Beyond IoT Manufacture Usage Description Deployments: Challenges and Future Directions

Abstract—**Due to the advancement of IoT devices in both domestic and industrial environments, the need to incorporate a mechanism to build accountability in the IoT ecosystem is paramount. In the last few years, various initiatives have been started in this direction addressing many socio-technical concerns and challenges to build an accountable system. One of the solution in this direction is deployment and use of Manufacturer Usage Descriptions (MUD) that are provided by IoT device manufacturers. MUD implementation is challenging not only due to the diversity of the IoT devices and manufacturer/operator/regulators but also due to the incremental integration of MUD-based flow control in the already existing Internet infrastructure. To provide a better understanding of these challenges, in this work, we explore and investigate the prototypes of three implementations proposed by different research teams and organisations, useful for the community to understand which are the different features implemented by the existing technologies. Additionally, we propose a MUD capable architecture, which integrate a User Policy Server to face up to security issues of the MUD specification itself. Hence, we provide a comprehensive survey of the challenges and future directions to address some of these challenges.**

Index Terms—**IoT, IoT Traffic Regulation, Accountability, Traffic Filtering, Anomaly Detection**

I. Introduction

Due to the advancement and adaptation of smart IoT systems in recent years, billions of sensors and actuator devices from thousands of vendors are connected to Internet. The data generated from these devices generally have spatial and temporal dimensions, which have more contextual value to their local users than remote users. However, in many use-cases, the collective data generated by distributed sensors/actuators bring value to their remote users. To meet the remote access and connectivity requirements, the standard Internet protocols and Infrastructure is used. The Internet connectivity not only speeds up the process of advancing new global and remote services but also make these devices more vulnerable to new security, data, and privacy attacks similar to traditional internet connected machines. To minimise possible threats, IoT deployments need to provide for various security requirements: authentication, confidentiality, integrity, non-repudiation and availability. However, due to resource constraints and heterogeneity of current IoT devices, deployment of full-fledged security solutions is not possible. Therefore, smart and distributed solutions are needed. One potential solution is to restrict the traffic communication pattern from these devices so that the attack surface is reduced. To simplify things and to bring progress in this direction, manufacturers of IoT devices should take responsibility and authority to provide legibility of the traffic generating from their IoT devices, and end-users who are also network managers in domestic settings should have user-friendly tools to analyse, detect and control over their IoT devices.

Giving the manufacturer the responsibility to provide the detailed description of the devices in terms of usage description, has been investigated [1] and an initiative to formalise specifications by IETF (RFC 8520) [2]. The research work investigating the Manufacturer Usage Descriptions (MUD) to control and limit traffic and using it to detect anomaly detection are studied by Hamza et al. [3]. In our work we present the summary of the existing work in this direction and identify many research questions and challenges that need future investigation. In our work, we present our solutions to address some of these challenges by focussing on the deployment of manufacturer-provided MUD along with user-defined policies for COTS devices.

Contributions: In this paper, we analyse state-of-the-art MUD deployment scenarios and discuss challenges and shortcoming with each approach. We present the User Policy Server, which allows to extend the traditional MUD architecture, in order to address different security problems that can not be managed directly by MUD. This extension, gives the opportunity to introduce new MUD Files (hereafter called UPS Files, in order to be distinguished from those stored in the MUD File Server), with more restrictive rules for the internal communication, and other possibilities of network management. Nevertheless, this proof-of-concept presents new challenges that will be analysed in this paper. Hence, we present a list of future work which we think is a contribution to research community who are working in this direction.

The rest of the paper is organised as follows. In Section II, we describe an overview of the MUD workflow in a home environment and describe how MUD can isolate the IoT devices from potential external and internal security attacks and provide short descriptions of three MUD deployments. In Section III, we present current open challenges and their security impacts from our knowledge gained by reviewing the three MUD deployments. In Section, IV, V and V-A, we list some of the security challenges which could be solved by our solutions and then present our solutions and evaluations in the respective Sections. The related work in this direction so far and future work we discuss in Section VI and VII respectively.

