Journal of Systems Architecture 110 (2020) 101779

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Enhancing Internet of Things Security using Software-Defined Networking

Bander Alzahrani[a,*], Nikos Fotiou[b]

[a] Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
[b] Mobile Multimedia Laboratory, Department of Informatics, School of Information Sciences and Technology, Athens University of Economics and Business, Athens, Greece

ARTICLE INFO

Keywords:
Access control
Software-Defined Networking
Constrained application protocol
Internet of Things

ABSTRACT

Access control technologies are fundamental for addressing the security and privacy requirements of the Internet of Things (IoT). This paper proposes an access control solution for Constrained Application Protocol (CoAP)-based IoT services. The proposed solution considers a network of a single provider that interconnects various IoT endpoints. It leverages the Software-Defined Networking (SDN) paradigm and implements application aware policy enforcement at the network level. All operations are transparent to the IoT endpoints and no modifications are required to the IoT communication protocol. Furthermore, our solution is built on standard OpenFlow, hence it is realistic and it can be easily deployed to an existing network. We prove the feasibility of our solution through a proof of concept implementation using network emulation.

1. Introduction

Nowadays, many aspects of our life are controlled–or assisted–by cyber-physical systems. The so-called Internet of Things (IoT) is already used in many domains, including agriculture, patient monitoring, home automation, well-being, smart cities, and many others. The IoT is mainly composed of devices which may be deprived of computational power, continuous network connectivity, energy, or even physical security. Therefore, it comes as no surprise that applying security solutions in this environment is a challenging problem. In this paper, we focus on a particular aspect of security, that is access control. We consider the case of a network of a single operator that interconnects various IoT devices. These devices provide resources or actuation services, and can be accessed using the Constrained Application Protocol(CoAP) [1].

In order to motivate our solution, we consider the use case of a smart city management system. This system is composed of IoT sensors (e.g., temperature sensors) and actuators (e.g., switches). Our goal is to enable system administrators to define context-aware access control policies that will mediate access to the IoT devices. More formally, we want to provide a *Mandatory Access Control* (MAC) solution where policies are centrally defined by the system administrators and cannot be modified or overridden by end users. An example of such policy, in our reference system, is the case of a switch that turns on and off street lights; in that case the system administrator could create an access control policy that defines that “street lights switches can be turned on after 8pm and turned off after 6am, and all operations should originate from the management center building”. It can be observed that this policy defines the resource (switch), the action (turn on/off), and defines constrains related to the time that an action can take place and the physical location from which it can originate.

It can be argued that it is possibly to implement such an access control system by following a distributed approach, using more powerful “gateways” attached to the IoT devices. We postulate, however, that this approach has many disadvantages, such as (i) policy management becomes difficult, since policy updates have to be “pushed” to all gateways, (ii) action “logging” is harder, (iii) a gateway may not have the necessary information to perform an access control decision (e.g., the location from which a request originated), and (iv) unauthorized requests still use network resources hence, they may be used for attacks such as Denial of Service (DoS). Instead, in this paper we follow a centralized approach where access control decisions are made by a centralized entity and they are enforces close to message senders. In order to achieve our goal, we leverage Software-Defined Networking (SDN) [2] and Edge computing [3] paradigms, and the OpenFlow [4] protocol. In summary, this paper makes the following contributions:

- We design, build, and evaluate a mandatory access control (MAC) system for the IoT.
- We propose a fine-grained context-aware access control mechanism that operates over network flows.
- We make our solution transparent to IoT endpoints.
- We build our solution using standardized protocols without requiring any modification.

The rest of this paper is organized as follows. In Section 2, we introduce CoAP and SDN, and we discuss related work in this area. We

* Corresponding author.
E-mail address: baalzahrani@kau.edu.sa (B. Alzahrani).

https://doi.org/10.1016/j.sysarc.2020.101779
Received 29 October 2019; Received in revised form 29 January 2020; Accepted 3 April 2020
Available online 26 April 2020

give a high level overview of our system in Section 3, and we present its design in Section 4. Finally, we evaluate our approach in Section 5, and present our conclusions and plans for future work in Section 6.

