applied sciences

MDPI

Article

Enforcing Behavioral Profiles through Software-Defined Networks in the Industrial Internet of Things

Sara Nieves Matheu García [1,*], Alejandro Molina Zarca [1], José Luis Hernández-Ramos [2], Jorge Bernal Bernabé [1] and Antonio Skarmeta Gómez [1]

[1] Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain; alejandro.mzarca@um.es (A.M.Z.); jorgebernal@um.es (J.B.B.); skarmeta@um.es (A.S.G.)
[2] European Commission, Joint Research Centre, 21027 Ispra, Italy; jose-luis.hernandez-ramos@ec.europa.eu
* Correspondence: saranieves.matheu@um.es

Received: 7 October 2019; Accepted: 25 October 2019; Published: 28 October 2019

Abstract: The fourth industrial revolution is being mainly driven by the integration of Internet of Things (IoT) technologies to support the development lifecycle of systems and products. Despite the well-known advantages for the industry, an increasingly pervasive industrial ecosystem could make such devices an attractive target for potential attackers. Recently, the Manufacturer Usage Description (MUD) standard enables manufacturers to specify the intended use of their devices, thereby restricting the attack surface of a certain system. In this direction, we propose a mechanism to manage securely the obtaining and enforcement of MUD policies through the use of a Software-Defined Network (SDN) architecture. We analyze the applicability and advantages of the use of MUD in industrial environments based on our proposed solution, and provide an exhaustive performance evaluation of the required processes.

Keywords: internet of things; MUD; SDN; digital twins; security

1. Introduction

In recent years, the integration of Internet of Things (IoT) technologies in industrial environments has enabled the development of the well-known Industry 4.0 or Industrial Internet of Things (IIoT) [1]. This concept has been widely recognized as one of the main pillars for economic growth at the European level in order to improve the efficiency of industrial sectors such as automotive or telecom equipment [2] With the increasing digitalization of the industry, systems and components can be continuously and remotely accessed to be monitored during their development cycle and deployment. Like in other IoT-enabled scenarios, security concerns rise in an increasingly pervasive industrial environment. Indeed, security concerns could lead to huge economic losses for manufacturers, as well as a decrease in users' trust in the IIoT [3].

The recent and well-known attacks and botnets (e.g., Mirai [4]) underline the need for approaches to the detection, analysis, and mitigation of cybersecurity attacks in IoT devices. In most of the cases, the issue is not a single device itself, but the huge potential number of devices and systems that may be affected. An effective detection process is the first step to react to potential attacks efficiently. For this, it is crucial to identify the expected operation of a certain device in order to detect possible anomalies in its behavior. However, the definition of these behavioral profiles can be challenging due to the heterogeneity of IoT devices, in which a wide variety of technologies and protocols can be used with different configurations. Toward this end, the Manufacturer Usage Description (MUD) [5] has been recently proposed as an Internet Engineering Task Force (IETF) standard. standard for defining

the intended use of specific-purpose devices. The main objective is to limit the attack surface of a certain device by defining policies or Access Control Lists (ACLs) to restrict the communication with other devices or services. MUD has received a significant interest from the industry, and is considered by the National Institute of Standards and Technology (NIST). as a promising approach for the protection of IoT devices against denial of service (DoS) attacks [6]. In fact, the standard is intended for device manufacturers to define the behavior profile of their devices in order to facilitate a secure and automated deployment. The MUD approach defines an architecture and format for the specification of the ACLs, but it does not define specific mechanisms for obtaining and enforcing such policies in a trusted and secured way. We believe that the definition of these processes based on standard technologies is crucial to encourage the use and deployment of the MUD standard.

To address such needs, this work proposes the design and implementation of a security architecture for obtaining and enforcing MUD policies through the use of Software-Defined Networks (SDN) [7]. First, we define a lightweight mechanism for obtaining the MUD file through the integration and extension of the MUD architecture. For this, we propose the use of the Protocol for carrying Authentication for Network Access (PANA) [8] as a standard protocol for the transport of Extensible Authentication Protocol (EAP) [9] authentication messages. This way, the obtaining of the MUD file is carried out once the device has been correctly authenticated by the corresponding network components. Additionally, we base our approach on pre-shared key (PSK) authentication, in order to accommodate our approach into devices with tight resource constraints. Second, we propose an SDN architecture that is responsible for translating MUD policies into rules to be enforced by SDN switches. In particular, the proposed SDN approach is based on some of the components of the security management architecture proposed in the scope of the EU H2020 ANASTACIA project [10]. Toward this end, MUD policies are translated into an intermediate policy language called Medium-level Security Policy Language (MSPL), and converted into flow rules, which are enforced by using the OpenFlow protocol [11]. We extensively evaluated the resulting approach, and analyzed its applicability in an Industry 4.0 scenario based on the use of digital twins [12].

The remainder of the paper is as follows: Section 2 analyzes recent works related to different aspects of the MUD standard. Then, Section 3 describes the MUD approach, including the representation format for defining the ACLs. Section 4 describes our proposed MUD obtaining and enforcement architecture, which is further explained in Section 5. The application of our approach in different IIoT-related use cases is analyzed in Section 6. Then, the performance evaluation for the different phases of our solution is described in Section 7. Finally, Section 8 concludes the paper with an outlook about our future work in this area.

2. Related Work

The definition of behavioral profiles (or normal behavior) for a certain device or system is key for an effective detection of potential attacks in an IoT scenario. The main purpose of the MUD specification [5] is the definition of such profiles, in order to restrict the communications from/to a certain device to be defined by the manufacturer. The MUD standard is focused on the definition of the behavior of devices with a specific or single purpose (e.g., IoT devices), in contrast to general-purpose devices (e.g., a smartphone). The main reason is that IoT devices often communicate by following easily recognizable patterns [13]. MUD is an IETF standard, which has attracted an increasing attention across industry partners and standardization organizations. In particular, the NIST initially considered the use of MUD to mitigate IoT-based automated distributed threats [14], as well as to prevent vulnerable or insecure devices from being exploited [15]. Indeed, the use of MUD is also considered to mitigate network-based attacks in small business and home IoT devices [6]. Furthermore, the NIST is planning to create a National Thing Behavior Database (NTBD) (https://www.nist.gov/itl/applied-cybersecurity/nist-initiatives-iot) as a service similar to the National Vulnerability Database (NVD) (https://nvd.nist.gov/), using industry standard specifications such as the MUD.

The MUD standard provides an approach to foster a secure and automated deployment of IoT devices. This has also attracted the interest from the research community to address different aspects related to the use and generation of MUD profiles. In particular, the authors in [16] analyze the use of MUD to protect IoT devices against Distributed Denial of Service (DDoS) attacks. To do this, the authors implement a tool for generating MUD profiles using network traffic analysis over pcap files. With a similar approach, the authors in [17] proposes the development of a tool called MUDgee (https://github.com/ayyoob/mudgee), which also makes use of pcap files to generate the profiles. The tool is then applied to 28 specific IoT devices, whose profiles are publicly available (https://iotanalytics.unsw.edu.au/mud/). Additionally, the authors develop a formal semantic framework to validate MUD files, as well as the compatibility with policies that can be defined in the scope of a certain organization. Based on this work, the same authors present an approach to monitor such profiles in [18]. The main objective of these proposals is to support the generation of MUD profiles to encourage the deployment of the standard. However, these works do not address the process of obtaining and enforcing the MUD profiles. Indeed, the mentioned solutions are complementary to our approach in which we assume the existence of an MUD file, which must be obtained and enforced in the domain in which a certain IoT device is deployed.

For obtaining the MUD file, the standard assumes the use of an MUD URL, which the device itself needs to send to the corresponding network components in the domain where it is deployed. In particular, the MUD specification considers the use of three alternative approaches. In the first option, the MUD URL is sent through the Dynamic Host Configuration Protocol (DHCP) [19] in a DHCP Option. The second alternative considers the use of the Link Layer Discovery Protocol (LLDP) [20]. Then, the third option proposes to use the IEEE 802.1AR standard [21] to embed the MUD URL in a X.509 certificate. However, the standard does not define the whole process required for a legitimate device to communicate the corresponding MUD URL. Indeed, it should be noted that, in the case of the first two options, a device may be able to forge its identity and credentials, thus gaining additional network access [5]. To fill this gap, this paper proposes a novel mechanism for obtaining securely and in a trusted way the MUD file during a device's bootstrapping, by using strong authentication between manufacturer and the device. The proposed MUD obtaining is based on Extensible Authentication Protocol (EAP) to enable multi-domain deployments demanded by Industrial IoT. Namely, in our approach, we consider an alternative based on pre-shared key (PSK) authentication, which is realized through the use of standard technologies (such as EAP and PANA) during the bootstrapping process of the device, that is, when the device is deployed on a certain network [22]. The main motivation to consider PSK authentication is that some IoT devices may not be able to manage the use of public key cryptography operations due to resource constraints.