II. MUD overview

According to IETF (RFC 8520) [2], a MUD deployment should consist of three architectural building blocks: (1) The

Manufacturer User Description (MUD) file, that is created by the device manufacturer to describe the device and its expected network behaviour. (2) A uniform resource locator (URL), that is used to retrieve the MUD file from manufacturer's server when a device is added to a home or a small business network. (3) A mechanism for local network management systems to retrieve the description. The workflow between these components is shown in Fig 1. In ①, when a device added in the network, it sends MUD-URL with a X.509 certificate or DHCP or LLDP (depending on the implementation). In ②, the local router (As Network Access Device (NAD)) sends the MUD-URL to AAA server for authentication of the request. In ③, AAA server sends the authenticated MUD-URL to MUD manager. The MUD Manager, in case of X.509 authentication, validate the signature and request (④) and retrieve MUD file from the manufacturer's file server (⑤). The MUD file is a YANG-based JSON file (IETF RFC 7951) [4] signed with a public key signature from the device manufacturer. The MUD manager validates and process the MUD file and store file locally for until its certificate is valid and generate Access Control Lists (ACL). In ⑥, the MUD manager sends ACL to NAD. The NAD enforces ACL rules it received from the MUD manager to all the traffic passing through it.

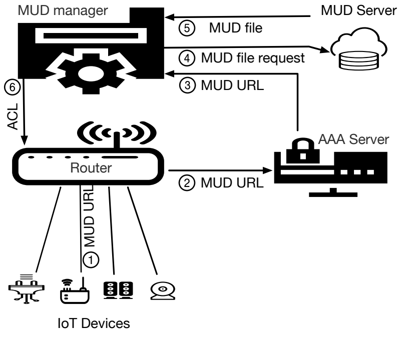

Figure 1: MUD Workflow

From the workflow and the MUD specification, we can conclude that MUD specification has provide enough flexibility and scope to choose different implementations to incorporate requirements of various deployment scenarios. In our work, we focus on investigating three different implementations of MUD deployments: Cisco proof-the-concept (PoC) implementation, National Institute of Standards and Technology (NIST) [5], [6] proposed SDN based implementation and OpenSource MUD [7] developed by a consortium of companies (Cable Labs, Cisco, CTIA, Digicert, ForeScout, Global Cyber Alliance, Patton, and Symantec) in device manufacturing and network security, coordinated by a cyber security firm, MasterPeace Solutions. We present comparison of these implementations in the Table I and provide brief descriptions in this section.

A. Cisco MUD PoC

The Cisco MUD deployment is an open-source proof-of-concept (PoC) implementation (as shown in Figure 2) that is intended to introduce advanced users and engineers to the MUD concept. The current version does not provide automated MUD manager implementation, and some protocol features are not supported currently [8]. The PoC deployment is designed with a single device serving as the MUD manager and *FreeRADIUS server* (as a AAA Server) that interfaces with the Catalyst 3850-S switch over TCP/IP. The Catalyst 3850-S switch contains a DHCP server that is configured to extract MUD URLs from IPv4 DHCP transactions. In the implementation, DHCP and LLDP are leveraged for sending the MUD URL by the IoT devices. The Cisco MUD Manager is built as a callout from *FreeRADIUS* Server and uses MongoDB to store policy information. The MUD manager is configured from a JSON file that will vary slightly based on the installation. This configuration file provides a number of static bindings and directives as to whether both egress and ingress ACLs should be applied [8].

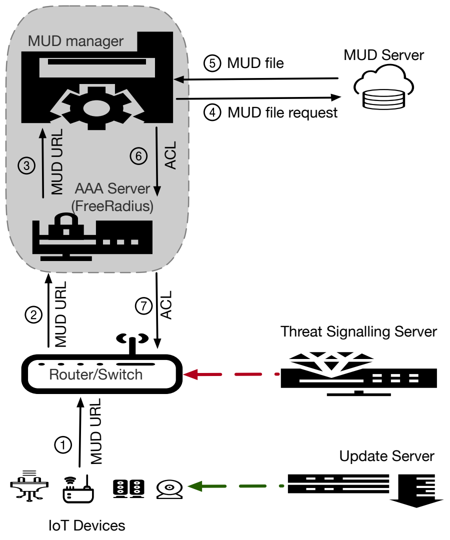

Figure 2: Logical Architecture of Cisco MUD implementation

Along with the standard MUD implementation, Cisco initial architecture also proposed two additional components, Threat Signalling Server and Update Server. The requirements for DDoS Open Threat Signaling (DOTS) is described in IETF RFC 8612 [9].