2. Background and related work

2.1. The constrained application protocol

The Constrained Application Protocol (CoAP) [1] aims to become the HTTP equivalent for the IoT. CoAP specifies two main entities, the CoAP *client* and the CoAP *server*. A CoAP server offers a *resource* which is identified by a CoAP-URI, i.e., a URI that follows the same syntax as HTTP URIs. A CoAP client may request to access or modify a resource using one of the CoAP methods: GET, PUT, POST, DELETE. CoAP has been designed to be used over unreliable transport protocols (e.g., UDP) for this reason it specifies acknowledgment mechanisms. A salient feature of CoAP is that it allows CoAP clients to request a resource that is not yet available. In that case, the CoAP server acknowledges the reception of the request and responds whenever the requested resource becomes available: in order for a CoAP client to be able to match a response with a request, it includes a token in the request, which is repeated by the server in the response. Our solution takes advantage of this token mechanism.

In addition to the two main entities, CoAP RFC (i.e., RFC 7257) specifies an optional entity, i.e., the CoAP proxy. A CoAP proxy can be transparent, or clients and/or servers may be aware of it. Proxies are used for providing additional functionality (e.g., caching), as well as protocol translation (e.g., CoAP-to HTTP proxies and vice versa). Our solution considers CoAP proxies that translates CoAP messages into the appropriate protocol invocations of our system.

2.2. Software-Defined Networking

Software-Defined Networking (SDN) [2] is an emerging technology that has received wide attention and has already been adopted by major network providers. SDN decouples the network control plane from the data forwarding plane. The latter is implemented by SDN switches using “rules” defined by a (logically) centralized component, i.e., the SDN *controller*. Using a separate control network, the controller “installs” flow-wide rules to switches that specify how a switch should handle network flows. The communication between SDN switches and the controller is implemented using protocols such as the OpenFlow protocol [4].

Flow rules include filters that are applied over various fields of incoming or outgoing flows, and define the actions that a switch should perform in case of a match. Generally, all flows should be matched to a rule: in case a switch is not able to find a rule for a specific flow it forwards the corresponding packet to the controller which in return responds with the appropriate instructions.

2.3. Related work

SDN technology as enabler for the IoT has been studied by numerous research efforts (for a survey in this area interested readers are referred to [5,6]). Our work is orthogonal to these efforts. We are concerned with the application layer and we design an access control mechanism for CoAP-based IoT services. Our mechanism is compatible with any protocol that uses the request-response communication paradigm to retrieve named resources, and has support for proxies (e.g., HTTP). Similarly, our solution concerns the communication among IoT endpoints and it is orthogonal to systems that secure access to IoT data *stored in a centralized powerful node*(see for example [7])

Hong et al. [8] propose an SDN-based framework for enforcing access control policies on user mobile devices in corporate networks (a.k.a Bring Your Own Device–BYOD). In contrast to our work, the solution in [8] considers a legacy network without SDN switches and proposes modifications to user devices. Our work follows the opposite approach: we make no modifications to end-devices and we rely on SDN switches to enforce our access control decisions.

Sonchack et al. [9] leverage SDN to provide in-network security mechanisms. However, in their approach they do not rely on standard OpenFlow, instead they propose and implement OpenFlow extensions that enable SDN switches to evaluate more complex rules. With these extensions, SDN switches can perform security-related operations without interacting with the controller. Similarly, Voellmy et al. [10] proposed a language, code-named Procera, for defining more complex rules that react in conditions such as user authentication, or time of the day. Our solution does not require modifications to the OpenFlow protocol. Furthermore, in order to decrease the amount of traffic between the SDN switches and the controller, we shift some of the control plane intelligence from the controller to the edge of the network.

Various systems, such as FRESCO [11] and OpenSec [12], define high level access control definition languages that are translated into OpenFlow rules. These systems can by used by our solution as mechanisms for creating access control policies.

