracting DoS attacks in the IoT context by means of a working middleware that receives open data feeds from remote sources in real time. DoS attacks are then simulated within the presented IoT platform in order to prove the effectiveness of the proposed solution by means of a real prototype. Relevant parameters are assessed in the experiments, in order to reveal the potentialities of the presented approach.

The paper is organized as follows. Section II presents some relevant works about DoS attack's prevention/recognition in the fields of WSNs, MANETs, and IoT. Section III outlines the details of the IoT platform analyzed in this paper, pointing out the related vulnerabilities against possible DoS attacks. Section IV proposes the solution, named *REATO*, for dealing with the emerged issues; while in Section V concrete experiments are described in order to validate the proposed approach. Section VI ends the paper, giving some hints for future developments.

II. Related Work

There are very different available techniques to prevent and recognize DoS attacks. They also vary on the basis of the target environment. We present hereby some relevant solutions in WSNs, MANETs, Service Oriented Architecture (SOA) and IoT contexts, in order to provide a wide overview of the existing approaches and better clarify our innovative contribution.

The work presented in [7] is targeted to the prevention techniques of DoS attacks in WSNs. The same authors, in [8] propose a cooperative immune system that is an improvement of another existing immune system, named Co-FAIS. More in detail, it uses a fuzzy logic that exploits multiple learning parameters in order to improve the accuracy rate of DoS attack's detection.

A machine learning technique, based on Self Organizing Maps (SOM), is adopted in [9]. Relevant network parameters, such as the number of packets generated/sent/received/dropped, the CPU and memory usage, are considered for each node and analyzed over the time, in order to reveal anomalies in the nodes' behavior (for example caused by a DoS attack). Hence, a reputation score is associated to each node belonging to the WSN; a threshold is further determined by means of simulations for revealing possible network impairment and isolating compromised nodes.

In [10], a message observation mechanism (MoM) to detect and defence WSNs against the DoS attack has been designed. It is based on the spatio-temporal correlation and utilizes the similarity function to identify the content attack as well as its frequency. Moreover, MoM adopts re-key and re-route countermeasures in order to effectively isolate the malicious nodes.

Code dissemination is the process of propagating program images or related commands to sensor nodes in a WSN. This task is used to provide bug fixes or new functionalities after the WSN itself has been deployed. [11] presents a new approach for guaranteeing confidentiality as well as robustness against DoS attacks in multi-hop code dissemination protocols. In fact, in a multi-hop scenario, a sensor node is required to broadcast its program image when requested by its neighbours; however, an adversary could repeatedly send program image requests to its neighbours, making them re-broadcast the same program image until the residual energy of the request recipients is totally depleted. Moreover, without an adequate authentication mechanism, an adversary could also capture or impersonate a sensor node and inject a malicious program image into the WSN and, finally, controlling it. To cope with such issues, the authors proposed to use session keys derived from hashing data packets to encrypt the data packets themselves and to manage the re-keying process, without requiring further energy-expensive mechanisms. Moreover, they defined a technique, based on a mathematical model, which is able to evaluate the amount of code update requests. The proposed solution performs such an analysis taking into account the distance level of the nodes in the WSN with respect to the data collection point (i.e., the sink). The load of code update requests at a certain nodes' distance level will be considered as the threshold value for actuating proper countermeasures against a DoS attack.

[12] focuses on an emerging kind of attack, named path-based DoS (PDoS). It happens when an adversary inside or outside the network compromises a set of sensor nodes or aggregator nodes by flooding a multi-hop end-to-end routing path towards the sink with replayed or injected packets. Such a behavior causes an excessive power consumption for the involved nodes that leads to a quick death of a part of the WSN. In order to face the presented issue, the authors propose a recovery strategy both for compromised nodes and databases; the solution is based on the use of mobile agents, which are able to detect PDoS attacks.

In order to mitigate DoS attacks in a hierarchical WSN, in [13] a secure and self-enforcing scheme has been implemented. For ensuring a correct authentication of the involved entities (i.e., sensor nodes, cluster heads) and making the sink available only for legitimated ones, it exploits: (i) pseudo-random functions associated to each node before deployment; (ii) a derivative key established by the sink with the cluster heads and further derivative keys established by the cluster heads with sensor nodes.

Some works focus on the authentication issue. For example, [14] reviews the state of art of user authentication schemes for WSNs in terms of security (i.e., resilience towards many kinds of attacks) and computational overhead. Such an analysis revealed that mitigating DoS attacks still remains a big challenge in the field of WSNs, mainly due to the limited resources owned by the sensor nodes, which expose them to insidious threats.

In [15], a flooding authentication mechanism, based on the so-called "Information Asymmetry" model, has been proposed for WSN, in order to overcome DoS attacks. Here, the sink owns a global key pool; while each sensor node is only associated with a small number of keys belonging to the key pool. Therefore, the "asymmetry" is achieved in the sense that the key distribution between the sender (i.e., the sink) and the receivers (i.e., the sensor nodes) is asymmetric. Such a mechanism is coupled with: (i) multiple MACs (Message Authentication Codes), which are appended to each packet for sensor nodes' message authentication; (ii) the Bloom filter-based Message Authentication Protocol (BMAP) for recognizing false negative among the legitimate control messages.

As regard MANETs, several works make use of proper packets' authentication schemes in order to counteract various kinds of attack (e.g., wormhole, man-in-the-middle, DoS, impersonation). As an example, [16] proposes the HEAP protocol, which authenticates packets hop-by-hop by using a modified HMAC-based (Hash Message Authentication Code) algorithm along with a pair-wise secret hash key and drops any packets originated from outsiders.

Instead, even in MANETs, [17] introduces a detection technique against Distributed DoS (DDoS) attack. In order to classify the legitimate and the malicious packets and find abnormalities during the pre-attack phase, it statistical analyzes feature extraction, reduction of entropy, clustering technique and feature ranking.

Also with regard to DDoS attack, but in WSN's field, the work in [18] points out a solution based on dynamic source routing (DSR), that uses the analysis of the residual energy of sensor nodes as a factor for detection and prevention of DDoS attack itself.

Otherwise, very few solutions have been proposed until now in the IoT context. Among them, [19] presents a survey on DoS attacks that may be targeted to IoT environments, along with the evaluation of existing systems that try to detect and mitigate them.

In [20] and [21], strategies for preventing DDoS attack in IoT networks is proposed. The former exploits SOA as a system model and uses the Learning Automata (LA) concepts; while the latter makes use of an agent. The main drawback of such approaches is that they only serve for preventing the attack, but no countermeasures is provided to react against a successful one. Instead, the work in [22] presents a practical approach for countering the CyberSlam DDoS attacks in a SOA environment.

Also concerning SOA, the authors of [23] and [24] propose an intrusion tolerant approach and a prevention technique, respectively, for recognizing DoS attacks which exploit XML vulnerabilities (i.e., XML-based Denial of Service (XDoS)). Such solutions consist of monitoring activities (e.g., detection of malicious service requests, XML headers ad body validation, unexpected CPU consumptions), intrusion diagnosis and recovery.

Whereas, [25] presents a DoS detection architecture for 6LoWPAN protocol, widely adopted in IoT communications, which integrates an intrusion detection system (IDS) into the network framework developed within the EU FP7 project Ebbits [26].

In [27], the possible impacts of a specific class-amplified reflection DDoS attacks (AR-DDoS) against IoT are assessed. Such an approach is used to empirically examine the possible effects caused by a running attack over a controlled environment that includes IoT devices.

In [28], a scheme, named Data Authentication and En-route Filtering (DAEF), has been described for WSNs in the context of IoT. Signature shares are generated and distributed to the sensor nodes on the basis of a verifiable secret sharing cryptography and of an efficient ID-based signature algorithm, agreed with the sink in an initial phase of network deployment. The final aim is to defend the network against node compromise attacks as well as DoS attacks by reacting in the following way: an event report is generated from any part of the network in case of detected attack and broadcast to all the interested nodes. More in detail, the event report may be collectively generated, digitally signed, and forwarded to one or more of the nodes belonging to the network through multipath routing. In this way, the network should be able to activate proper countermeasures.