B. NIST MUD

This implementation is based on the concept of Software Defined Networking (SDN) and uses OpenFlow SDN switches. An OpenFlow switch implements flow rules in the data plane by arranging rules in more than one flow tables. The SDN controllers take the responsibility of inserting and updating the rules dynamically (when a packet arrives at the controller) or proactively when the switch connects to the controller [5]. Ranganathan et al. [6] presented an implementation using SDN flow rules in three flow tables. this implementation is focused on achieving scalability of the class-based rules at

SDN switch. The first two flow tables classify source and destination MAC addresses whereas, the third table implements the MUD Access Control Entries (ACEs) with rules that stated in terms of the packet classification metadata that is assigned in the first two tables. Once this flow pipeline is generated the packet being finally sent to a table that implements L2Switch flow rules which is provided by another application. For these reasons general Network ACEs are not implemented by this model. The implementation highlights are the following:

- implements MUD-defined ACLs (including Manufacturer, Controller, Model classes) on SDN Switches.
- uses IETF published YANG models, thus supports model driven design approach.
- implements DHCP or directly administered MUD profiles; supports DHCP support without modifications to the DHCP server; all DHCP interactions are handled in the SDN controller.
- supports linear scalability of the rule tables by supporting $O(N)$ flow rules for N distinct MAC addresses at a SDN switch.
- implements dynamic DNS.

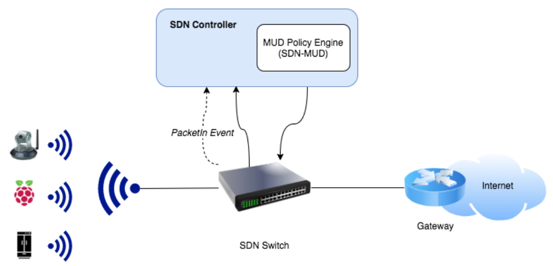

Figure 3: NIST-MUD Architecture

C. Open Source MUD

Open Source MUD (osMUD) implementation is developed by a consortium of device manufacturing and network security companies and implementation source code is maintained at github[1]. The architecture of osMUD is shown in Fig. 4; in the current implementation MUD Manager is directly run on the router. The osMUD is currently designed to easily build, deploy, and run on the OpenWRT platform and there is integrations with the *dnsmasq*, that is designed to be lightweight and have a small footprint, suitable for resource constrained routers and firewalls and provides network infrastructure services, such as DNS, DHCP, *router advertisement* and *network boot* for small networks and OpenWRT firewall services [7]. The osMUD could be compiled outside of OpenWRT for most C compatible environments. For running the current implementation outside OpenWRT requires a compatible firewall and a MUD compliant DHCP server which can extract MUD url from the DHCP header packet.

[1]https://github.com/osmud/osmud

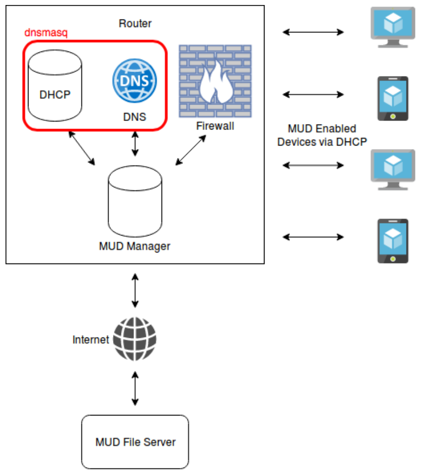

Figure 4: osMUD Architecture

III. Findings: Limitations and Security issues

In this section, we discuss the limitations and the security issues of the three prototypes described in the previous section. We investigated these implementations by setting a test environment in our lab. We understand these implementations are proof-of-concepts and are not fully implemented versions, but we thought this is a good starting point to understand the pros and cons of each implementations, which provides a motivation for further work in this context. Furthermore, this is the first existing analysis in this direction, which explores the challenges that must be considered during the building of a MUD capable network. At the end will be clearer which are the costs of adopting the MUD specification in an existing network.