The system proposed by River et al. [13] has the same goals with our work but in a different context: this system is tailored to robotic applications and the Robotic Operations System (ROS). ROS is a publish-subscribe system which in addition to the communicating endpoint, it considers an intermediate entity responsible for the “topic” management. The proposed solution requires modifications (in the form of extensions) to all elements of a ROS system.

Papachristou et al. [14] propose runtime and routing policies for enhancing the security and the quality of service in SDN-based IoT architectures. The work in that paper is a high level approach and does not discuss how these policies can be implemented. Our work defines a mechanism that allows SDN controllers and network edge nodes to enforce user-defined access control policies.

3. System overview

3.1. Underlay architecture

The underlay architecture of our solution is based on the systems described in [15] and [16]. In the core of our architecture there is an SDN network, at the edges of which there exist *Network Attachment Points* (NAPs). Each NAP is identified by a NAP_{id} and all NAPs know all NAP_{id}s. NAPs are connected to the SDN network using an SDN switch; in the following, when it is stated that an “OpenFlow rule is installed in a NAP” it is meant in the SDN which that the NAP uses for connecting to the network.

There are CoAP endpoints (i.e., CoAP clients and servers) attached to each NAP. Each CoAP server offers a *resource* which is associate with (at least) one CoAP URI. It should be noted that there can be multiple resources, belonging to different CoAP servers, sharing the same URI (e.g., coap://city/lights). Each NAP knows all CoAP URIs of all CoAP servers attached to it (e.g., through a configuration file or a CoAP discovery protocol such as CoRE resource directory [17]). NAPs act as CoAP proxies and CoAP clients are configured to communicate through the proxy of the NAP in which they are attached. Therefore, all CoAP requests are routed through a NAP and NAPs are able to parse these requests and extract information. All CoAP messages are sent using the IPv6 protocol, it is assumed that all CoAP requests include a token, and that a single CoAP message fits in a single network packet, i.e., packet fragmentation is not required.[1]

Packet forwarding among NAPs is implemented using Bloom filters approach [18] which has been described in [19]. In a nutshell, each link of the core network is identified by a *bit array*, referred to as the link

[1] This assumption has been made for facilitating the implementation of our solution. Our implementation can be extended to support packet fragmentation. In any case, considering that

identifier; the path that a packet should follow is encoded in a Bloom filter, referred to as the path identifier, constructed by ORing the link identifiers of the appropriate links. Path identifiers are stored in the IPv6 address fields, hence it is easy to construct the appropriate flow rules in order to achieve Bloom filter-based forwarding. In particular, and from a high level perspective,[2] each SDN switch is configured with an OpenFlow rule per interface, this rule performs a subnet mask check on the IPv6 destination address (i.e., it checks if the destination address belongs to the network defined by the netmask) and if the check succeeds the packet is forwarded from the corresponding interface; this check is performed for all interfaces, hence a packet may be forwarded to multiple interfaces (achieving this way multicast). The netmask used in each rule is the link identifier of the corresponding link.[3] The OpenFlow rules required for implementing Bloom filter-based forwarding are installed once, during the network setup phase.

Path identifiers in our architecture are bi-directional, i.e., a path identifier used for forwarding a packet from a NAP A to a NAP B, can be used for forwarding packets from B to A as well. Another interesting property of path identifiers is that they can be combined to create multicast trees. For example, given a path identifier $Path_{A\rightarrow B}$ of the path from NAP A to NAP B and another path identifier $Path_{A\rightarrow C}$ of the path from NAP A to NAP C, then $Path_{A\rightarrow B}|Path_{A\rightarrow C}$ (where | denotes bitwise OR) results to a new path identifier that can be used for multicasting packets from NAP A to NAPs B and C.

Finally, our system assumes that the SDN controller knows the network topology, all link identifiers, and the CoAP URIs associated with each NAP, and it provides a “Northbound” API that allows resource owners to install, update, or remove access control policies.

3.2. System entities and interactions

Our system is composed of the following entities.