Summarizing, a lot of work has already been done in WSNs' field in order to face DoS attack; while the research is not mature enough in the IoT context. In fact, the proposed approaches have a limited target of application. What mainly lacks is a prevention and detection technique for DoS attacks which is integrated into a general-purpose IoT platform, able to exploit the system's functionalities in order to protect the legitimate connected devices as well as the platform itself. This is the aim of our work, as explained in the next sections.

III. IoT Platform and DoS Attack

In order to provide an effective solution, able to recognize and recover an IoT system in case of DoS attack, we should start with the definition of a proper IoT architecture along with the involved entities. Such an architecture has been defined by the authors in [29] and represents a flexible and cross-domain middleware, named *NetwOrked Smart object (NOS)*, tailored to the IoT environment.

NOSs are able to manage, in a distributed manner, the data provided by heterogeneous sources and evaluate, by means of proper algorithms [30], the security and data quality of the information, in order to allow the users to be aware of the levels of reliability and trustworthiness of the services gathered by NOSs themselves. NOSs also provide a lightweight and secure information exchange process, based on an authenticated publish and subscribe mechanism [31], using the MQTT protocol. Finally, an enforcement framework monitors the behavior of NOSs in order to guarantee the correct application of the established policies [32].

In the next sections, the architectural components of NOSs will be described, along with some details about their design and implementation. Moreover, the weaknesses against DoS attack of the presented platform will be clearly pointed out.

A. *Networked Smart Object Architecture*

Two main entities compose a typical IoT system: (i) the nodes, conceived as heterogeneous data sources (e.g., RFID, NFC, actuators, sensors, social networks, etc.) which generate data for the IoT platform; (ii) the users, who interact with the IoT system through services making use of such IoT-generated data, typically accessing them by means of a mobile device (e.g., smartphone, tablet) connected to the Internet (e.g., through WiFi, 3G, or Bluetooth technologies).

Therefore, proper interfaces for the communications of NOSs with the data sources (i.e., the nodes) and with the users have been defined.

In the former case, HTTP protocol is adopted for collecting the data from the IoT devices and for allowing sources' registration. In fact, NOSs deal 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 increasing the level of protection of their communications (encryption keys' distribution is made by the algorithms presented in [33]). The information related to the registered sources are put in the storage unit, named *Sources*. Instead, for each incoming data, both from registered and non-registered sources, the following information are gathered: (i) the kind of data source, which describes the type 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 content type (e.g., number, text) and the format of the received data; (iv) the data itself; (v) the reception timestamp.

Since the received data are of different types and formats, NOSs initially put them in the *Raw Data* storage unit. In such a collection, data are periodically processed, in a batch way, by the *Data Normalization* and *Analyzers* phases, in order to obtain a uniform representation and add useful metadata regarding the security (i.e., level of confidentiality, integrity, privacy and robustness of the authentication mechanism) and data quality (i.e., level of accuracy, precision, timeliness and completeness) assessment. Such an assessment is based on a set of rules stored in a proper format in another storage unit, named *Config*, and are detailed in [30]; it allows users who access the IoT data to directly filter by themselves the data processed by NOSs, according to their personal preferences.

While, in the latter case, Message Queue Telemetry Transport (MQTT) protocol[1] is used for disseminating the information to the interested users. To this end, a topic is assigned by NOSs to each processed data. In order to manage the access to resources on the basis of the assigned topics and on the active policies, the original MQTT protocol has been further extended with *AUPS (AUthenticated Publish&Subscribe system)* [31], and integrated with the enforcement framework [32]. It also concerns the adoption of a new component, named *Key Topics Manager (KTM)*, which is in charge of managing temporary keys for topics access control. Note that the keys are not fixed, but they come with an expiration timestamp, in order to improve the system's resilience towards malicious attacks (e.g., man in the middle attacks, replay attacks, password discovery). Figure 1 summarizes the NOS's components just introduced.

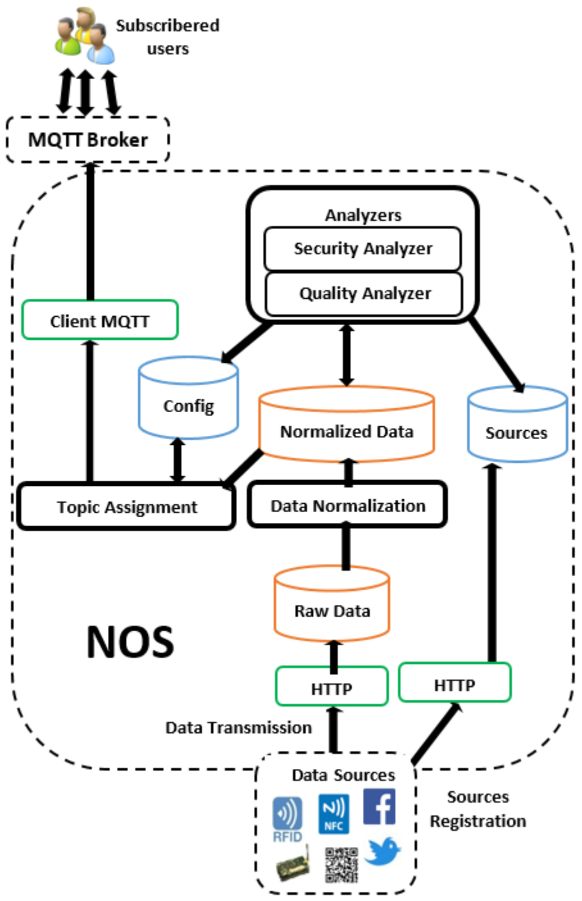

Fig. 1: NOS architecture

NOSs modules interact among themselves through *RESTful* interfaces; they have been implemented in a real prototype, which is openly accessible at https://bitbucket.org/alessandrarizzardi/nos.git. *Node.JS*[2] platform has been used for developing NOSs' core operations, *MongoDB*[3] has been adopted for the data management, and Mosquitto[4] has been chosen for realizing the open-source MQTT broker. To know more details about the implementation, we refer to [30].

[1]IBM and Eurotech, Mqtt v3.1 protocol specification, http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

[2]http://nodejs.org/

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

[4]Mosquitto, An open source mqtt v3.1/v3.1.1 broker, http://mosquitto.org

B. Vulnerabilities towards DoS attack

Once the IoT platform has been presented, we can proceed to practically reveal its vulnerabilities towards DoS attack. Note that possible countermeasures for reacting against other kinds of attack, such as data integrity violation, packet sniffing, password attack, have been just considered for NOSs in [30]; while, in this work, we want to focus on DoS effects only.

We remark that the main scope of a DoS attack is to deny the legitimate users/sources to access the desired resources. In the IoT context, this means that users are no longer able to exploit the services offered by the IoT platform. Such a task is normally caused by flooding the network with useless traffic so that legitimate packets will not be received by the legitimate recipients or, in another case, by depleting the network resources until the system is no longer able to provide useful information.

As regard NOSs, the following situations might happen:

- **DoS attack to the data sources**: one or more data sources might be compromised by a malicious entity that tries to overflow the bandwidth or their memory buffer so that they are not able to send valid data to NOSs. In such a situation, the IoT platform itself is not involved, but part of the IoT network may have to be isolated due to possible network's resources depletion caused by such compromised nodes. Moreover, some services will be affected, since NOSs will no longer receive data from that part of the network. Such an attack could be carried out by means of the common strategies, such as traffic injection, jamming, routing spoofing, and so on.
- **DoS attack to NOSs**: one or more NOSs might be directly compromised by external attackers aimed at exhausting their resources in terms of memory occupancy (i.e., the *Raw Data* storage unit may overflow), number of concurrent connections, and computational load. Besides NOSs are not conceived as constrained devices, their resources are even precious in order to guarantee a promptly service provision to the users. For these reasons, if one or more NOSs receive from a malicious entity (also masqueraded as a non-registered source) a huge amount of useless data, thus its computational and memory resources will be employed for useless processing tasks, preventing valid information to be provided in a reasonable time to the interested users. Such a situation could easily become true since NOSs deal both with registered and non-registered sources, which are both legitimated to connect to NOSs without the need of performing authentication.