A. Cisco MUD

As a proof-of-concept, the limitations of this implementation are mainly related to the static configuration of the MUD Manager. The MUD Manager does not invoke a DNS resolution service to automatically resolve the fully qualified domain names (FQDNs), but is performed manually by a human operator before beginning execution of the MUD manager service [8]. This limitation makes deployment of this implementation in real environments impractical due to the following reasons: (1) Scalability problem due to the manual configuration of DNS resolution. (2) Availability problem due to the fact to add new FQDN address resolutions or change existing ones, the MUD manager process needs to be restarted to pick the new DNS changes from the *JSON* configuration file. This could also represent a vulnerability because an attacker can exploit the period in which the MUD Manager is not available, to attack an IoT device or steal its identity. Furthermore, ingress Dynamic ACLs (DACLs), i.e., the traffic that is received from external sources to the network and directed to local IoT devices, are not supported by this version. Consequently, even if a MUD-capable IoT device's MUD file indicates that the IoT device is not authorized to receive traffic from a particular external domain, the DACL that is

Table I: Comparison of MUD implementations

Implementation	Support	URL via DHCP	URL via LLDP	URL via x509	URL via another ways	Component of an AAA system	Sig Verification
Cisco-MUD	PoC - Cisco Catalyst 3850	Yes (no DHCPv6)	Yes	No	No	Yes	OpenSSL
NIST-MUD	OpenVSwitch	Yes	No	No	with MAC + MUD	No	No
osMUD	Run on openwrt	dnsmasq	No	No	No	No	OpenSSL

needed to prohibit that ingress traffic will not be configured on the switch. Therefore, it will still be vulnerable to attacks originating from that domain, even though the device's MUD file makes it clear that the device is not authorized to receive traffic from that domain. Because egress DACLs, i.e., DACLs that pertain to traffic that is sent from IoT devices to an external domain, are supported. However, if an attacker is able to get packets to an IoT device from an outside domain, it will not be possible for the attacker to establish a TCP connection with the device from that outside domain, thereby limiting the range of attacks that can be launched against the IoT device [8].

B. NIST-MUD

This scheme requires that a packet must be processed at the controller and a rule installed before packet processing may proceed. The first rule in the MAC address classification stage (first two flow tables) that is installed when the switch connects, sends the packet to the controller but not to the next table. Thus, a packet may not proceed in the pipeline before it can be classified. This might be necessary if strict ACL dictated behaviour is required. Still, there are some resultant performance consequences, i.e., a disconnected or failed controller causes a switch failure because no packets from a newly arriving device can get through prior to the classification rule being installed. To address this problem, this implementation loosens up the interpretation of the ACE specification. It defines a "relaxed" mode of operation where packets can proceed in the pipeline while classification flow rules are being installed. This may result in a few packets being allowed to continue, in violation of the MUD ACEs with the condition that the system will become eventually compliant to the MUD ACEs. Consequently, these packets could get through prior to the classification rule being installed at the switch. This could result in a temporary violation of the ACEs [6].

C. Open Source MUD

As highlighted in the comparison (Table I), the current version 0.2.0 of osMUD does not validate the signature across the MUD file. If left unimplemented, this could lead to a serious vulnerability, which a potential attacker could exploit both from external and internal network, some of these attacks are: (1) intercept the communication between the MUD Manager and the MUD File Server, (2) create its own MUD File, in order to allow the desired communication (i.e. to conduct a dos attack) and send this file created to the MUD manager, (3) intercept the communication between the IoT device and the MUD Manager when DHCP or LLDP is used, and (4) change the MUD URL with another one that refers to a MUD File Server that provide MUD File forged by the attacker. Furthermore, the current implementation does not have MUD file rules for lateral movement, thus attackers can progressively move through a network, searching for targeted key data and assets. These vulnerabilities could be avoided in further versions.