Moreover, the MUD specification does not define specific mechanisms for the enforcement of MUD profiles. To deal with this aspect, most of the recent approaches consider the use of SDN for the deployment of the rules contained in a certain MUD file. In particular, the authors in [23] propose an SDN-based architecture to translate MUD policies into flow rules to be enforced. These rules are proactively configured into network switches and used to detect attacks by using an Intrusion Detection System (IDS). The same authors use an SDN-based approach to monitor the compliance of a device's behavior with the corresponding MUD profile [24]. Toward this end, they develop an anomaly detection mechanism to identify potential attacks, such as DoS and Address Resolution Protocol (ARP) spoofing. An SDN-based framework is also considered by [25] for enforcing network access control and mitigating ARP spoofing attacks in the scope of smart homes. For evaluation purposes, they use the online tool *mudmaker* (https://www.mudmaker.org/). Furthermore, an implementation of an SDN-based enforcement approach of MUD profiles is provided in [26]. This implementation is supported by the NIST, and it is publicly available (https://github.com/usnistgov/nist-mud). However, these proposals do not provide details or evaluation results related to the translation of the MUD policies into flow rules to be enforced through SDN components. These aspects are summarized in Table 1.

Table 1. Comparison with existing approaches.

Reference	General Description	Gaps Addressed in Our Proposal
[16]	MUD files are generated from pcap files to protect IoT devices against DDoS attacks.	The process required to obtain the generated MUD file is not considered.
[17]	MUD files are generated from pcap files to create public profiles.	MUD files are generated after the device is installed in the network. The process required to obtain the generated MUD file is not considered.
[18]	Usage of the MUD to monitor suspicious behaviors.	The obtaining and enforcement phases of the MUD files are not addressed.
[23]	SDN-based architecture to translate MUD policies into flow rules to be enforced.	Details of the process to translate MUD files are not provided. Furthermore, the obtaining of the MUD files is not considered.
[24]	SDN-based approach to monitor the compliance of a device's behavior with the corresponding MUD profile.	The process required to obtain the generated MUD file is not considered.
[25]	SDN-based framework for enforcing network access control and mitigating ARP spoofing attacks validated with MUD files.	Details of the process to translate MUD files are not provided. Furthermore, the obtaining of the MUD files is not considered.
[26]	SDN-based enforcement approach of MUD profiles.	Details of the process to translate MUD files are not provided. Furthermore, the obtaining of the MUD files is not considered.

In contrast to previous approaches, our solution defines an architecture to manage the obtaining and enforcement of MUD profiles. In particular, we integrate both aspects into a bootstrapping mechanism, so that the network components in the deployment domain get the MUD files after the authentication of the IoT device. This way, only behavioral profiles of legitimate devices will be obtained and enforced. Our approach also represents a lightweight alternative to the use of certificates, which are proposed in the MUD standard. This aspect is crucial to ensure the deployment of the standard even in the presence of constrained devices or networks. Another advantage is the use of standard technologies for the process, which is transparent for the IoT device being deployed. Furthermore, this approach is integrated into the SDN-based architecture defined in the scope of the H2020 EU project ANASTACIA [10]. Such architecture follows a policy-based approach, so that MUD profiles are translated into an intermediate policy language before being translated into flow rules to be enforced by SDN switches. This intermediate translation provides a flexible approach, so potential changes in the MUD representation do not affect the rules enforcement. Furthermore, we evaluate our approach and analyze the potential application in the context of industrial scenarios. The details of our solution are further described in the following sections.

3. Specifying Behavioral Profiles in IoT

As already mentioned, MUD is a core component in our proposed solution. This section describes the main aspects of the standard, including its architecture and the representation format for the specification of network behavioral profiles.

3.1. Manufacturer Usage Description (MUD)

The main purpose of the MUD specification is to restrict the threat and attack surface of a device. While software updates will be still required during the device's lifecycle, it enhances the detection and mitigation of potential network attacks. Furthermore, MUD provides a scalable and flexible approach to the definition of policies beyond the use of IP addresses to enable communications with other services. In particular, a manufacturer could specify to allow the access to specific services in the cloud, but also the communication with devices of the same manufacturer. To provide a more fine-grained definition of access control rules, MUD also enables to specify specific protocols and ports for each communication. Moreover, it provides the possibility of extending the scheme, so that manufacturers can express other types of conditions or policies according to their needs. Indeed, although the current version of the standard is focused on network access control, potential extensions are envisioned, for example, for Quality of Service (QoS).

One of the main advantages of the MUD approach is that the definition of the devices' behavioral profiles is a responsibility for the manufacturer (instead of the typical network administrator). In fact, the MUD architecture and format allow for automating the definition of network access policies based on the MUD profile defined by the manufacturer. However, it should be noted that the instantiation of

these profiles could depend on the network domain where the device is being deployed. Such profiles are included in an MUD file, which defines the behavior associated with a certain device, using the Yet Another Next Generation (YANG) [27] and JavaScript Object Notation (JSON) [28] standard. To find a certain MUD file, the specification proposes the use of an MUD URL, which is sent by the device to the corresponding network components. The process to obtain this file and the components involved are explained in the following subsection.

3.2. MUD Architecture

The MUD architecture consists of four main components. Figure 1 shows the interactions among them for obtaining MUD files:

- Thing or Device, which is responsible for sending an MUD URL,
- Router or Switch to which the device is connected,
- MUD Manager, which is in charge (among other tasks) of requesting an MUD file based on a certain MUD URL,
- MUD File Server, a web server hosting MUD files.

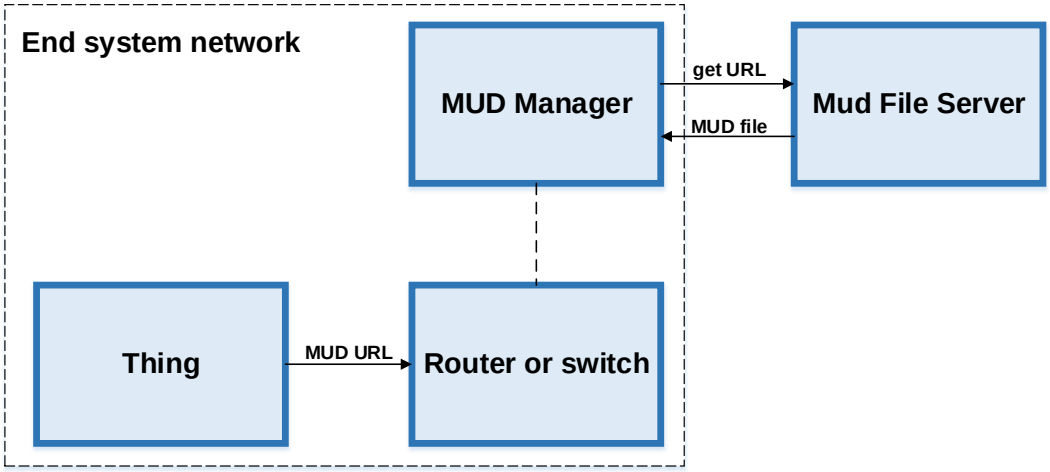


Figure 1. Manufacturer Usage Description architecture.

As defined in the MUD standard, the process for obtaining an MUD file requires the thing to communicate the location of the MUD file by using an MUD URL. Toward this end, the standard defines three alternatives: DHCP, LLDP or X.509 certificates. However, the standard also recognizes the possibility to consider other options for devices that are not able to communicate the MUD URL, or in scenarios with limited Internet connection. Then, the router or switch receives the MUD URL, which is forwarded to the MUD manager, which represents the core component of the architecture. Then, this entity requests the MUD file (and a signature file associated) to a certain MUD file server. After validating the corresponding signature, the MUD manager is also intended to configure the corresponding network components (e.g., a switch) based on the information contained in the MUD file. Indeed, it is also in charge of translating (and maintaining updated) MUD rules to specific network configurations [5].

As described in Section 2, DHCP and LLDP options set out security issues. Moreover, the use of X.509 certificates may represent an obstacle to consider MUD in scenarios with resource-constrained devices. Furthermore, the standard does not define the process required for the device to communicate the MUD URL once it has been successfully identified and authenticated in the domain where it is deployed. Moreover, according to the standard, the enforcement of the MUD rules remains a local deployment decision. In our solution, we address both aspects by defining a standard-based architecture to enable the obtaining and enforcement of MUD rules through SDN. Our architecture integrates and instantiates the MUD architecture's components, as will be described in Section 4.

3.3. MUD Model

MUD files define the type of communications and access of a certain device in the form of policies or access control lists (ACLs). Some examples of these restrictions could be "allow the communication to devices of the same manufacturer", "allow the access to a specific DNS service", or "deny the access for a specific port". As already mentioned, MUD is based on YANG [27] standard to model such restrictions, and JSON [28] as a serialization format.

A MUD file contains the "mud" and "acls" containers. The former defines different aspects related to the obtaining and validity period of a certain file. For example, the field "mud-url" identifies the MUD file itself, and the "last-update" specifies when the file was generated. In addition, the "to-device-policy" and "from-device-policy" containers represent access lists references by indicating the appropriate direction of a specific flow to define the communication pattern of the device. Consequently, the "acls" container defines those access control lists (ACLs). It should be noted that it is based on the YANG Data Model for Network Access Control Lists (ACLs) [27], which is augmented by the MUD standard to define more expressive ACLs. For example, the nodes "manufacturer" and "same-manufacturer" enable the definition of policies to allow or deny the interaction with devices from the same manufacturer. Other fields allow for referencing network components (e.g., "controller" or "local-networks") without the need to know the associated IP addresses.