- **System administrator**. This is a real world entity (although this role can be simultaneously assigned to multiple real world entities) which is responsible for defining the access control policies that govern access to IoT resources.
- **CoAP clients**. These are applications that interact with CoAP servers (see below) using the CoAP protocol. CoAP clients abide by the CoAP RFC and are oblivious about our access control solution.
- **CoAP servers**. These are applications that may provide a resource or an actuation service using the CoAP protocol. Similar to CoAP clients, CoAP servers abide by the CoAP RFC and are oblivious about our access control solution.
- **Policy Decision Point** (PDP). The PDP is a centralized system entity located alongside the SDN controller. It is responsible for interpreting an access control policy and deciding (based on this policy) whether or not a CoAP operation can be permitted.
- **Policy Enforcement Point** (PEP). The PEP is a distributed system entity located in each NAP. It is responsible for implementing the access control decision of the PDP.

From a high level perspective these entities interact with each other (through the underlay architecture) as follows (Fig. 1). The system administrator uses the controller’s northbound API to interact with the PDP and define the desired access control policies. A CoAP client issues a CoAP request that reaches the NAP in which the client is attached. If the PEP of the NAP does not know how to handle this request it communicates with the PDP (green line). The PDP inspects the request, examines if there exists any applicable access control policy, and makes the appropriate decision (red line). Based on the PDP decision, the PEP either allows the request to reach the intended CoAP server(s) or drops it (orange line). A positive PDP decision is accompanied by an OpenFlow rule which is used by the NAP in order to forward the CoAP request to the network. This rule may have a limited lifetime, defined (implicitly or explicitly) by the access control policy on which the policy decision was based.

3.3. Security assumptions

In this paper we assume that the SDN network is trusted, therefore it is not possible for an attacker to affect the operation of the SDN controller, to view/edit SDN control messages, or manipulate the flow rules of the SDN switches. Furthermore, NAPs are also considered trusted and a secure “network attachment” process is assumed, i.e., it should not be possible for an attacker to attach a new NAP to the network nor to attach himself to an existing NAP by bypassing the network attachment process.

Furthermore, our system assumes that system administrators are properly authenticated, therefore access control policies are protected against unauthorized modifications. Similarly, NAPs are able to authenticate CoAP servers and verify the ownership of a CoAP URI.

Our system treats all CoAP clients as anonymous users and it is not concerned with CoAP clients’ authentication. Similarly, it is assumed that the secrecy and the integrity of CoAP messages is protected using higher layer mechanisms which are out of the scope of this paper.

Given the above security assumptions, the threat model of our system considers malicious CoAP clients wishing to access a protected CoAP server; although these CoAP clients are authorized to attach to a NAP they are not authorized to access the resource in question.

4. System design

4.1. Policy definition

Policies are expressed in our system using an adapted version of the policy definition language defined in [8]. This XML-based language defines four basic elements:

- *Target*. This element defines the CoAP-URI of the resource(s) controlled by that policy. Wildcard characters can be used to indicate a CoAP-URI prefix.
- *Match*. This element defines a filter, in the form of an OpenFlow rule, that can be used for associating network flows to policies (e.g., a policy is applied if a flow has originated by a particular mac address– that may correspond to the MAC address of a NAP).
- *Predicate*. This element defines filters for context-aware matching. Currently the predicate element supports time-based and CoAP method-based filtering.
- *Actions*. This element defines the OpenFlow rules that should be installed in the NAP that sent the request.

Two policies should not use the same values for the target, match, and predicate elements. The *actions* element allows the definition of a path, which can be used as a variable in the OpenFlow rules. The path identifier is calculated at runtime and the corresponding rules are modified accordingly. Figure 1 shows an example of an access control policy definition. This policy is evaluated when the URI “coap://city/lights” is invoked form a particular network location, after 8pm, using the CoAP PUT method; a path identifier is calculated towards some NAPs and the appropriate OpenFlow rule is created.