IV. Security Challenges

In this section, we summarise the potential security implications of the three MUD implementations. None of the implementations uses X.509 option to emit the MUD URL and provides support through DHCP or LLDP. Therefore, if a device is compromised, the MUD URL issued through it can potentially be spoofed. Conversely, it is secure when the device uses the X.509 extension since the MUD URL is added to the certificate by the manufacturer. This means that the MUD URL emitted by an X.509 device can not be spoofed without detection, even if the device is exploited. Further security issues are related to the MUD Specification itself. It does not provide any inherent security protections to IoT devices themselves. If a device's MUD file permits an IoT device to receive communications from a malicious domain, traffic from that domain can be used to attack the IoT device. Similarly, if the MUD file permits an IoT device to send communications to other domains, and if the IoT device is compromised, it can be used to attack those other domains. Users implementing MUD are advised to keep some security considerations in mind. These are directly provided by the specification [2] in the *Security Consideration* section. However, we list some additional security challenges which need further addressing: (1) How to prevent a IoT device from a compromised IoT device's local attack in the local environment. (2) How to handle DHCP lease when a device acquires a new IP, and the MUD Manager needs to update ACL in the corresponding NAD. DHCP lease time may affect the performance and functionality of the MUD manager, and this needs further investigation. (3) When MUD signing certificate gets expires while MUD file is still deployed and in function locally; MUD Manager should fetch a newly signed file with signature and updates rules without interrupting the IoT device availability and functionality. (4) We have found, MUD enforced rules does not provide sufficient fine-grained control. Individual users may have their deployment setup with may require additional rules; thus a mechanism to incorporate the user policy to filter traffic to provide more fine-grained control is needed. (5) How to incorporate updates from Threat Signalling

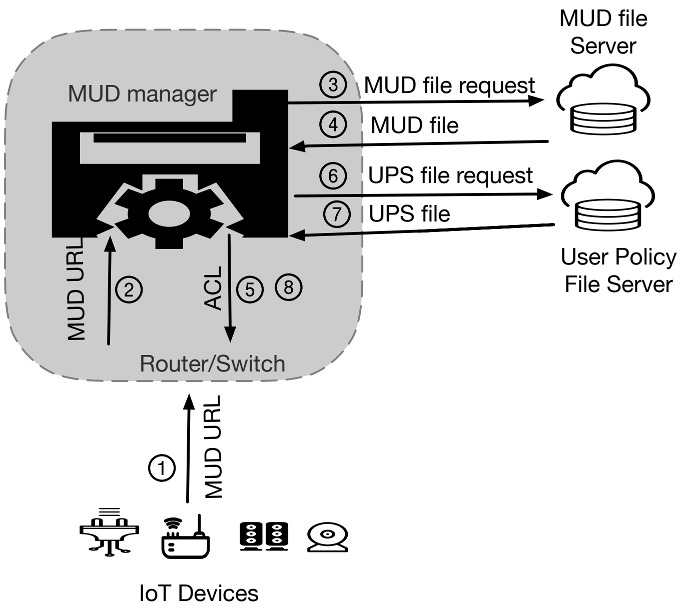

Figure 5: Our setup: osMUD with remote MUD file server and local User Policy Server.

Server. The further question and investigation in this direction is - should they go through MUD file or through a generalised common server updates (and filters) for all IoT devices.