Obviously, the worst case happens if the IoT platform itself (i.e., one or more NOSs) is brought down. As a consequence, the MQTT broker will not receive valid data, since NOS is busy in processing and analyzing of malicious information.

Note that the IoT system's administrators cannot restore the functionalities of the compromised data sources, because they are not under their control. However, the IoT system's administrators can preserve NOSs functionalities from violation attempts. Hence, all the actions of prevention, recognition, and recovery in case of DoS attack should start from NOSs' side. The main goal of the solution presented in the next section by *REATO* is to provide an efficient mechanism able, at the same time, to avoid, identify, and block a DoS attack in short interval times.

IV. DoS Attack Recognition and Recovery

As emerged in the previous section, NOSs become the principal actors for preserving the whole IoT system from a possible DoS attack. In fact, often, data sources are unable to directly counteract such a kind of threat, mainly due to their limited computational resources. Ad hoc solutions have been studied for particular set of sources, such as WSNs and MANETs, as presented in Section II. Now we want to propose a solution, named *REATO*, aimed to protect the IoT platform and its resources.

First of all, a scenario composed by multiple NOSs is considered hereby. They are supposed to be deployed in a wide area; therefore, data sources usually connect to the closer NOS. In order to allow the data provision by devices (i.e., sources), each NOS_i exposes a public port $port_{NOS_i}$ for connection requests.

When a source src_j wants to send data to NOS_i, it has to made a connection request req_k via HTTP (as explained in Section III) to the public port $port_{NOS_i}$. If the response sent by NOS is positive, then the source can start to provide its information, also in an encrypted way in case of registered sources. However, at the present state, the system is very vulnerable to DoS attack, because a malicious entity (that may be also represented by a non-registered source) might start to send a huge amount of invalid connection requests; then, if the NOS accepts to establish the connection, the malicious entity can also begin to send useless packets. Moreover, this may happen also if the req_k is encrypted, because a malicious request can be sent in any form. Note that such packets will be probably discarded, after processing, by NOS, due to their low quality and security scores, assigned by the algorithm used for data assessment presented in [30]. In fact, it is able to evaluate the behavior of both registered and non-registered data sources over the time and, thus, assessing the trustworthiness of the incoming information and recognize unreliable sources. If a source is considered unreliable, then its data will be automatically discarded. Nevertheless, NOS cannot prevent or block the waste of resources caused by the DoS attack.

In order to cope with such an issue, some countermeasures have to be undertaken. The first one regards the creation of a variable number of dynamic virtual ports $vport_{NOS_{i,m}}$ on each NOS (where m is the $m-th$ virtual port created on NOS_i). More in detail, when a source src_j requests for a connection req_k, then the NOS can reply with an acceptance message containing another encrypted port's address, which must be used by src_j for data provision. In this way, the set of active connections of the sources is not managed on a single port, known by all, but are switched on different virtual ports. Source src_j must perform another connection request to the virtual port $vport_{NOS_{i,m}}$ before starting normal interactions; in this way, NOS_i can inform the virtual ports about the authorized sources. The number M of created virtual ports depends on the number of connected data sources and on the bandwidth, and it should be established by means of simulations and/or real experiments (some examples are provided in Section V). Note that, more than one data source could be associated with the same virtual port. Also in this case, the number of threads generated for each data source per virtual port should be estimated by testing/real case studies (as reported in Section V). It is worth to remark that the physical and virtual memory of each NOS is shared by all the active threads (i.e., depending on the connected sources) on that NOS.

At this point, the second countermeasure consists in binding the identifier $vport_{NOS_{i,m}}$ of each NOSs' port to a UID, so as to obtain the following address: $UID - vport_{NOS_{i,m}}$, which is more complex to guess by malicious entities. It is composed by the name of the virtual port $vport_{NOS_{i,m}}$ and a UID, so as to obtain the following address: $UID - vport_{NOS_{i,m}}$. The UID is a unique identifier randomly generated and assigned to each NOS by a new IoT network component, named *Overload Balance Manager (OBM)*. Such a component is conceived as a remote server (which can run on a cloud), connected to all the NOSs belonging to the IoT system, by means of a secure VPN (Virtual Private Network). In this way, we assure that the communications among the OBM and the NOSs are private and not compromised by external entities. In the following, the functionalities of the OBM will be detailed.

It is important to note that, in case of registered data sources, $UID - vport_{NOS_{i,m}}$ is not sent in clear to the requesting source, but it is sent in a ciphered way, in order to provide a further security level to the communication. Such a behavior cannot be carried out for non-registered sources. Therefore, with the final aim of protecting as much as possible NOSs from attackers, some virtual ports are reserved for communications with registered and non-registered sources, respectively. In other words, the same virtual port will not be shared by both registered and non-registered sources.

Figure 2 sketches the analyzed scenario, showing the relations among OBM, NOSs, and the virtual ports, which are further outlined in Figure 3.

Several problems arise in the presented scenario, in case of DoS attack:

- The source src_j makes too many connection requests to NOS_i on the public port $port_{NOS_i}$;
- The source src_j sends too many packets to NOS_i on the address $UID - vport_{NOS_{i,m}}$.

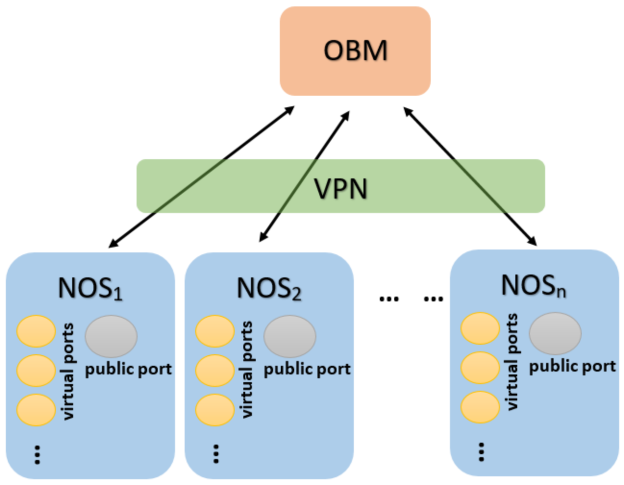

Fig. 2: General scenario

In the former case, being the name of the port for connection requests public, every malicious device could start an attack; while, in the latter case, the malicious entity should know the address $UID - vport_{NOS_{i,m}}$. Even if the use of virtual ports reduces the risk of sources' impersonation, such an address could be derived by performing an analysis of the actual communications among the NOS and the current connected data sources. To this end, a comparison among the transmitted packets could be carried out for detecting the virtual port addresses. It is important to note that, by means of such a method, the UID of the NOS and the name $vport_{NOS_{i,m}}$ of the virtual port could be recognized together or separately, as explained in the following cases.