The MUD model enables a flexible approach for defining network access control lists. However, it should be noted that this flexibility leads to the need for defining translation or interpretation approaches to convert MUD policies into rules, which can be enforced by network components. Our SDN-based architecture provides a translation approach, so that MUD restrictions can be automatically deployed into SDN switches to enforce the device's intended use.

4. Architecture

This section describes the architectural components, as well as the technologies and protocols of our solution. Furthermore, we identify different phases for the obtaining and enforcement of MUD files that will be further detailed in Section 5. To foster the deployment of the MUD standard, our architecture is based on the components already defined in Section 3.2. In particular, our solution extends the MUD architecture to accommodate the obtaining of the MUD file into the bootstrapping process of an IoT device. This process is defined in [22] as the phase of a device's lifecycle in which it is installed and commissioned within a network. This way, the MUD policies are obtained and deployed in the network even before the device gets access. This is a key aspect of our solution, since we adopt a proactive approach to avoid potential attacks (e.g., DoS attacks) to target devices. Furthermore, as will be described in Section 5, the process is transparent to the device itself; that is, network components are responsible for getting the MUD URL, MUD file, as well as to translate and enforce the rules contained in such file.

Our proposed architecture also represents an instantiation of the MUD processes by using standard technologies to encourage its use and deployment. In particular, we propose the use of the Extensible Authentication Protocol (EAP) [9], which is employed together an Authentication, Authorization and Accounting (AAA) Framework [29] for the bootstrapping phase. EAP is a flexible authentication framework, which allows different types of authentication mechanisms or EAP methods (e.g., pre-shared key (PSK) or Transport Layer Security (TLS) [30]). Indeed, the use of an EAP-AAA infrastructure is suggested in the MUD standard, but considering X.509 certificates. In contrast, we consider the EAP-PSK method [31], as a lightweight EAP method to be used even with resource-constrained devices, which cannot perform public key cryptographic operations. It should be noted that an EAP-AAA infrastructure has been widely considered for bootstrapping of IoT devices [32,33] for scalability reasons and support for multi-domain scenarios. In this case, an EAP session is established between an EAP peer (the device) and an EAP server through an EAP authenticator, which acts as a mere EAP packet forwarder.

For the transport of the EAP messages between the EAP peer and EAP authenticator (a.k.a. low layer protocol in EAP terminology [34]), we use the Protocol for Carrying Authentication for Network Access (PANA) [8]. Although there is a plethora of technologies and protocols proposed for IoT bootstrapping [35], PANA is an IETF protocol that is used by ZigBee IP [36] and the European Telecommunications Standards Institute (ETSI) Machine-to-Machine (M2M) [37]. Furthermore, as described in [38], PANA provides a more lightweight alternative than the Internet Key Exchange Protocol Version 2 (IKEv2) [39], which represents a standard alternative for the transport or EAP messages [34]. In the case of the communication between the EAP authenticator and the EAP server, we use the well-known Remote Authentication Dial In User Service (RADIUS) [40], in which the MUD URL is included after the device or thing is successfully authenticated, as will be described in Section 5.

As already mentioned, MUD files are enforced using an SDN-based approaches based on the architecture defined in the scope of the EU H2020 project ANASTACIA [10]. In particular, we accommodate some of the components of such architecture into the MUD manager, which represents the main architectural building block of the MUD standard. Furthermore, we extend their functionality to support the translation of MUD files into flow rules to be enforced by the corresponding network components (i.e., SDN switches). To this aim, we design and implement a component responsible for translating MUD abstractions (e.g., "same-manufacturer" or "my-controller") into specific security configurations. The translation process is supported by a Medium-level Security Policy Language (MSPL), which is intended to represent information about endpoints and communication protocols, but they are agnostic of the enforcement process. Then, these intermediate policies are translated into the corresponding flow rules. In this case, we use the well-known OpenFlow protocol [11], so that an SDN controller is enabled to install, modify, and remove SDN flows (among other operations) in such SDN switches.

Based on previous considerations, Figure 2 shows our proposed architecture to manage the obtaining and enforcement of the MUD file. We consider the existence of two different domains; the *deployment domain*, in which the device is being installed (i.e., bootstrapped [22]), and the *manufacturer domain*, in which the device was created, and, consequently, it maintains the associated MUD file. Furthermore, based on the selected technologies, each component plays different *roles* during the whole process. In particular, our architecture is composed by the following components:

- **Smart Object**: it represents a device intended to join the deployment domain. This entity represents the device itself (a.k.a. Thing in the MUD standard, as described in Section 3). This entity acts as an EAP peer to initiate the EAP exchange, and PANA Client (PaC) to transport the corresponding EAP-PSK interactions as part of PANA messages.
- **SDN Switch**: this entity acts as the entry point of the network for the smart object. Furthermore, it is responsible for enforcing the corresponding flow rules, which are derived from the rules contained in the MUD file.
- **Authentication Agent**: it is the intermediate entity in charge of forwarding the EAP-PSK messages between the smart object and the AAA Server. Therefore, it acts as an EAP Authenticator in terminology EAP. For the transport of EAP messages, it acts as a PANA Authentication Agent (PAA) receiving PANA messages from the PaC, and as a RADIUS Client for the communication with the AAA Server through the RADIUS protocol. Furthermore, when the device is authenticated, it gets the MUD URL from the AAA Server and forwards to the MUD Manager.
- **MUD Manager**: this is the core component of the architecture based on the functionality described in the MUD standard. Based on ANASTACIA architecture [10], this entity plays the following roles:
  - **SDN Orchestrator**: it is in charge of managing the translation of MUD rules into a specific security configuration to be enforced. After receiving the intermediate translation to MSPL,

it selects a specific *security enabler*, which represents a software component in charge of implementing the security function associated with a certain security policy. It also manages the enforcement of that security configuration.
  - **IoT Controller**: it makes the request to the corresponding MUD File Server based on the MUD URL, which is obtained from the Authentication Agent.
  - **Policy Interpreter**: it is responsible for the translation process itself by converting the MUD rules into intermediate MSPL policies. Then, it translates these policies into specific security configurations based on the security enabler selected by the SDN Orchestrator.
  - **SDN Controller**: it manages the enforcement of the security configurations that are sent by the SDN Orchestrator.
- **AAA Server**: it acts as a RADIUS Server for the exchange of EAP messages with the Authentication Agent, and as the EAP Server in EAP terminology. This entity is supposed to store the MUD URL associated with a smart object's MUD file.
- **MUD File Server**: following the MUD specification [5], it hosts the MUD files associated with the devices of a specific manufacturer.

Besides the identification of these components and roles, Figure 2 shows the main phases required for our approach to extend the bootstrapping of an IoT device. The initial phase is called *Smart Object Authentication* (in blue color), and it involves the required messages until the smart object and the Authentication Agent share the Master Session Key (MSK), which is sent by the AAA server after finishing the EAP-PSK method. Then, during the *MUD Obtaining* phase (red color), the Authentication Agent gets the MUD file (and the associated signature file) from the MUD File Server through the IoT Controller. This MUD file is translated into an MSPL policy in the *MUD Translation* phase (green color) by the Policy Interpreter, which receives the required information through the IoT Controller and SDN Controller, as part of the MUD Manager entity. Then, such intermediate policies are translated to specific security configurations, which are sent by the SDN Controller to the SDN Switch in the *MUD Enforcement* phase (yellow color). Based on these phases, the next section provides a detailed description of the message exchange required in our approach.

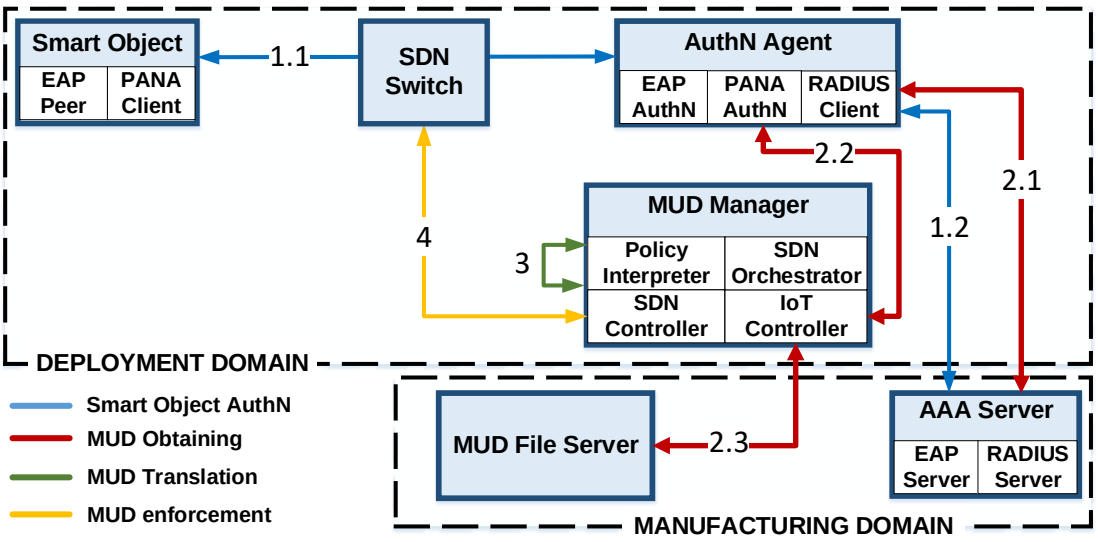


Figure 2. Architecture of the proposal.