V. User Policy Server in a MUD capable network

In this section we present some of the possible solutions of the problems we listed in the previous section. We investigated solutions for some of the security issues using Open Source MUD implementation [7]. Our local setup is shown in the Figure 5. In this architecture we have these main components: (1) OpenWRT router running osMUD Manager (2) Manufacturer MUD file server running remotely and (3) User Policy Server (UPS) running locally on a local server. In our current proof-of-concept (PoC) version, the User Policy Server is a *restful* server whose architecture is similar to the Manufacturer MUD File Server, it stores UPS Files and sign them using *openssl*. The UPS gives the chance to an administrator/end-user of an internal network, to add new rules using YANG compliant JSON files. In the UPS, to include new rules for an IoT device, we use MAC address of the IoT device to represent the UPS files, thus the following nomenclature is being followed:*<device-mac-address>.json*. The usage of USP files, can be useful in different scenarios in which the rules produced by the *manufacturer* mud files, are less restrictive than the one desired by an administrator and end-user. For example, in the internal local communications, it is possible to have malicious actors, which could generate device behaviours not prevented by the manufacturer. Hence, giving the possibility to an administrator to add new rules by using the USP files, allows to increase the level of security in the local network by exploiting the existing architecture of osMUD.

To incorporate USP with osMUD architecture, we extended osMUD implementation to allow us to incorporate following workflow: (1) MUD Manager retrieves the MUD file from the manufacturer MUD file server, (2) MUD Manager verify and parse files and generate ACL and insert ACL on Netgear router. (3) MUD Manager asks to the UPS if there is another UPS file for the device. (4) If yes, MUD Manager retrieves the file, verify and parse files and generate ACL and insert ACL on Netgear router. Thus, at the end of this procedure we have the union of the manufacture rules (MR) and the administrator (AR).

Our current architecture as shown in Figure 5 generate redundancy of rules, which is not good when we have limited storage. However, this can be simply solved by checking which are the rules already present in the IP tables and avoid to insert the equivalent ones. The current architecture also lacks in resolving the conflicting rules, for example, the mud file created by the manufacture of an IP camera, allows the communication with two different external hosts, say, $A$and B, but the UPS file allows only A and C. In our current system, IP camera is allowed communication with all three hosts (A, B and C). But, if we provide UPS rules priority over MUD rules, IP camera should be allowed with only A and C. By default, all MUD whitelist rules are inserted, so adding new whitelist rules using UPS is easy however, for removing/reversing MUD rules using UPS needs a thoughtful consideration. Therefore, in this work we have further investigated these four extensions of our architecture:

- **Automatic UPS File generation**: Use a tool that automatically generates the YANG compliant JSON files by monitoring the internal traffic ([10]).
- **New MUD File structure for the UPS Files**: Extension of current file structure to obtain more restrictive rules such as packet rate, maximum packets, time restrictions, maximum ingress and egress points. To implement this, we require extending osMUD Manager, parsing capabilities, and rule enforcement logic.
- **Publish/subscribe architecture**: it is possible to extend the UPS infrastructure with a publish/subscribe model, where server becomes the publisher and the router becomes subscriber. The server can publish, by understanding the traffic behaviour, new rules that must be inserted in the IP tables of the router. In this architecture we need signature based authentication between the router and server, otherwise an attacker could impersonate the server and insert less restrictive rules.
- **GUI based Policy Engine**: A policy engine, which takes rules from MUD and UPS and signal users if there are any conflicting rules.

A. Evaluations

Even though our setup shown in Figure 5 is a prototype and still in development phase, we performed some of the basic evaluations to check the feasibility of running such systems at a router in home or small business environment. We run experiments in our lab environment on an OpenWRT compliant Netgear router (model WNDR 3700v2) and compared performance of osMUD [7] and our solution based on osMUD (shown in Figure 5). The tests are performed by considering the time spent (latency calculation) by the MUD Manager (1) retrieving the MUD file, (2) verify and storing the file locally at the router, and (3) processing file and installing rules. The experiment is setup to emulate the router booting phase; during booting phase, router aggregates the dhcp requests from all connected IoT devices in a file. The MUD-URL from the file then processed by MUD Manager one by one to retrieve MUD files. This scenario happens when IoT devices are turned on before the router boot process and thus represent the worst case scenario as all IoT devices MUD files need to be retrieved at the same time. The MUD manager implementation by osMUD retrieves files sequentially from the MUD file servers. For testing purpose we use MUD file server hosted by osMUD[2]. For UPS, we hosted a local server in our internal network. The size of MUD and UPS files range between 2-6K. In our current implementation, MUD Manager could have repeated twice (osMUD with USP) all three operations:(1) retrieving the MUD file, (2) verify and storing the file locally at the router, and (3) processing file and installing rules.