Each NOS can react to such situations, in order to block the DoS attack in short times, in the following ways:

- **Case 1 - too many connection requests**: until a certain threshold thr_{conn} (to be dynamically calculated on the basis of a proper algorithm, as specified in the following) is exceeded, NOS can deny other further connection requests through the public port $port_{NOS_i}$ by the same requestor src_j, by sending back an exception, named ex_{conn}, and dropping future requests sent by src_j; in this way, no further session is opened on the virtual ports; the situation is shown in Figure 4;
- **Case 2 - too many packets sent**: until a certain threshold thr_{pck} (as for thr_{conn}, to be dynamically calculated by means of a proper method, as specified in the following) is exceeded, NOS can deny other further packets sent towards the address $UID - vport_{NOS_{i,m}}$ by the same requestor src_j, by sending back an exception,

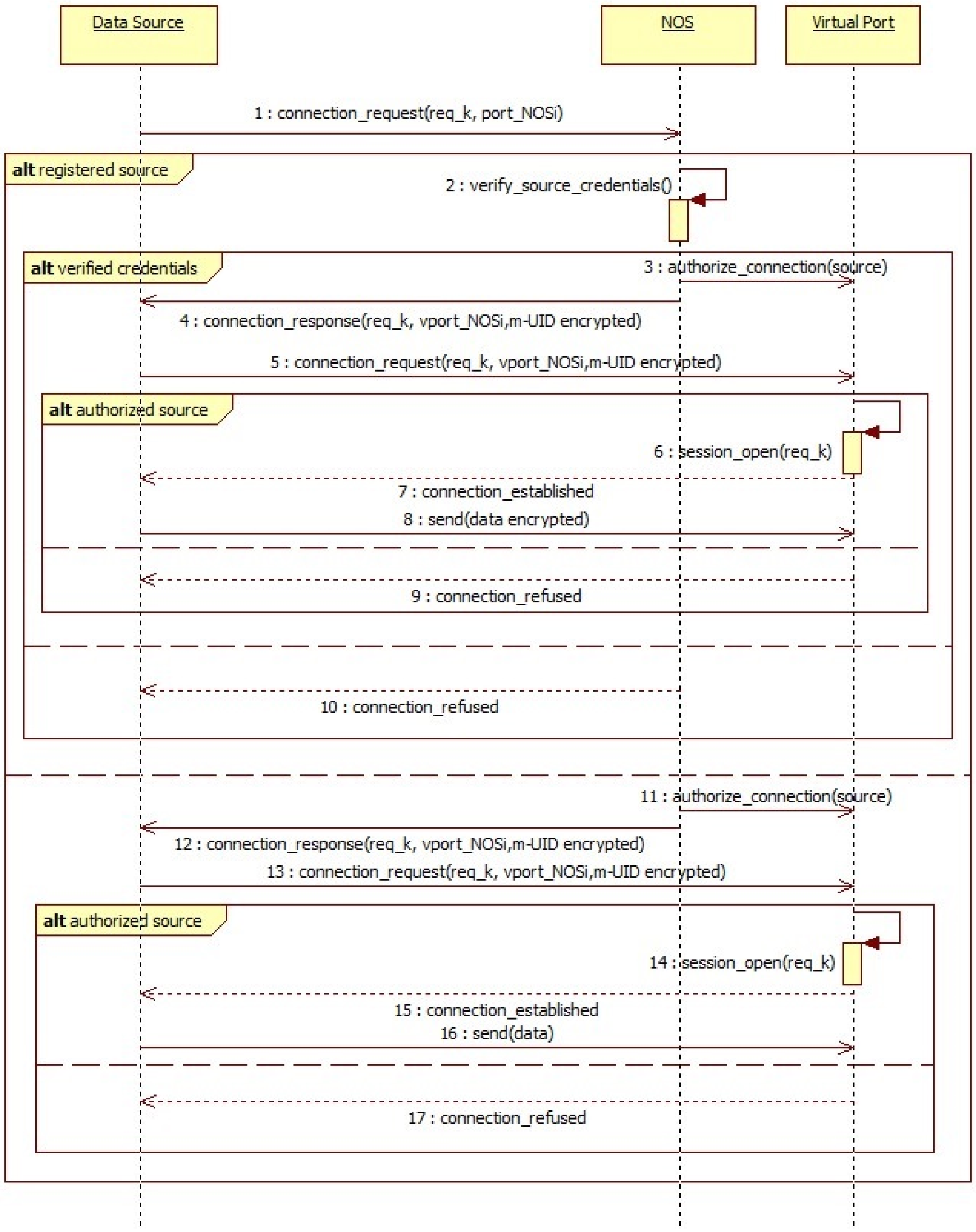
Data Source
NOS
Virtual Port
1 : connection_request(req_k, port_NOSi)
alt registered source
2 : verify_source_credentials()
alt verified credentials
3 : authorize_connection(source)
4 : connection_response(req_k, vport_NOSi,m-UID encrypted)
5 : connection_request(req_k, vport_NOSi,m-UID encrypted)
alt authorized source
6 : session_open(req_k)
7 : connection_established
8 : send(data encrypted)
9 : connection_refused
10 : connection_refused
11 : authorize_connection(source)
12 : connection_response(req_k, vport_NOSi,m-UID encrypted)
13 : connection_request(req_k, vport_NOSi,m-UID encrypted)
alt authorized source
14 : session_open(req_k)
15 : connection_established
16 : send(data)
17 : connection_refused

Fig. 3: Basic interactions

named ex_{pck}, and dropping future packets sent by src_j; all the opened sessions on such a virtual port are closed and also further connection requests, performed by the same source, via the public port $port_{NOS_i}$ will be prevented; in this way, NOS avoids to process and analyze information that are supposed to be useless. The main risk is to block a legitimate source if the value of the threshold thr_{pck} is set to a wrong value; in this case, the outcomes of the data assessment algorithm [30] can be used for performing a second level of evaluation about the behavior of a particular source and determining whether ban or not it. Such interactions are depicted in Figure 5;

- **Case 3 - malicious entity knows the whole address UID $-$ vport$_{\mathbf{NOS_{i,m}}}$**: if, besides its packets are dropped and no further session is allowed by NOS, the source src_j continues to overload the network with useless traffic on the known address $UID - vport_{NOS_{i,m}}$, NOS may decide to kill all the active connections on that port and rename it by means of a new port number; more

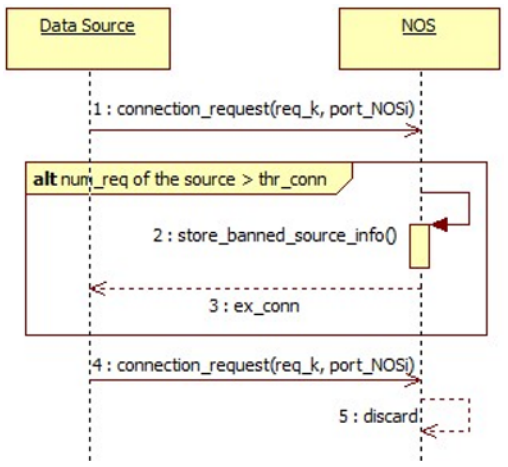
Data Source
NOS
1 : connection_request(req_k, port_NOSi)
alt num_req of the source > thr_conn
2 : store_banned_source_info()
3 : ex_conn
4 : connection_request(req_k, port_NOSi)
5 : discard

Fig. 4: Case 1: too many connection requests

in detail, an exception, named $ex_{req-conn}$, is sent to all the data sources that present an opened session on the attacked virtual port. In this way, they are forced to request a new connection via the public port $port_{NOS_i}$ (which is always the same), through which they will receive a new virtual port address $UID{-}vport_{NOS_{j,m}}$, possibly in a ciphered way. Therefore, NOS abruptly kills the old connections and delete the attacked virtual port so as the malicious entities cannot continue to perform the attack. Obviously, the re-creation of the right connections wastes time and resources and may slow down the provision of certain services; however, if the NOS is under an attack, the depletion of resources is still in progress, therefore such a countermeasure is acceptable. Note that NOS stores the information related to the source identified as malicious and, therefore, prevents future possible interactions. Figure 6 represents the interactions just described;