4.2. NAP to controller communication

In the following subsection we will present operations that require communication between a NAP and the SDN controller. This functionality has been implemented as follows. As already discussed each NAP is connected to an SDN switch. These switches are configured with an OpenFlow rule that instructs them to forward all packets that use

[2] Interested readers can find details about this approach in [19]

[3] OpenFlow specifications allow “arbitrary” network masks, i.e., masks do not have be a series of 1s followed by 0s, instead they can be any bit array with size equal to the size of an IPv4/v6 address.

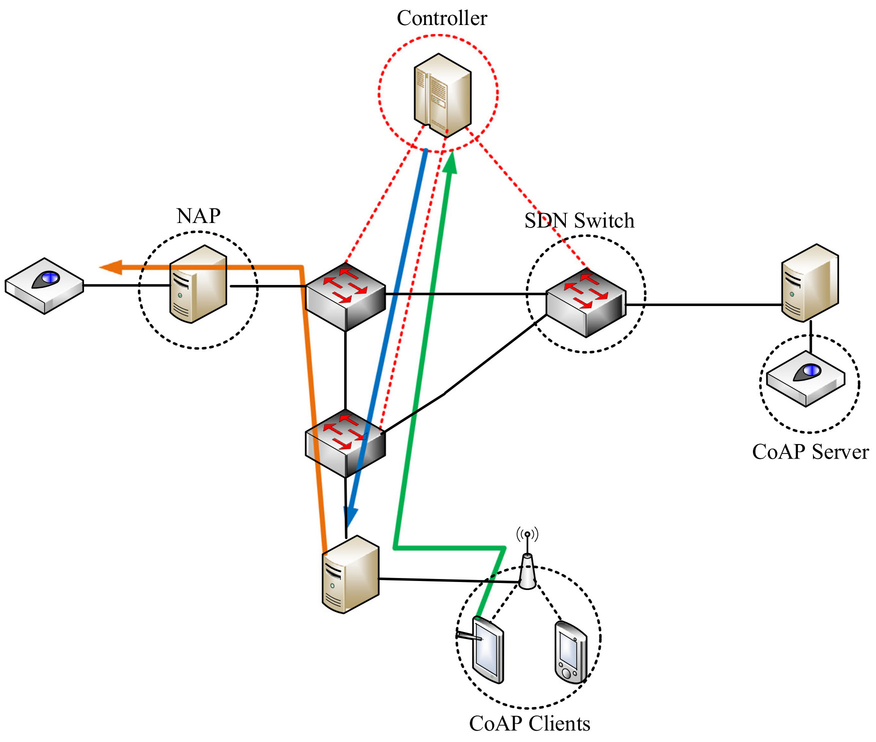
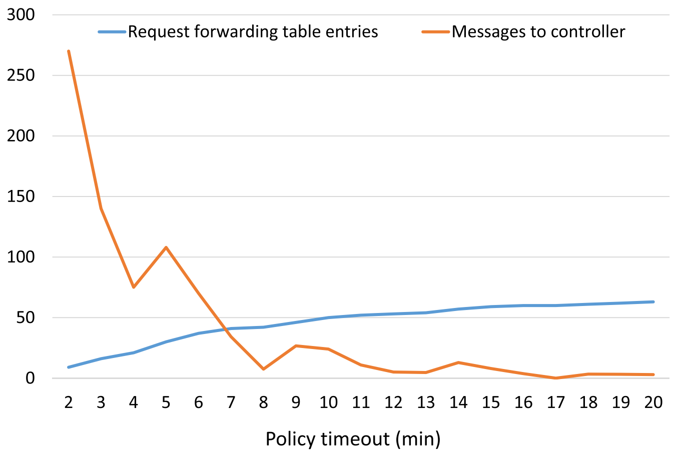


Fig. 1. High-level overview of the proposed solution. A CoAP request from a client is forwarded to the PDP (green line); the PDP installs the appropriate rules (red line); the PEP forwards the request to the CoAP server (orange line). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



Listing 1. Access control policy definition.