The figure 6 illustrates the performance of the router in the osMUD and our work of UPS with osMUD. Both the tests have been repeated 20 times. In our evaluation settings, we set USP file request and MUD file request as: $(1,1),(2,2),(2,4),(3,8),(6,16)$ the numbers in the brackets (a,b) indicate: a = number of devices, which requested for a UPS file; b = number of devices, which request MUD file.

Starting from only one device, we can see that the performances registered by osMUD with UPS are a bit worse than the osMUD. This is because of the additional request that must be done to the local server. Nevertheless, the results obtained are better than expected; the mean of latency in osMUD with UPS is 4.1 seconds as compared with 2.9 seconds of the osMUD, therefore, in average only 1 second worse. This is also due to the fact, that in our current setup we run UPS server in the local environment, so the download problems are limited to the router bit rate. These results are promising and motivate us to the feasibility of running a User Policy Server to include user rules.

Overall, the time spent on doing these operations is not linear, because of connectivity that is not completely stable and this can influence the download time. But, at the same time is not exponential, so the number of device does not weigh so much on the performance. This is not true for our model. In fact, in this case all depends on how many devices have an additional mud file. For example, in the case of 16 devices (Figure 6) we have 6 devices, which request the additional mud file. This leads to a spike of 55 seconds compared with 45 seconds of the classical model. Furthermore, in both the cases with 16 devices, the time exceeds more then twice the time taken by 8 devices. This leads us to think that when the number of devices is over a threshold, the router starts to be slower.

[2]https://mudfiles.nist.getyikes.com/

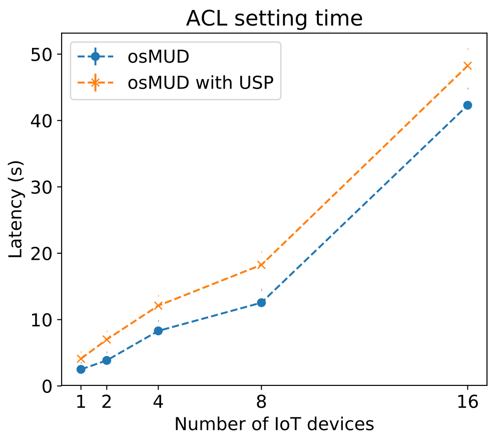

Figure 6: Access Control List (ACL) Rule setting time with osMUD and osMUD with USP.

In conclusion, we can say that the growth becomes exponential after 4 devices. Of course, it represents the worst case situation, due to a router failure or to an unexpected reboot of it. In a steady environment in which we add one or two devices at a time, the performance registered are not bad in both the cases, even if two devices request an additional mud file (Figure 6).

VI. Related Work

The use of MUD as an isolation-based defensive mechanism to restrict traffic generated from the IoT devices is still in its early phase, therefore, only a few deployment scenarios and proof-of-concept (PoC) implementations currently exist. Andalibi et al. [11] proposed an extended MUD functionalities to include traffic patterns - such as message size, peak-rate and maximum message transmissions within a time-period and proposed to implement those filtering at the fog network firewall. However, the paper does not provide any implementation and also does not provide further details related to how these traffic control lists will be enforced at the fog networks and how this solution will be scalable in a large organisation. Yadav et al. [12] has proposed scenario of combining user control policies with MUD to provide fine gained control of IoT data. This work lacks to provide how to use user-based policies with MUD to control messages directly exchanged with external services. Ranganathan et al. [6] presented scalable implementation of the MUD standard on OpenFlow-enabled Software Defined Networking switches and Hamza et al. [3], [13] developed machine learning methods to detect volumetric attacks such as DoS, reflective TCP/UDP/ICMP flooding, and ARP spoofing to IoT devices in MUD compliant traffic.

VII. Conclusions and Future works

The MUD specification presents some limits in terms of security and archietecture. Evidence of that is the fact that is not intended to address network authorisation of those IoT devices that have very broad communication patterns (like Alexa and Google Home). Thus, in this paper, we have proposed a MUD capable network, which integrate osMUD as MUD manager and a User Policy Server as a supervisor of the internal communication. The introduction of an additional supervisor as a threat singnaling servers and user policy based MUD manager that interact directly in the local network, can be used to improve the security level.