- **Case 4 - malicious entity knows UID**: if in the previous case (i.e., case 3) the virtual ports can be somewhat easily replaced without significantly affecting the NOS system functionalities, the situation is more complicated if the attacker has recognized the UID of the NOS. In fact, it can proceed to fill NOS with an unpredictable amount of malicious traffic because it is able, by knowing the UID, to derive the address of more than one active virtual ports. Remember that the address $UID - vport_{NOS_{i,m}}$ is obtained concatenating the identifier UID with the name $vport_{NOS_{i,m}}$ of the virtual port, which generally is a sequential number. The discovering of UID may be pursued by performing a brute force attack in order to reveal the name of one or more active virtual ports. NOSs recognize such a kind of event because more than one virtual port is attacked at the same time by one or more malicious entities (this behavior is revealed by the threshold thr_{pck}). In such a situation, the only solution for the victim of the attack (i.e., the NOS) is to move to another network location. For these reasons, the NOS contacts the OBM in order to start up a new instance NOS'_i of itself into another network location, also by replacing the UID with a new one. All the connected sources src_j are disconnected from NOS_i by sending an exception, named $ex_{nos-conn}$ and are invited to connect to the new instance. It is worth to remark that NOS_i is aware of who is the attacker and, therefore, its transactions are not re-directed to NOS'_i. Since NOS_i is under a DoS attack, it is not sure that all the sources will be notified of this event; anyway they should re-connect to another NOS or to the new instance NOS'_i of the suspended NOS. Figure 7 summarizes this case.
- **Case 5 - compromised sources continue to consume the network's resources**: besides the countermeasures just described, it is worth to remark that compromised sources can still waste network resources (i.e., bandwidth, CPU, memory), for example by generating fake packets/requests not directly sent to NOS, but transmitted over the network. As a consequence, the activity of such a part of the network is still compromised and further actions must be undertaken to block the attack. Extending the case 4 discussed above, when NOS recognizes that its functionalities are still at risk, then it notifies the OBM, which instantiates a new NOS instance, as before, but situated on physically separated networks (e.g., on different network cards or hubs) with respect to the previous one. In this way, the compromised network (e.g., a WiFi channel) is definitely isolated from the NOS system: only connections with legitimate sources are properly preserved; while compromised nodes are cut off. Note that it is up to the platform operator to set up the appropriate configuration settings in order to exploit such a mechanism and let the OBM to choose the better solution depending on the current network's state. Another important remark regards the possibility to add another module to NOSs, which is able to interact with the existing hardware and perform further blocking actions at the lower levels of the network, in response to certain kinds of attack. However, such a module would enforce a dependency on the underlying hardware platform, and it is out of the scope of the paper.

The situations just detailed revealed the possible vulnerabilities of the NOSs' system and how a DoS attack can be carried out and blocked, thus preserving the IoT system to be shut down. Note that more than one NOS may be attacked at the same time, but a chain of attacks is not possible, because no interaction is performed among different NOSs. This is due to the

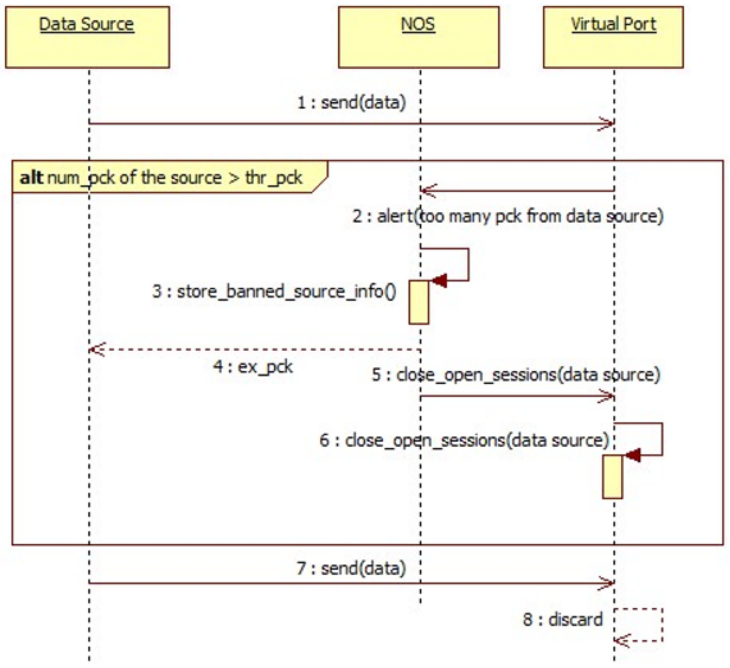
Data Source
NOS
Virtual Port
1 : send(data)
alt num_pck of the source > thr_pck
2 : alert(too many pck from data source)
3 : store_banned_source_info()
4 : ex_pck
5 : close_open_sessions(data source)
6 : close_open_sessions(data source)
7 : send(data)
8 : discard

Fig. 5: Case 2: too many packets sent

introduction of the OBM, which is responsible for the monitoring of the resources used by each NOS and, therefore, to react in case it verifies or it is informed by a NOS to be in a critical state. To this end, a notification system has been set up, in order to allow OBM to be aware, possibly in real-time, of the state of NOSs. Such a system allows NOS to periodically send to the OBM the following state information:

- Number $nActiveConn_{NOS_i}$ of active connections;
- Number $nFreeConn_{NOS_i}$ of available connections;
- Number $nReq_{NOS_i,src_j}$ of requests/packets received from each connected source;
- Number $nBadReq_{NOS_i,src_j}$ of invalid requests/packets (e.g., unknown type of data, bad requests/packets) received from each connected source;
- Average response time to requests $respTime_{NOS_i}$;
- CPU usage $uCPU_{NOS_i}$;
- Memory usage $uStorage_{NOS_i}$.

Such parameters, exchanged through a proper secure interface between each NOS and the OBM, can be analyzed and continuously monitored by the OBM itself in order to undertake further actions in order to keep the system in a healthy state. They would also help systems engineers to evaluate the efficiency of the system and/or to perform proper tasks, when needed, to optimize the network structure or prevent possible threats at the lower layers of the protocol stack.

In this paper, $nActiveConn_{NOS_i}$, $nReq_{NOS_i,src_j}$ and $nBadReq_{NOS_i,src_j}$ are used by NOSs to determine the above-mentioned parameters: thr_{pck} and thr_{conn}. Due to the importance, for a DoS detection system, to adapt to changes in network's condition, we cater for a mechanism able to dynamically changethr_{pck} and thr_{conn} at runtime. In particular, we used the algorithms and methods described in [34]. It is worth to remark that other valid methods are available in literature and it would be interesting, in the next future, to assess the applicability and performance of other approaches to the NOS case. More in detail, the work in [34] proposes a simple statistical-based analysis to monitor the traffic across the network's ports. Such a method is particularly effective in our case because it considers the comparison among the number of connections/packets/requests coming from the different ports of each NOS; such a comparison is useful to determine how large is the deviation of a certain port status from the known patterns of other ports. Traffic is continuously captured step by step from very short time periods (e.g., seconds) to relatively longer ones (e.g., hours). In this way, connections/packets/requests (i.e., $nActiveConn_{NOS_i}$, $nReq_{NOS_i,src_j}$ and $nBadReq_{NOS_i,src_j}$) fluctuations are analysed over time. Then, the algorithm calculates the values for thr_{pck} and thr_{conn} on the basis of the average of number of connections/packets/requests at a certain time scale. Such averages are conceived as the baseline regions to identify if one or more sources, belonging to the network, falls outside this region. Note that many variants and optimization could be added to such an approach, which however certainly represents a valid starting point.