5. MUD-Enhanced Bootstrapping

The explanation of the proposed solution is based on the phases previously identified. Figure 3 shows the detailed message exchange for the smart object authentication, as well as the obtaining, translation, and enforcement of the MUD file. It should be noted that, for deployment purposes, we consider separate entities for the different roles of the MUD Manager.

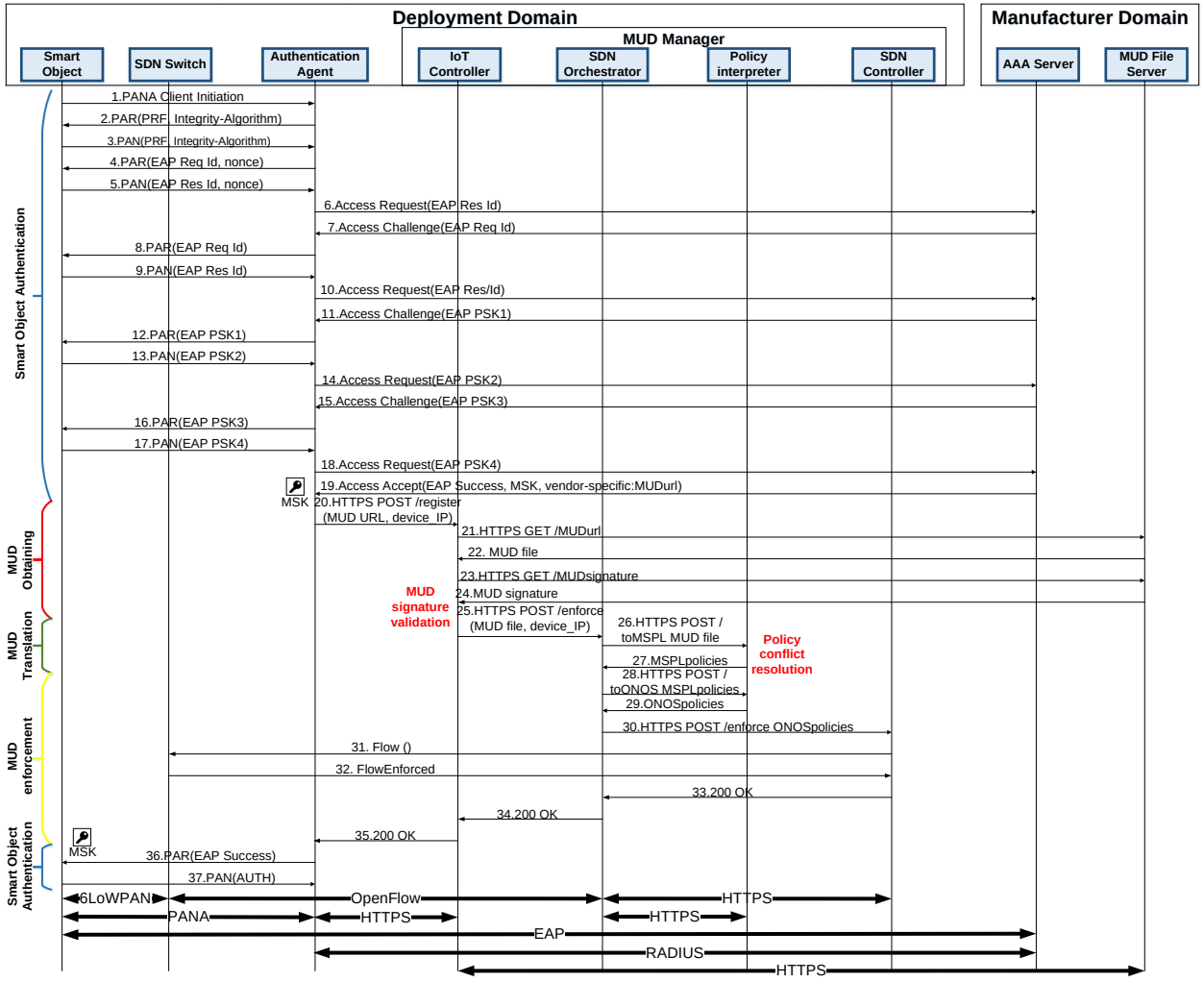


Figure 3. Flow diagram of the proposal.

5.1. Smart Object Authentication

The first phase of the MUD-enhanced bootstrapping is the *smart object authentication* based on the use of PANA to transport the required EAP-PSK messages. Therefore, this process is initiated by the smart object (acting as the PaC) with a PANA-Client-Initiation message (message 1) to the Authentication Agent, which acts as the PAA. Then, this entity sends a PANA-Auth-Request (PAR) (message 2) including the set of supported PRF and integrity algorithms, which are required for key derivation purposes. The smart object selects a PRF and integrity algorithm, and it includes them into a PANA-Auth-Answer (PAN) (message 3). After these parameters are agreed, the Authentication Agent requests the smart object identity, which is sent to the AAA Server. Toward this end, the smart object, Authentication Agent, and AAA server perform an EAP message exchange (messages 4–10) in which a specific EAP method is selected (EAP-PSK in our case). Then, the EAP-PSK exchange [31] is carried out for the authentication of the smart object. For this exchange (messages 11–18), PAR and PAN messages are used to transport EAP Request and EAP Response messages between the smart object and the Authentication Agent. Furthermore, this entity forwards EAP packets through a RADIUS Access-Request and RADIUS Access-Challenge messages to the AAA Server. If the process is successfully completed, the Authentication Agent receives an EAP Success packet, which is included in a RADIUS Access-Accept message (message 19). In our proposal, this message contains the MUD URL of the smart object's MUD file that is included as a RADIUS attribute *Vendor-Specific* [41]. Such attribute is intended to allow vendor to support their own extended attributes. This message also contains the Master Session Key (MSK), which is derived between the smart object (acting as EAP peer) and AAA server (acting as EAP server), and exported by the EAP-PSK exchange.

5.2. MUD Obtaining

Once the Authentication Agent has received the previous message including the MUD URL, it initiates the process to request the MUD file. That is, instead of forwarding the EAP success message to the smart object, it sends the MUD URL and IP address of such device to the IoT Controller. This component requests the MUD file to the MUD File Server through HTTPS (messages 21–22). After obtaining the MUD file, the IoT Controller extracts the *"mud-signature"* field from the MUD file that represents a Uniform Resource Identifier (URI) to get the associated signature (messages 23–24). These exchanges follow the MUD specification. However, it should be noted that the process is performed after the smart object is successfully authenticated. This way, only legitimate devices will be able to join a certain domain, in which they will be configured according to their MUD profile.

5.3. MUD Translation

Once the MUD file's signature has been validated, MUD rules can be enforced in different ways depending on the organization. However, before the enforcement of such rules, they should be translated to specific security configurations for a certain domain. Figure 4 shows the MUD file translation process at a high-level. The MUD file is provided to a domain-specific MUD translator, which is in charge of transforming the MUD rules in domain-specific security policies (i.e., MSPL policies). In order to perform the translation, the MUD translator needs to gather specific system domain information, such as the information regarding devices from the "same-manufacturer", or the concrete data of the field "my-controller". That is, it needs to translate the high-level MUD terms into a set of specific devices. After getting such information, the MUD translator transforms each rule in a set of security policies based on the translation configuration, and depending on the type of security policies implemented by the organization. In our proposal, the MUD translator was implemented as a module of the Policy Interpreter. Therefore, according to Figure 3, when the SDN Orchestrator receives an MUD enforcement request (message 25) with the MUD file and IP address of the smart object, it requests the MUD translation to the Policy Interpreter (message 26). Then, the MUD translator module parses the MUD file and performs the translation as follows: (i) it retrieves system model information

for the end-points and technologies involved in the MUD ACL; (ii) it identifies the main capability (i.e., an available feature of the system, such as filtering or channel protection) for the MUD ACL by taking into account a combination of the ACL information and the system model information. For instance, if the ACL aims to allow the communication between the smart object and a specific service (e.g., a Network Time Protocol (NTP) [42] service), a *forwarding* capability is detected. Furthermore, if the organization implements traffic encryption by default, then also channel protection (e.g., based on Datagram Transport Layer Security (DTLS) [43] could be detected; (iii) for each capability, the MUD Translator instantiates a capability-specific policy modelled MSPL. MSPL is used to represent security policies at medium-level abstraction, which represent information of the endpoints and communications, such as IPs and protocols, but they are interoperable and agnostic of the enforcement process. For instance, a filtering MSPL policy could be enforced in a traditional firewall or in an SDN network; (iv) the MUD translator repeats the process according to the number of MSPLs based on the devices and capabilities. It should be noted that, during this process, the new MSPL policies generated from the MUD file could represent conflicting restrictions with the security policies deployed in the deployment domain. Although it is out of the scope of this paper, a *policy conflict detection* procedure could be used to identify such issues. Finally, the result of the MUD translation process is notified to the SDN orchestrator (message 27).