Nevertheless, during the development of our proof-of-concept, some architectural and security challenges have been arisen. For example, from the osMUD perspective, there are no checking on the rules redundancy and the difference between DHCP requests from MUD capable devices and from general purpose devices, is not managed; from the User Policy Manager perspective, the introduction of new security features requires changes in the UPS File structure and consequently in the osMUD parser implementation. In conclusion, our future directions, considering the security issues not prevented by the MUD specification and the problems described above, will be focused on the extension of the functionalities of the User Profile Server and on solving implementation inconsistency problems, in order to guarantee a higher level of security. The work we presented in this paper provides a useful insight and we hope research and development community get benefit from this. Furthermore, this work opens up more discussions before we start adapting these solutions in the real deployment scenarios.

References

[1] Cisco. (2018) Manufacturer usage descriptions. [Online]. Available: https://developer.cisco.com/site/mud/

[2] E. Lear, R. Droms, and D. Romascanu, “Manufacturer Usage Description Specification,” RFC 8520, Mar. 2019. [Online]. Available: https://rfc-editor.org/rfc/rfc8520.txt

[3] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, “Detecting volumetric attacks on lot devices via sdn-based monitoring of mud activity,” in *Proceedings of the 2019 ACM Symposium on SDN Research*, ser. SOSR’19. New York, NY, USA: ACM, 2019, pp. 36–48. [Online]. Available: http://doi.acm.org/10.1145/3314148.3314352

[4] L. Lhotka, R. Droms, and D. Romascanu, “Manufacturer Usage Description Specification,” RFC 7951, August 2016. [Online]. Available: https://tools.ietf.org/html/rfc7951

[5] D. Dodson, T. Polk, M. Souppaya, W. C. Barker, P. Grayeli, M. Raguso, and S. Symington, “Securing small-business and home internet of things (iot) devices,” https://www.nccoe.nist.gov/projects/building-blocks/mitigating-iot-based-ddos, 2019.

[6] R. Mudumbai, D. Montgomery, and O. E. Mimouni, “Soft mud: Implementing manufacturer usage descriptions on openflow sdn switches,” in *International Conference on Networks (ICN)*, 2019.

[7] A. consortium of network security companies, “Open source manufacture usage specification,” https://osmud.org, 2018.

[8] D. Dodson, W. Polk, M. Souppaya, W. Barker, E. Lear, B. Weis, Y. Fashina, P. Grayeli, J. Klosterman, B. Mulugeta *et al.*, “Securing small business and home internet of things (iot) devices: Mitigating network-based attacks using manufacturer usage description (mud),” National Institute of Standards and Technology, Tech. Rep., 2019.

[9] A. Mortensen, T. Reddy, and R. Moskowitz, “DDoS Open Threat Signaling (DOTS) Requirements,” RFC 8612, May 2019. [Online]. Available: https://tools.ietf.org/html/rfc8612

[10] ayyoob. (2019) Generate mud profiles using pcap. [Online]. Available: https://github.com/ayyoob/mudgee

[11] V. Andalibi, D. Kim, and L. J. Camp, “Throwing MUD into the FOG: Defending iot and fog by expanding MUD to fog network,” in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. Renton, WA: USENIX Association, July 2019. [Online]. Available: https://www.usenix.org/conference/hotedge19/presentation/andalibi

[12] P. Yadav, V. Safronov, and R. Mortier, “Poster abstract: Enforcing accountability in smart built-in iot environment using mud,” in *The 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys 2019)*. New York: ACM, Nov 2019.

[13] A. Hamza, H. H. Gharakheili, and V. Sivaraman, “Combining mud policies with sdn for iot intrusion detection,” in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, ser. IoT S&#38;P ’18. New York, NY, USA: ACM, 2018, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/3229565.3229571

[14] osmud. (2019) Open source mud manager. [Online]. Available: https://osmud.org