Furthermore, the role of the OBM could be extended

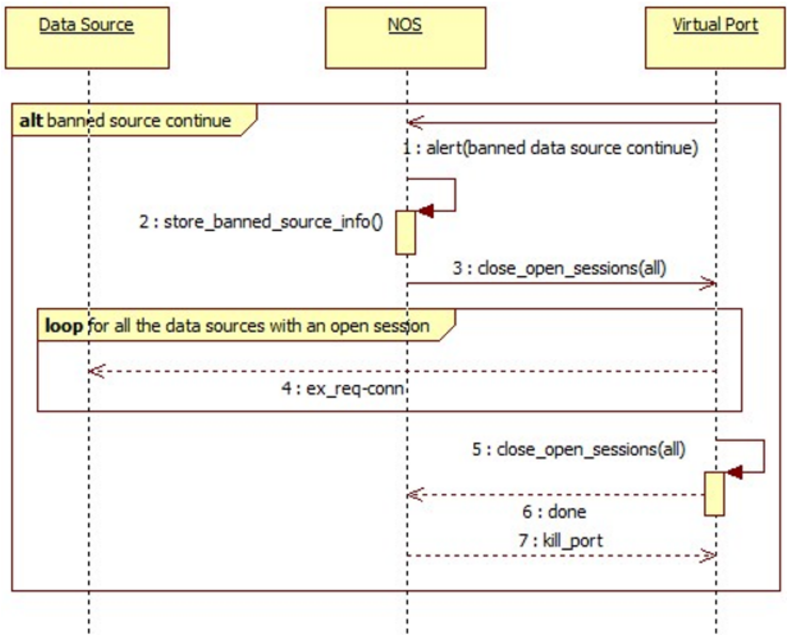
Data Source
NOS
Virtual Port
alt banned source continue
1 : alert(banned data source continue)
2 : store_banned_source_info()
3 : close_open_sessions(all)
loop for all the data sources with an open session
4 : ex_req-conn
5 : close_open_sessions(all)
6 : done
7 : kill_port

Fig. 6: Case 3 - continuous packets sent after banning

beyond *REATO*, in the next future, as to become a real balancer for better managing the load among the various NOSs, as may be suggested by the above-mentioned parameters. For example, part of the traffic on one NOS could be migrated to another one by communicating to the interested data sources the address $UID - vport_{NOS_{i,m}}$ of the virtual port that resides on the other NOS.

V. Validation and Experiments

In this section we want to demonstrate the feasibility of the proposed solution, as presented in Section IV. The test-bed is composed by a NOS that runs on Raspberry Pi platform, by an OBM, and by a variable number of data sources, which virtually run on a personal computer that receives data in real time from the meteorological station placed in the town of Campodenno (Trentino, Italy). Meteorological information can be accessed through the *Trentino Open Data* portal[5] and include: temperature, humidity, wind, energy consumption and air quality. WiFi connections are adopted for communications among the personal computer and the Raspberry Pi platform (i.e., the NOS). Another WiFi connection is used for the communications with the MQTT broker and with the OBM. Note that the presence of only one NOS does not compromise the relevance of the experiments because, as said in Section IV, no interaction is performed among different NOSs and so a chain of DoS attacks is not possible. Therefore, we safely conduct the analysis by means of one running NOS.

In the following examples, NOS fetches the data at 10 packet per second. Such a frequency, along with the number of connected data sources, obviously influences the memory occupancy as well as the computational effort. However, such a parameter is fixed in order to put in light the behavior of NOS in terms of CPU load, latency, and attack recovery time, with respect to other relevant variable parameters, such as the number of virtual ports and the outcomes in presence of a DoS attack or not, as described in the next sections. Instead, malicious sources send 25 requests/packets per second.

The two testing scenarios, respectively, have: (i) 10 registered data sources and 4 non-registered ones; (ii) 50 registered data sources and 20 non-registered ones. The number M of virtual ports varies depending on the number of connections/requests to be managed at a certain time. Such a configuration is set to 50 and Figure 8 shows the variation in the number of virtual ports during the system's execution. Note that connections/requests are equally split on the generated virtual ports, besides separating registered and non-registered sources. Furthermore, we suppose that the DoS attack is carried out by a variable number of data sources (i.e., from 1 to 15 malicious nodes in the two scenarios). Finally, simulations have been measured over a period of 24 hours. Table I summarizes the parameters used in the test-bed.

A. Computing effort

It is very interesting to analyze how the CPU load on the NOS changes in three different situations: (i) a normal situation without the presence of any attack; (ii) a case of DoS attack where NOS does not activate

[5]http://dati.trentino.it/dataset/raw-data-in-near-realtime-stazione-cmd001

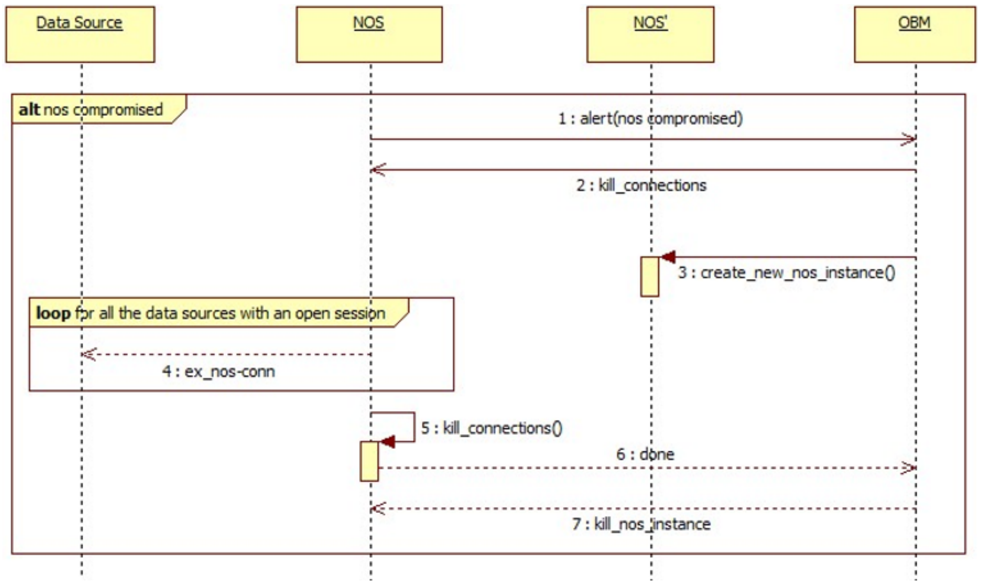

Fig. 7: Case 4 - continuous packets sent by knowing the UID of the NOS

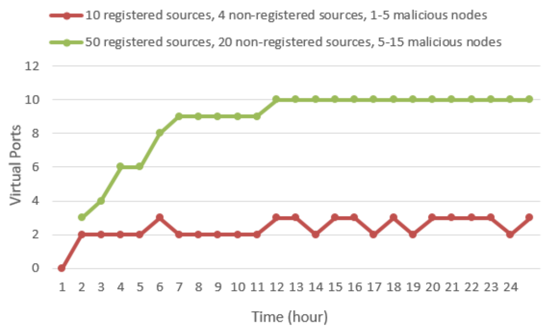

Fig. 8: Number of virtual ports during the time

TABLE I: Test-bed parameters

Parameter	**Value**
Registered sources	10, 50
Non-registered sources	4, 20
Malicious sources	from 3 to 15
Number of connections/requests to be managed by a single virtual port	50, 100
Data rate	10 pck/sec
Malicious sources' data rate	25 pck/sec
Observation time	24 h

REATO; (iii) a case of DoS attack where NOS activates *REATO*. Figure 9 shows the distribution of the CPU load on the analyzed NOS in the four cases presented in Section IV, with and without the activation of *REATO*, with a variable number of data sources and malicious nodes. As we expected, the CPU load increases from the first case to the last one, but without reaching worrying values when *REATO* is used. Therefore, NOS is supposed to behave in the correct manner in recognizing the recovering from the attack. It is worth to remark that computational effort is stable during system running and compliant with the results obtained in our previous work on NOS, such as [31] and [32]. When *REATO* is not used, performance is similar for the four cases because, practically, the malicious source acts in different manner but with the same outcomes (i.e., wasting NOS's resources); in general, the CPU load increases until reaching the 100% because a typical DoS attack causes the network to collapse. An important consideration must be highlighted with regards to the scalability of the presented system. In fact, we can see from the figure also how NOS platform reacts in presence of more sources and increasing malicious nodes. While additional studies, covering larger deployments, are needed, the results suggest that *REATO* can actually scale rather well: with 14 sources, CPU load is below 50%; increasing the number of sources to 50 sources, the CPU load is still below 65%.