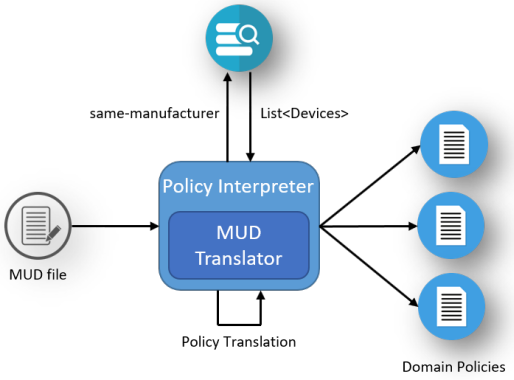


Figure 4. MUD translation.

5.4. MUD Enforcement

After the MUD translation has been performed, the SDN Orchestrator receives one or more MSPL policies, which must be translated into specific security configurations before they can be enforced. The SDN Orchestrator analyzes the different MSPL policies to identify the main required capabilities. Based on this, it decides the most appropriate security enabler in charge of enforcing each MSPL policy. In particular, a security enabler is defined as a software component in charge of implementing the security function associated with a MSPL policy, i.e., it allows enforcing the security policy in the managed system. For example, there exist different implementations to enforce the MSPL filtering policy. Our framework supports the enforcement of such policy with two enablers: through Netconf protocol [44], and using SDN through Openflow [11]. In order to make the decision, the Security Orchestrator takes into account the current status of the underlying infrastructure and technologies, as well as the available enabler plugins. These plugins are pieces of software that implement the translation from MSPL policies into specific security configurations for the security enablers. After selecting the appropriate security enabler for each MSPL policy (based on ONOS [45] in this case), the SDN Orchestrator requests the MSPL translation to the Policy Interpreter (message 28). Then, this entity retrieves the required plugins for the selected security enablers from a repository [10] to be executed for the translation. In the current scenario, since MUD policies are represented as MSPL forwarding policies, they are translated into ONOS API northbound configurations. When the translation process has finished, configurations are sent back to the SDN Orchestrator (message 29).

After that, the Security Orchestrator requests the policy enforcement for each enabler and configuration (message 30). As already mentioned, it enforces the SDN configuration through ONOS northbound API. The SDN Controller registers the configurations and performs the flows modifications by using OpenFlow *flow-mod* messages to the corresponding SDN switch (or switches). This communication is protected through TLS. The *match* field of the *flow-mod* message contains the "matches" packets, which is equivalent to the ACLs contained in the MUD file (e.g., IP of the same-manufacturer). Likewise, the *action* field in the *flow-mod* message contains the actions that should be taken for matching packets, as described by the MUD file, for instance, to specify a certain port to forward allowed traffic. Then, the SDN switch(es) apply the new flow rules in their corresponding flow tables (messages 31 and 32). After the enforcement of the rules, if the process has been successfully carried out, the SDN Controller notifies the Orchestrator (message 33) that confirms the process to the IoT Controller (message 34), which, in turn, notifies the Authentication Agent (message 35). Then, the MUD-enhanced bootstrapping process is finished with an EAP Success packet, which is transported in a PANA PAR (message 36) to the smart object. Finally, this entity acknowledges this message with a PANA PAN (message 37). This way, legitimate smart objects only will be able to access the domain after their intended use (reflected by the MUD file) is enforced by the corresponding network components.

As described, the proposed MUD-enhanced bootstrapping is intended to restrict the potential attacks to/from a certain smart object. Indeed, smart objects' communications will be only allowed once the MUD rules are enforced. After describing the required interactions, the next section describes the application of our approach in the scope of IIoT scenarios. Such use case is based on the use of *digital twins*, which represents an emerging concept to improve the processes required in industrial environments.

6. Enhancing Digital Twins with MUD Profiles

In the context of Industry 4.0, the concept of *digital twin* has become in a widely extended approach to improve industrial processes through a virtual representation of a system or product. Indeed, Gartner considers this concept as one of the strategic technologies for the coming years [46] with a massive deployment by organizations working on IoT. Although there is a plethora of definitions, digital twins can be described as *data-driven virtual representations that replicate, connect, and synchronize the operation of a manufacturing system or process. They utilize dynamically collected data to track system behaviors, analyze performance, and help make decisions without interrupting production* [47]. In this direction, we believe that the inclusion of MUD profiles as a component of a digital twin can enhance the virtual representation of a device or system being manufactured or monitored during its operation. This way, a *MUD-aware digital twin* can predict a potential security attack on a certain device based on the information coming from the device itself (or other components in the network). It enables testing potential countermeasures to such attack in the virtual device before deploying a certain solution on the real device.

Based on this, Figure 5 describes this scenario in which our proposed solution is integrated to foster a secure and automated deployment of IoT devices. Following with our approach described in Section 5, a smart object initiates the authentication process, in which PANA and RADIUS are used to transport the corresponding EAP-PSK messages (step 1). When it is authenticated, the Authentication Agent obtains the MUD URL associated with that device (step 2), and gets the MUD File from the MUD File Server (step 3). Listing 1 shows an MUD file example in which a device is only allowed to communicate with devices from the same manufacturer (*"same-manufacturer": [manufacturer1]*) by using the User Datagram Protocol (UDP) (*"protocol": 17*). This example is also employed for evaluation aspects, which are described in Section 7.

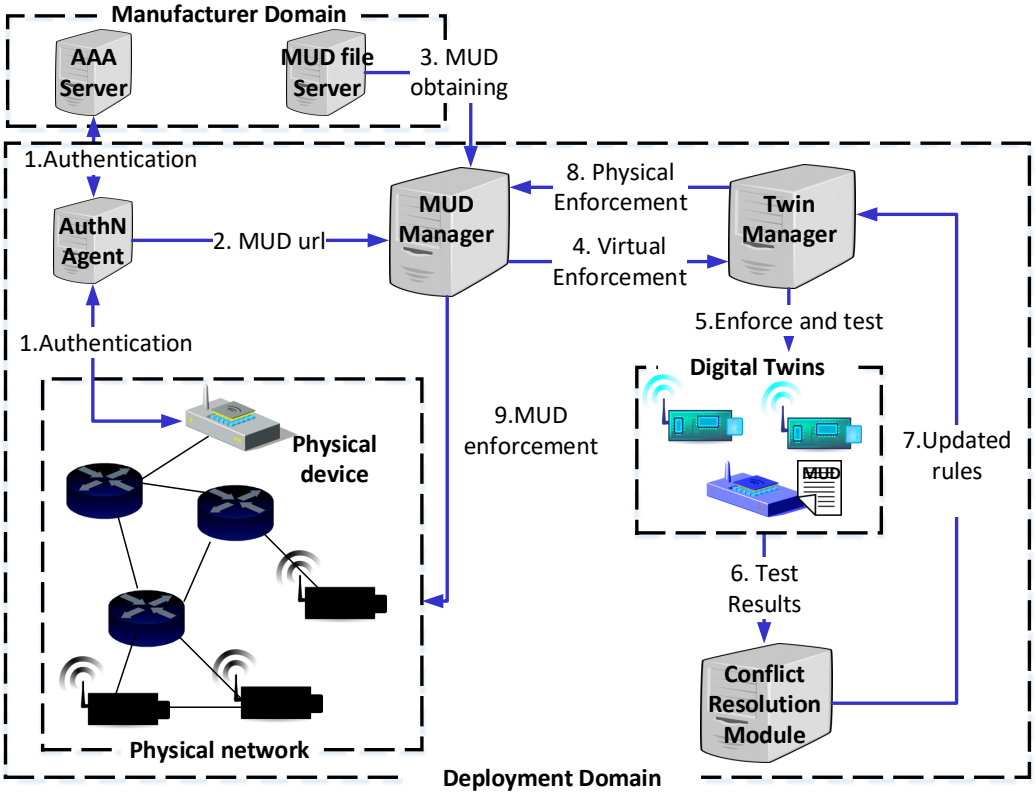


Figure 5. Integrating MUD-enhanced bootstrapping with digital twins.

Listing 1: Extract from the MUD file.

```
{
"ietf-mud:mud": {
"mud-version":1,
"mud-url":"https://manufacturer1/mote1",
"last-update":"2019-01-22T12:49:03"
...
"from-device-policy": {
"access-lists": {
"access-list": [
{"name": "ACL_fromDevice"}
]
}
},
...
}
"ietf-access-control-list:acls": {
"acl": [
...
"name": "ACL_fromDevice",
"type": "ipv6-acl-type",
"aces": {
"ace": [{
"name": "ACE_fromDevice_1",
"matches": {
"ietf-mud:mud": {
"same-manufacturer": [manufacturer1]
},
"ipv6": {"protocol": 17},
},
"actions": {"forwarding": "accept"}
}]
}
]
}
}
```

After obtaining the MUD file, the MUD Manager performs the MUD translation process. Listing 2 shows an example of translation from the Listing 1 MUD example to a Medium-level security policy language (MSPL). Specifically, it shows a policy for orchestration that contains several forwarding policies (*ITResource* elements) to allow the communication between devices from the *same-manufacturer*. In this example, the security policy indicates all UDP traffic from the IPv6 address *aaaa::1* (*mote1* device)

must be able to reach the destination *aaaa::2* that corresponds with other devices provided by the same manufacturer.