An interesting aspect is related to the evolution of thr_{conn} and thr_{pck} when *REATO* is executed in cases one and two, respectively. The variation of such thresholds is documented in Figure 10.

B. Latency

Another fundamental metric to be considered is the latency introduced by the adoption of *REATO* with respect to the previous version of NOS [29] (i.e., with such a security extension disabled) in three settings: a normal situation, and in presence of an attack with and without *REATO*. For this evaluation, we considered the case three described in Section IV, with a variable number of sources and malicious nodes.

Note that latency is computed as the distribution of the elapsed time from the data reception to NOS until it is sent to the MQTT broker. Therefore, such a metric is useful for determining the efficiency of the service provision by NOS to the interested users.

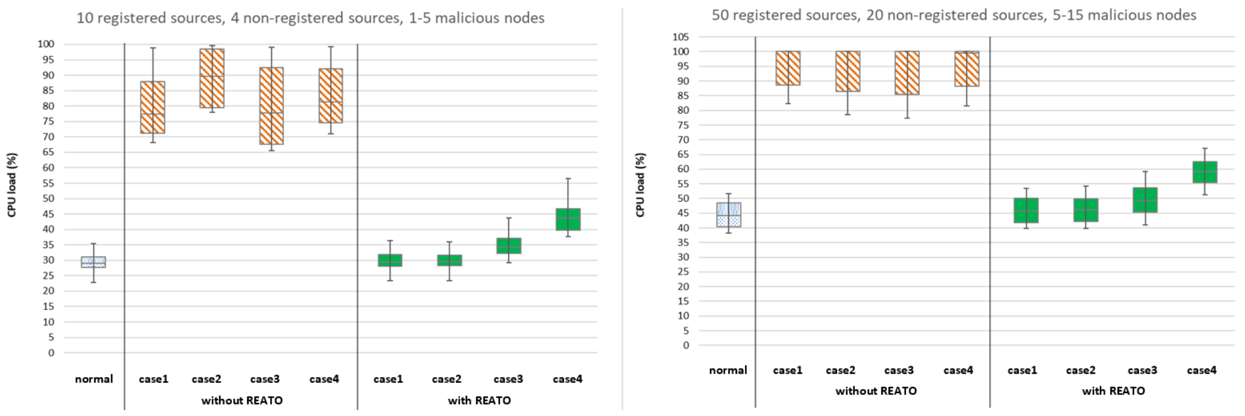

Fig. 9: CPU load: whiskers-box diagram for a NOS with and without *REATO*

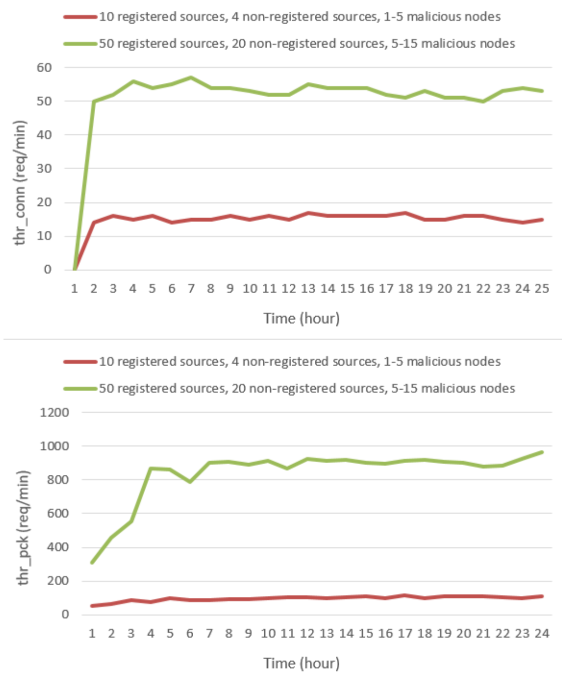

Fig. 10: Thresholds thr_{pck} and thr_{conn} during the system's execution with *REATO*

Figure 11 shows the distribution of the latency for the analyzed NOS, which, as the CPU load, results rather stable over the time. The analysis reveals that *REATO* greatly helps to maintain the latency times (in presence of DoS attack) near those of a normal situation (i.e., without the presence of DoS attack); while, in presence of a DoS attack, and without the activation of *REATO*, the performances are expected to get worse over the time, since no countermeasure is undertaken in order to block the attack. More in detail, by using *REATO*, with 14 sources (10 registered, 4 non-registered), the latency is below 15 ms; while, with 70 sources (50 registered, 20 non-registered), the latency is below 30 ms.

C. Attack recovery time

A last significant metric to be considered is the time required by the NOS with *REATO* to recover after a DoS attack. Such an evaluation is mainly relevant for cases three and four in Section IV. Figure 12 shows the distribution of the measured times, with different number of sources and malicious nodes. The difference in recovery times with respect to the number of sources is rather evident. It is worth to remark that, in case three, NOS has to kill the active connections on the attacked virtual port and re-create a new one; while, in case four, NOS has to be disconnected by the OBM and a new instance is enabled in another network location.

As said at the beginning of this section, many variable parameters have be evaluated to have a global view and an accurate estimation of the system behavior. However, the test-bed developed in this paper has demonstrated to be a valuable starting point in establishing a new solution in the defence from DoS attack in an IoT scenario.

VI. Conclusions

The paper has presented a method, named *REATO*, for detecting and counteracting a DoS attack against the IoT middleware, named NOS. The work started from the need to find a solution able to defend an IoT system towards DoS attacks, considering all the possible situations that can occur (i.e., attacks to the data sources and attacks to the IoT platform itself). The designed solution, tailored to NOS architecture, has been validated by means of a real test-bed, composed by a NOS prototype installed on a Raspberry Pi that receives open data feeds in real time by a variable set of sources. Performance indices like computing effort, latency, and recovery time have been evaluated in presence of malicious nodes. As a future development of the presented work, we aim at testing the described