Listing 2: Extract from the MSPL file.

```
<ITResourceOrchestration id="omspl_X">
<ITResource id="mspl_X"
orchestrationID="omspl_X">
<configuration
xsi:type="RuleSetConfiguration">
<capability>
<Name>Traffic_Divert</Name>
</capability>
<configurationRule>
<configurationRuleAction
xsi:type="TrafficDivertAction">
<TrafficDivertActionType>FORWARD</>
<packetDivertAction>
<packetFilterCondition>
<DestinationAddress>aaaa::2/128</>
</packetFilterCondition>
</packetDivertAction>
</configurationRuleAction>
<configurationCondition
xsi:type="TrafficDivertConfCondition">
<packetFilterCondition>
<SourceAddress>aaaa::1/128</>
<Protocoltype>UDP</>
</packetFilterCondition>
</configurationCondition>
<externalData xsi:type="Priority">
<value>60000</value>
</externalData>
</configurationRule>
</configuration>
</ITResource>
...
</ITResourceOrchestration>
```

However, instead of enforcing the resulting rules over the real network (i.e., through SDN switches), the rules contained in the MUD file are tested in the digital twin associated with the smart object (step 4). Toward this end, the *Twin Manager* is in charge of monitoring a set of digital twins in a certain network or system. This component generates a set of tests to simulate the smart objects' intended behavior, which is represented by the rules associated with the MUD file (step 5). The main purpose of these tests (executed in the digital twin) is to assess the impact of the new device's behavior into the deployment domain. For example, it can generate ping requests to test that a certain service is reachable according to a specific MUD rule. These tests are also used to detect potential inconsistencies with the current network behavior, in such a way that the behavior of deployed devices is not affected by those rules. Then, the results of such tests are sent to the *Conflict Resolution* component (step 6), which is intended to assess the compliance of the MUD rules with the security policies associated with that domain. For example, the results of the tests could reveal that the smart object needs to communicate with a certain website on the Internet that is considered as a non-trusted service by the domain. In this case, this component could specify an alternative trusted service to enable the intended behavior of the smart object. Indeed, it should be noted that, according to the MUD specification [5], MUD rules are not directives, but suggestions. Therefore, they could be instantiated in a different way according to the security policies of the domain in which the device is deployed.

In case of detecting potential conflicts, an alternative network behavior is proposed for the smart object being bootstrapped (step 7). These updated rules are sent to the Twin Manager, which generates new tests according to such rules. The process of testing and conflict resolution is repeated until the Conflict Resolution component checks that the resulting policies are compliant with the policies associated with the domain. In this case, such policies can be enforced in the real system (step 8), according to the process described in Section 5.

7. Performance Evaluation

For the evaluation of our MUD-enhanced bootstrapping, we have implemented and deployed the components described in Section 5. It should be noted that we consider a different deployment component for each role of the MUD manager. Table 2 provides a summary with the hardware and software libraries for each component of our testbed. For the Smart Object, we used the Cooja Network

Simulator (https://anrg.usc.edu/contiki/index.php/Cooja_Simulator), which enables the simulation of constrained devices based on the Contiki operating system [48]. By using Cooja, a *border router* entity is required for the communication between the simulation environment and the external entities.

For the deployment of the Policy Interpreter and SDN Orchestrator entities, we use our own implementation in Python for the Representational State Transfer(REST) Application Programming Interfaces (APIs). The Policy Interpreter implements four different HTTP APIs to translate and enforce the two different security policy levels. In our implementation, we consider a policy repository to maintain the policies that makes use of two different APIs to store the policy translations as well as the policy enforcement status. To enforce the MUD filtering policies, we use ONOS Controller (ONOS: https://onosproject.org/), since it is widely used in productions systems. The Policy Interpreter implements the translator from MSPL into SDN low-lever configurations understandable by ONOS controller, namely set of ONOS northbound APIs calls.

Table 2. Testbed specifications.

Component	Hardware	Role	Software
Smart Object	Zolertia Z1 with 92 kB of nominal ROM and 8 kB of RAM	EAP peer PANA Client	Cooja (Contiki OS 2.7) OpenPana
Authentication Agent	Linux Ubuntu VM with 2 GB of RAM, 30 GB HDD and a processor Intel(R) Core(TM) i7-8550U at 1.9 GHz, using 1 core	EAP Authenticator	FreeRadius 2.0.2.
		PAA	OpenPana
AAA Server	Linux Ubuntu VM with 2 GB of RAM, 30 GB HDD and a processor Intel(R) Core(TM) i7-8550U at 1.9 GHz, using 1 core	AAA Server	FreeRadius 2.0.2.
		EAP Server	C application
MUD Manager	Intel Core Processor (Haswell) at 1.5 GHz using 2vCores, 2 GB of RAM and 15 GB of HDD Intel(R) Core(TM) i7-2600 CPU at 3.4 GHz, using 3 vCores, 3.5 GB of RAM and 30 GB of HDD	IoT Controller	Python application
		SDN Orchestrator	Django 2.2.2 and Falcon 2.0
		SDN Controller	ONOS
		Policy Interpreter	Django 2.2.2 and Falcon 2.0
MUD Server	Linux Ubuntu VM with 2 GB of RAM, 30 GB HDD and a processor Intel(R) Core(TM) i7-8550U at 1.9 GHz, using 1 core	MUD Server	Apache 2.4.39
Border router	Zolertia Z1 with 92 kB of nominal ROM and 8 kB of RAM	Border router	Contiki OS 2.7

7.1. Smart Object Authentication and MUD Obtaining

The delay required for each message during the smart object authentication phase (see Section 5 is shown in Table 3. Each value represents the time of receiving and processing a message, and the delay to generate the next one according to the process. As shown, the messages to be processed by the smart object (with constrained capabilities) require more delay, specially the EAP Success (message 36), in which the device receives the cryptographic material to generate the shared secret. According to these partial times, the total delay for the authentication process requires a mean time of 4454.164 milliseconds. This delay is acceptable taking into account the advantages provided by our approach in terms of flexibility, multi-domain, and standardization properties. It should be noted that our approach is an alternative to the use of certificates, as proposed in the MUD standard. In that case, the use public key cryptography could be unfeasible in devices with tight resource constraints. Furthermore, the use of certificates requires the transport of more heavy messages that could have a negative impact, especially in the case of Low Power and Lossy Networks (LLN).

Table 3. Device authentication performance by messages.

Message Number	Mean Time	Standard Deviation	Message Number	Mean Time	Standard Deviation	Message Number	Mean Time	Standard Deviation
1	0.297	0.094	8	462.842	±53.522	15	0.105	±0.028
2	642.532	±62.484	9	0.098	±0.020	16	588.944	± 67.125
3	0.104	±0.045	10	0.637	±0.083	17	0.094	±0.021
4	496.934	±92.464	11	0.156	±0.154	18	0.194	±0.036
5	0.093	±0.009	12	593.152	± 0.828	19	0.195	±0.154
6	0.165	±0.018	13	0.100	±0.020	36	1013.92	±97.167
7	0.140	±0.123	14	0.197	±0.053	37	631.034	±50.481

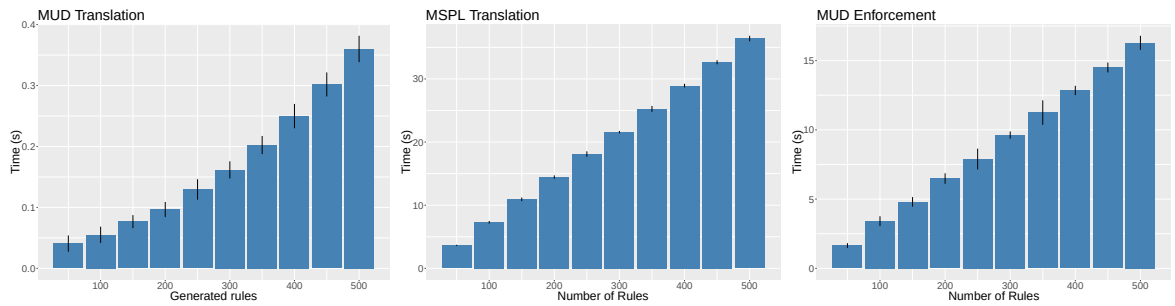
Regarding the MUD Obtaining phase, we distinguish three steps according to Figure 3: MUD file obtaining (messages 17–18), MUD signature obtaining (messages 19–20), and the validation of the MUD signature. Table 4 shows the time measurement for these steps. For these values, we used the MUD file shown in Listing 1 and an Elliptic Curve Digital Signature Algorithm (ECDSA). signature of 256 bits generated through OpenSSL 1.1.1 (https://www.openssl.org/).

Table 4. Manufacturer Usage Description phases performance.

Phase	Mean Time (ms)	Standard Deviation
Manufacturer Usage Description obtaining	37.133	6.791
Manufacturer Usage Description signature obtaining	34.833	3.905
Manufacturer Usage Description signature verification	7.733	1.964
Total	79.700	8.243

7.2. Translating and Enforcing MUD Rules

Regarding the MUD translation phase, the MUD model provides high-level terms like "same-manufacturer", which must be translated in specific deployed devices provided by the same manufacturer. We use these aspects to evaluate the scalability of our approach. Thus, Figure 6 shows the time and standard deviation (with 30 executions for each value) required for the MUD translation (messages 26–27 from Figure 3) for an MUD file that allows bi-directional connectivity among the devices provided by the same manufacturer. For evaluation purposes, the tests were divided in ten iterations where we considered a deployment of 50–500 IoT devices (with intervals of 50 devices). As shown, the MUD translation process follows a logarithmic progression. This is because, even if an MUD rule could be translated into different MSPL policies, only specific parameters (e.g., IP addresses) are modified.



(**a**) MUD translation (**b**) MSPL translation (**c**) MUD Enforcement

Figure 6. MUD translation and enforcement performance.

Furthermore, Figure 6 shows the means and standard deviation for the MSPL to final configuration translation process (messages 28–29 in Figure 3). In this case, the results also show a linear progression.

This second translation process requires considerably more time than the first one, since, unlike the first that is basically a data model translation, this second process involves the execution of the software plugin required to generate specific configurations using MSPL.

Finally, all the obtained configurations during the policy translation are enforced through the northbound API of the SDN controller (messages 30 to 35, from Figure 3). In this case, the result is also linear since the enforced configurations are quite similar but we can see a slight change in the deviations which is attributed to network latency. Since the MUD file employed (Listing 1 contains rules with the word *same-manufacturer*, and all the devices deployed are from the same manufacturer, each rule leads to the number of devices within the network multiplied by two; a rule limiting the communication from the device, and another one to that device. Although this expressivity is an advantage of the MUD standard, the efforts of generating all the rules, translating, and enforcing them are visible in our results.

Figure 7 shows an overview of the time consumption for each phase according to a different number of generated policies. As shown, for a small number of rules, the smart object authentication and MUD enforcement times represent most of the time consumption over the process (45.15% and 36.91%, respectively). However, for a large number of rules, the translation of the MSPL policies and the MUD enforcement become into the most consuming phases. Indeed, for 500 rules, these processes represent the 28.29% and 63.21% respectively. This coincides with the previous results, where the impact of translation and enforcement is highlighted. In this direction, part of our future work is focused on the optimization of MSPL policies translations, while the expressivity of the MUD standard can be still leveraged. This aspect could enhance the potential deployment of MUD in broad-scale scenarios.

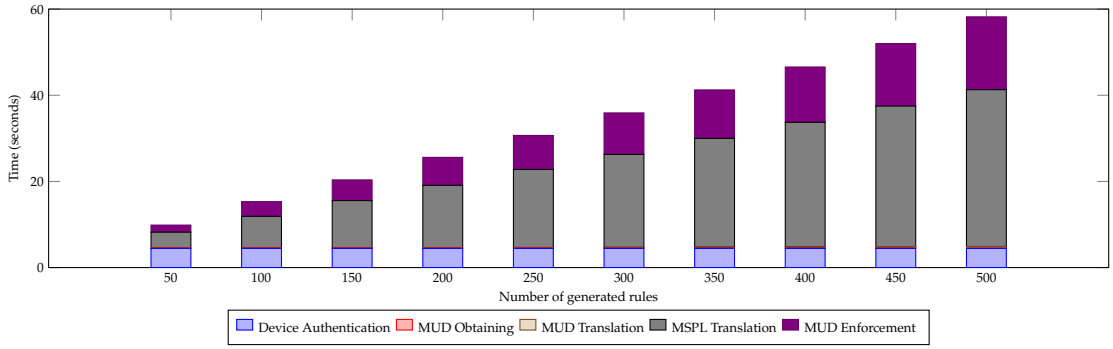


Figure 7. MUD complete process performance.

8. Conclusions

Industry 4.0 represents an emerging IoT-enabled ecosystem with a huge social and economic impact. In a more ubiquitous industrial scenario, security is crucial to cope with the effects of potential attacks, which could lead to a decreasing trust of end users. In this work, we have proposed a proactive security approach to manage a secured and automated deployment of IoT devices behavioral profiles by using an SDN-enabled security architecture and standard technologies. Our mechanism integrates and extends the recent MUD standard architecture, so that only legitimate devices are enabled to access a certain domain, in which security behavioral profiles are enforced to restrict the attack surface. We also analyzed the application of our approach in the context of industrial scenarios, and carried out an exhaustive evaluation. As a future work, we plan to extend MUD profiles with a more fine-grained approach by considering specific security aspects to drive devices' communications. These aspects will be enforced through an extended SDN-NFV architecture to dynamically deploy the required functionality according to such profiles.

Author Contributions: All of the authors conceived and designed the proposed architecture. S.N.M.G. and J.L.H.-R. provided input on the bootstrapping and MUD obtaining phases, whereas J.B.B. and A.M.Z. provided

input on the MUD translation and enforcement phases. S.N.M.G. and A.M.Z. carried out the performance evaluation. A.S.G. led the research line. All of the authors contributed to writing the paper.

Funding: This work has been partially funded in part by the Spanish Ministry of Economy and Competitiveness and the ERDF funds cofinantiation through the PERSEIDES project under GrantTIN2017-86885-R and in part by the European Commission through the CyberSec4Europe H2020-830929 and the H2020-780139 SerIoT projects. The research has been also supported by a postdoctoral INCIBE grant within the "Ayudas para la Excelencia de los Equipos de Investigación Avanzada en Ciberseguridad" Program, with code INCIBEI-2015-27363 and by the FPU-16/03305 research contract of the Ministry of Education and Professional Training of Spain.

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

Abbreviations

The following abbreviations are used in this manuscript:

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACL	Access Control List
API	Application Programming Interface
CoAP	Constrained Application Protocol
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial Of Service
ECDSA	Elliptic Curve Digital Signature Algorithm
ETSI	European Telecommunications Standards Institute
IETF	Internet Engineering Task Force
IPv6	Internet Protocol version 6
IoT	Internet of Things
JSON	JavaScript Object Notation
LLDP	Link Layer Discovery Protocol
MUD	Manufacturer Usage Description
NIST	National Institute of Standard and Technology
NTBD	National Thing Behavior Database
PSK	Pre-Shared Key
REST	Representational State Transfer
SDN	Software Defined Network
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
YANG	Yet Another Next Generation

References

1. Gilchrist, A. *Industry 4.0—The Industrial Internet of Things*, 1st ed.; Apress: New York, NY, USA, 2016.
2. European Commission. Digitising European Industry. In *Digital Single Market*; European Commission: Brussels, Belgium, 2015. Available online: https://ec.europa.eu/digital-single-market/en/policies/digitising-european-industry (accessed on 13 April 2015).
3. Sadeghi, A.; Wachsmann, C.; Waidner, M. Security and privacy challenges in industrial Internet of Things. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference, San Francisco, CA, USA, 8–12 June 2015; pp. 1–6. doi:10.1145/2744769.2747942. [CrossRef]
4. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and Other Botnets. *Computer* **2017**, *50*, 80–84. doi:10.1109/MC.2017.201. [CrossRef]
5. Lear, E.; Romascanu, D.; Droms, R. Manufacturer Usage Description Specification (RFC 8520), Internet Engineering Task Force, 2019. Available online: https://tools.ietf.org/html/rfc8520 (accessed on 24 October 2019).
6. National Institute of Standards and Technology. Securing Small-Business and Home Internet of Things Devices, NIST Special Publication 1800-15, 2019. Available online: https://www.nccoe.nist.gov/publication/1800-15/ (accessed on 24 June 2019).