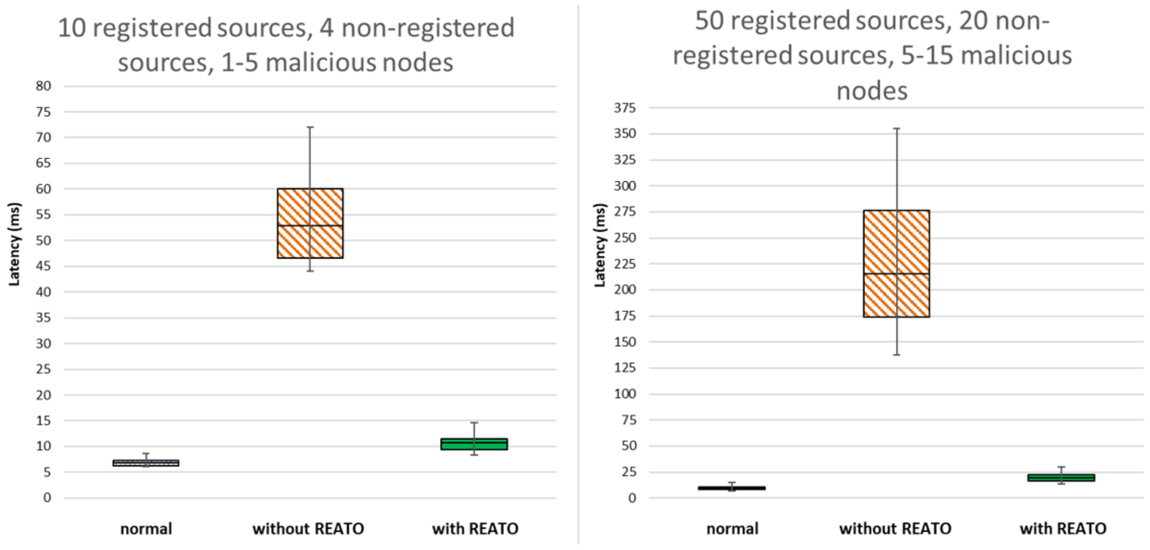

Fig. 11: Latency: whiskers-box diagram for a NOS with and without *REATO*

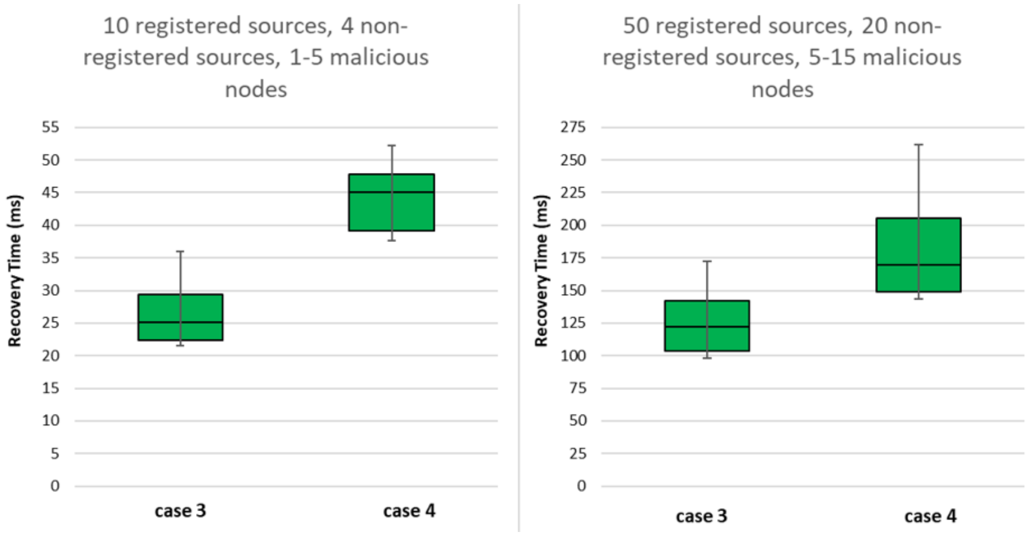

Fig. 12: Attack recovery time: whiskers-box diagram for a NOS with *REATO*

scenario in a more complex environment, composed of multiple NOSs and a huge number of data sources and malicious entities, in order to carry out further experiments about the performance of *REATO* into the whole system.

References

[1] 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.

[2] V. Zlomislić, K. Fertalj, and V. Sruk, "Denial of service attacks, defences and research challenges," *Cluster Computing*, pp. 1–11, 2017.

[3] S. Mavoungou, G. Kaddoum, M. Taha, and G. Matar, "Survey on threats and attacks on mobile networks," *IEEE Access*, vol. 4, pp. 4543–4572, 2016.

[4] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[5] S. Likmabam and R. Aaseri, "A review on detection and mitigation technique of distributed denial of services attack."

[6] E. Bertino and N. Islam, "Botnets and internet of things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.

[7] A. Patil and R. Gaikwad, "Comparative analysis of the prevention techniques of denial of service attacks in wireless sensor network," *Procedia Computer Science*, vol. 48, pp. 387–393, 2015.

[8] S. Patil and S. Chaudhari, “Dos attack prevention technique in wireless sensor networks,” *Procedia Computer Science*, vol. 79, pp. 715–721, 2016.

[9] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, “Gone: Dealing with node behavior,” in *Consumer Electronics-Berlin (ICCE-Berlin), 2015 IEEE 5th International Conference on*. IEEE, 2015, pp. 358–362.

[10] Y.-y. Zhang, X.-z. Li, and Y.-a. Liu, “The detection and defence of dos attack for wireless sensor network,” *The journal of china universities of posts and telecommunications*, vol. 19, pp. 52–56, 2012.

[11] H. Tan, D. Ostry, J. Zic, and S. Jha, “A confidential and dos-resistant multi-hop code dissemination protocol for wireless sensor networks,” *Computers & Security*, vol. 32, pp. 36–55, 2013.

[12] B. Li and L. Batten, “Using mobile agents to recover from node and database compromise in path-based dos attacks in wireless sensor networks,” *Journal of Network and Computer Applications*, vol. 32, no. 2, pp. 377–387, 2009.

[13] R. Nanda and P. V. Krishna, “Mitigating denial of service attacks in hierarchical wireless sensor networks,” *Network security*, vol. 2011, no. 10, pp. 14–18, 2011.

[14] S. Kumari, M. K. Khan, and M. Atiquzzaman, “User authentication schemes for wireless sensor networks: A review,” *Ad Hoc Networks*, vol. 27, pp. 159–194, 2015.

[15] J.-H. Son, H. Luo, and S.-W. Seo, “Denial of service attack-resistant flooding authentication in wireless sensor networks,” *Computer Communications*, vol. 33, no. 13, pp. 1531–1542, 2010.

[16] R. Akbani, T. Korkmaz, and G. Raju, “Heap: A packet authentication scheme for mobile ad hoc networks,” *Ad Hoc Networks*, vol. 6, no. 7, pp. 1134–1150, 2008.

[17] P. Devi and A. Kannammal, “An integrated intelligent paradigm to detect ddos attack in mobile ad hoc networks,” *International Journal of Embedded Systems*, vol. 8, no. 1, pp. 69–77, 2016.

[18] R. Upadhyay, U. R. Bhatt, and H. Tripathi, “Ddos attack aware dsr routing protocol in wsn,” *Procedia Computer Science*, vol. 78, pp. 68–74, 2016.

[19] A. Aris, S. F. Oktug, and S. B. O. Yalcin, “Internet-of-things security: Denial of service attacks,” in *2015 23nd Signal Processing and Communications Applications Conference (SIU)*, May 2015, pp. 903–906.

[20] S. Misra, P. V. Krishna, H. Agarwal, A. Saxena, and M. S. Obaidat, “A learning automata based solution for preventing distributed denial of service in internet of things,” in *2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, Oct 2011, pp. 114–122.

[21] K. Sonar and H. Upadhyay, *An Approach to Secure Internet of Things Against DDoS*. Springer Singapore, 2016, pp. 367–376.

[22] X. Ye and S. Singh, “A soa approach to counter ddos attacks,” in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 567–574.

[23] M. Ficco and M. Rak, “Intrusion tolerant approach for denial of service attacks to web services,” in *Data Compression, Communications and Processing (CCP), 2011 First International Conference on*. IEEE, 2011, pp. 285–292.

[24] S. Padmanabhuni, V. Singh, K. S. Kumar, and A. Chatterjee, “Preventing service oriented denial of service (presodos): A proposed approach,” in *Web Services, 2006. ICWS’06. International Conference on*. IEEE, 2006, pp. 577–584.

[25] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, “Denial-of-service detection in 6lowpan based internet of things,” in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*. IEEE, 2013, pp. 600–607.

[26] “EBBITS project,” http://www.ebbits-project.eu/.

[27] J. J. Costa Gondim, R. de Oliveira Albuquerque, A. Clayton Alves Nascimento, L. J. García Villalba, and T.-H. Kim, “A methodological approach for assessing amplified reflection distributed denial of service on the internet of things,” *Sensors*, vol. 16, no. 11, p. 1855, 2016.

[28] H. Yu, J. He, R. Liu, and D. Ji, “On the security of data collection and transmission from wireless sensor networks in the context of internet of things,” *International Journal of Distributed Sensor Networks*, vol. 9, no. 9, p. 806505, 2013.

[29] 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.

[30] 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.

[31] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini, “Aups: An open source authenticated publish/subscribe system for the Internet of Things,” *Information Systems*, vol. 62, pp. 29–41, 2016.

[32] 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.

[33] S. Sicari, A. Rizzardi, D. Miorandi, and A. Coen-Porisini, “Internet of Things: Security in the keys,” in *12th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, Malta, Nov 2016, pp. 129–133.

[34] A. Waskita, H. Suhartanto, P. Persadha, and L. T. Handoko, “A simple statistical analysis approach for intrusion detection system,” in *IEEE Conference on Systems, Process & Control (ICSPC)*. IEEE, 2013, pp. 193–197.