7. Li, W.; Le Gall, F.; Spaseski, N. A Survey on Model-Based Testing Tools for Test Case Generation. In *Tools and Methods of Program Analysis*; Itsykson, V., Scedrov, A., Zakharov, V., Eds.; Springer International Publishing: Cham, Switzerland, 2018; Volume 779, pp. 77–89. doi:10.1007/978-3-319-71734-0_7. [CrossRef]
8. Ohba, Y.; Patil, B.; Forsberg, D.; Tschofenig, H.; Yegin, A.E. Protocol for Carrying Authentication for Network Access (RFC 5191), Internet Engineering Task Force, 2008. Available online: hhttps://tools.ietf.org/html/rfc5191 (accessed on 24 October 2019).
9. Vollbrecht, J.R.; Aboba, B.; Blunk, L.J.; Levkowetz, H.; Carlson, J. Extensible Authentication Protocol (RFC 3748), Internet Engineering Task Force, 2004. Available online: https://tools.ietf.org/html/rfc3748 (accessed on 24 October 2019).
10. Zarca, A.M.; Bernabe, J.B.; Trapero, R.; Rivera, D.; Villalobos, J.; Skarmeta, A.; Bianchi, S.; Zafeiropoulos, A.; Gouvas, P. Security Management Architecture for NFV/SDN-aware IoT Systems. *IEEE Internet Things J.* **2019**, *6*, 8005–8020 .doi:10.1109/JIOT.2019.2904123. [CrossRef]
11. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow - enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69. doi:10.1145/1355734.1355746. [CrossRef]
12. Rosen, R.; von Wichert, G.; Lo, G.; Bettenhausen, K.D. About The Importance of Autonomy and Digital Twins for the Future of Manufacturing. *IFAC* **2015**, *48*, 567–572. doi:10.1016/j.ifacol.2015.06.141. [CrossRef]
13. Sivanathan, A.; Sherratt, D.; Gharakheili, H.H.; Radford, A.; Wijenayake, C.; Vishwanath, A.; Sivaraman, V. Characterizing and classifying IoT traffic in smart cities and campuses. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Atlanta, GA, USA, 1–4 May 2017. doi:10.1109/INFCOMW.2017.8116438. [CrossRef]
14. Polk, T.; Souppaya, M.; Barker, W.C. Mitigating IoT-Based Automated Distributed Threats, National Institute of Standards and Technology and National Cybersecurity Center of Excellence, 2017. Available online: https://www.nccoe.nist.gov/sites/default/files/library/project-descriptions/iot-ddos-project-description-\draft.pdf (accessed on 24 October 2019).
15. Voas, J.; Kuhn, R.; Laplante, P.; Applebaum, S. NISTIR 8222: Internet of Things (IoT) Trust Concerns, National Institute of Standards and Technology, 2018. Available online: https://csrc.nist.gov/publications/detail/nistir/8222/draft (accessed on 24 October 2019).
16. Caspar Schutijser. Towards Automated DDoS Abuse Protection Using MUD Device Profiles. Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 2018.
17. Hamza, A.; Ranathunga, D.; Gharakheili, H.H.; Roughan, M.; Sivaraman, V. Clear as MUD: Generating, Validating and Applying IoT Behaviorial Profiles (Technical Report). *arXiv* **2018**, arXiv:1804.04358.
18. Hamza, A.; Ranathunga, D.; Gharakheili, H.H.; Benson, T.A.; Roughan, M.; Sivaraman, V. Verifying and Monitoring IoTs Network Behavior using MUD Profiles. *arXiv* **2019**, arXiv:1902.02484.
19. Droms, R. Dynamic Host Configuration Protocol (RFC 2131), Internet Engineering Task Force, 1997.Available online: https://tools.ietf.org/html/rfc2131 (accessed on 24 October 2019).
20. Institute of Electrical and Electronics Engineers. IEEE Standard for Local and metropolitan area networks—Station and Media Access Control Connectivity Discovery. In *IEEE Std 802.1AB-2016*; Institute of Electrical and Electronics Engineers: Piscataway, NJ, USA, 2016; pp. 1–146. doi:10.1109/IEEESTD.2016.7433915. [CrossRef]
21. Institute of Electrical and Electronics Engineers. IEEE Standard for local and metropolitan area networks—Secure Device Identity, 802.1AR, 2018. Available online: https://1.ieee802.org/security/802-1ar/ (accessed on 24 October 2019).
22. Garcia-Morchon, O.; Kumar, S.S.; Sethi, M. Internet of Things Security: State of the Art and Challenges (RFC 8576), 2019. Internet Engineering Task Force, 2019. Available online: https://tools.ietf.org/html/rfc8576 (accessed on 24 October 2019).
23. Hamza, A.; Gharakheili, H.H.; Sivaraman, V. Combining MUD Policies with SDN for IoT Intrusion Detection. In Proceedings of the 2018 Workshop on IoT Security and Privacy, Budapest, Hungary, 20 August 2018; ACM: New York, NY, USA, 2018; pp. 1–7.
24. Hamza, A.; Gharakheili, H.H.; Benson, T.A.; Sivaraman, V. Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity. In Proceedings of the 2019 ACM Symposium on SDN Research, San Jose, CA, USA, 3–4 April 2019; pp. 36–48.

25. Al-Shaboti, M.; Welch, I.; Chen, A.; Mahmood, M.A. Towards Secure Smart Home IoT - Manufacturer and User Network Access Control Framework. In Proceedings of the IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), Krakow, Poland, 16–18 May 2018; pp. 892–899.
26. Ranganathan, M. Soft MUD: Implementing Manufacturer Usage Descriptions on OpenFlow SDN Switches. In Proceedings of the International Conference on Networks (ICN), Valencia, Spain, 24–28 March 2019.
27. Jethanandani, M.; Blair, D.; Huang, L.; Agarwal, S. YANG Data Model for Network Access Control Lists (RFC8519). Internet Engineering Task Force, 2019. Available online: https://tools.ietf.org/html/rfc8519 (accessed on 24 October 2019).
28. Bray, T. The JavaScript Object Notation (JSON) Data Interchange Format (RFC8259). Internet Engineering Task Force, 2017. Available online: https://tools.ietf.org/html/rfc8259 (accessed on 24 October 2019).
29. Vollbrecht, J.; Holdrege, M.; Laat, C.; Calhoun, P.; Gommans, L.; Farrell, S.; Bruijn, B.d.; Gross, G.; Spence, D. AAA Authorization Framework (RFC 2904). Internet Engineering Task Force, 2000. Available online: https://tools.ietf.org/html/rfc2904 (accessed on 24 October 2019).
30. Hurst, R.; Aboba, B.; Simon, D. The EAP-TLS Authentication Protocol (RFC 5216). Internet Engineering Task Force, 2008. Available online: https://tools.ietf.org/html/rfc5216 (accessed on 24 October 2019).
31. Bersani, F.; Tschofenig, H. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol Method (RFC 4764). Internet Engineering Task Force, 2007. Available online: https://tools.ietf.org/html/rfc4764 (accessed on 24 October 2019).
32. Garcia-Carrillo, D.; Marin-Lopez, R. Lightweight CoAP-Based Bootstrapping Service for the Internet of Things. *Sensors* **2016**, *16*, 358. doi:10.3390/s16030358. [CrossRef] [PubMed]
33. Perez, S.; Garcia-Carrillo, D.; Marin-Lopez, R.; Hernandez-Ramos, J.L.; Marin-Perez, R.; Skarmeta, A.F. Architecture of security association establishment based on bootstrapping technologies for enabling secure IoT infrastructures. *Future Gener. Comput. Syst.* **2019**, *95*, 570–585. doi:10.1016/j.future.2019.01.038. [CrossRef]
34. Aboba, B.; Simon, D.; Eronen, P. Extensible Authentication Protocol (EAP) Key Management Framework (RFC 5247). Internet Engineering Task Force, 2008. Available online: https://tools.ietf.org/html/rfc5247 (accessed on 24 October 2019).
35. Sarikaya, B.; Sethi, M.; Garcia-Carillo, D. Secure IoT Bootstrapping: A Survey. Internet Engineering Task Force, 2018. Available online: https://tools.ietf.org/id/draft-sarikaya-t2trg-sbootstrapping-05.html (accessed on 24 October 2019).
36. Alliance, Z. ZigBee IP Specification. ZigBee Alliance, 2013. Available online: http://www.sandelman.ca/tmp/6tisch/13002r01ZB_Marketing-ZigBee_IP_Specification_Public_Download.\pdf (accessed on 24 October 2019).
37. European Telecommunications Standards Institute. Machine-To-Machine Communications (M2M); Functional Architecture. ETSI TS 102 690, Sophia Antipolis, France, 2013. Available online: https://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60/ts_102690v020101p.pdf (accessed on 24 October 2019).
38. Kanda, M.; Chasko, S. PANA applicability in constrained environments. In Proceedings of the Workshop on Smart Object Security, Paris, France, 23 March 2012. Available online: http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/papers/MitsuruKanda.pdf (accessed on 24 October 2019).
39. Eronen, P.; Kaufman, C.; Nir, Y.; Hoffman, P. Internet Key Exchange Protocol Version 2 (RFC 5996). Internet Engineering Task Force, 2010. Available online: https://tools.ietf.org/html/rfc5996 (accessed on 24 October 2019).
40. Aboba, B.; Calhoun, P.R. RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP) (RFC 3579). Internet Engineering Task Force, 2003. Available online: https://tools.ietf.org/html/rfc3579 (accessed on 24 October 2019).
41. Willens, S.; Rubens, A.C.; Rigney, C.; Simpson, W.A. Remote Authentication Dial In User Service (RFC 2865). Internet Engineering Task Force, 2000. Available online: https://tools.ietf.org/html/rfc2865 (accessed on 24 October 2019).
42. Burbank, J.; Mills, D.; Kasch, W. Network Time Protocol Version 4: Protocol and Algorithms Specification (RFC 5905). Internet Engineering Task Force, 2010. Available online: https://tools.ietf.org/html/rfc5905 (accessed on 24 October 2019).

43. Rescorla, E.; Modadugu, N. *Datagram Transport Layer Security Version 1.2* (RFC 6347). Internet Engineering Task Force, 2012. Available online: https://tools.ietf.org/html/rfc6347 (accessed on 24 October 2019).
44. Enns, R.; Bjorklund, M.; Schoenwaelder, J. Network Configuration Protocol (RFC 6241). Internet Engineering Task Force, 2011. Available online: https://tools.ietf.org/html/rfc6241 (accessed on 24 October 2019).
45. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS—Towards an Open, Distributed SDN OS. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 17–22 August 2014; pp. 1–6. doi:10.1145/2620728.2620744. [CrossRef]
46. Panetta K. Gartner Top 10 Strategic Technology Trends for 2019. Gartner, 2018. Available online: https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/ (accessed on 24 October 2019).
47. Shao, G.; Kibira, D. Digital manufacturing: requirements and challenges for implementing digital surrogates. In Proceedings of the 2018 Winter Simulation Conference, Gothenburg, Sweden, 9–12 December 2018; doi:10.1109/WSC.2018.8632242. [CrossRef]
48. Dunkels, A.; Gronvall, B.; Voigt, T. Contiki, a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, Tampa, FL, USA, 16–18 November 2004; pp. 455–462. doi:10.1109/LCN.2004.38. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).