“00:00:00:00:00:01” as an Ethernet destination to the controller. Therefore, if a NAP sets the Ethernet destination address of a packet to “00:00:00:00:00:01”, this packet will eventually reach the controller. Similarly, when a controller receives such a packet, it creates a *Packet-out* OpenFlow message, that contains the controller’s response and instructs the SDN switch to forward this message to port in which the NAP is connected.

In the rest of this paper, we hide this detail and we simply say that “a NAP sends a packet to the controller”.

4.3. *CoAP request and response handling*

Whenever a CoAP request arrives to a NAP, the NAP should decide which path identifier to use for forwarding the request to the network. Similarly, whenever a CoAP response arrives from the network, a NAP should be able to decide the CoAP client to which it will forward the responses. In order to achieve this functionality, each NAP maintains two data structures, a *request forwarding table* and a *pending responses table*.

The request forwarding table contains rows of the form *[URI, Path$_{ID}$, expires]*. Whenever a packet that contains a CoAP request arrives at a NAP, the NAP extracts the URI of the resource included in the request, and searches the forwarding table for an existing, non-expired entry. If an entry is found, the NAP replaces the IPv6 destination field of the packet with the corresponding path identifier, and forwards it to the network. Otherwise, the NAP sends the request to the controller, the controller extracts all the required information, evaluates the appropriate access control policies, and responds accordingly. If the controller’s response includes a path identifier, then the NAP updates the forwarding table, modifies the packet, and sends it to the network.

The pending responses table contains rows of the form *[token, Client$_{IP}$]*. Whenever a packet that contains a CoAP request arrives at a NAP, the NAP extracts the CoAP client IP address and the token included in the request and updates the pending responses table accordingly. The corresponding CoAP response must include the same token, therefore, whenever a response arrives at a NAP, the NAP extracts the token, retrieves the client’s IP address from the pending responses table and replaces the path identifier of the packet with the clients address. Finally, the NAP forwards the response to the client.

5. Evaluation

We implemented our solution using Open vSwitch [20] SDN switch and the POX [21] SDN controller. We implemented CoAP endpoints and the CoAP proxy (located at the NAPs) using the libcoap library.[4] We evaluated our implementation using the mininet network emulator [22].

Our solution requires from the NAPs to maintain two data structures, as well as to occasionally forward CoAP requests to the controller. We measure this overhead using the workload of the temperature measurement sensors deployed in the testbed of SmartSantander, a large-scale smart city deployment located in Santander, northern Spain. The workload, as reported in [23], is composed of 70 temperature sensors, gener-

[4] https://libcoap.net/.

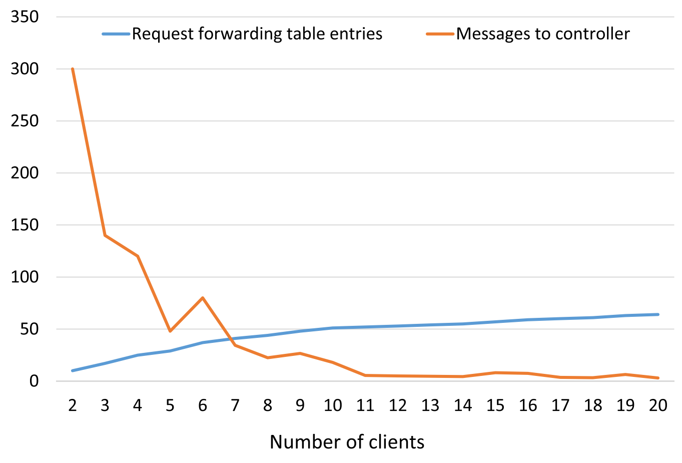


Fig. 2. Number of entries of the request forwarding table, and number of messages to the controller, as a function of the path identifier timeout.

ating a temperature measurement approximately every 5 min. We consider that each temperature sensor is identified by a CoAP URI. For each CoAP URI there is a single access control policy that “accepts” a request and creates a path identifier: the expiration time of each path identifier is used as a variable in our measurements, and as we discuss in the following subsection this creates a security-performance tradeoff. We assume that each client-side NAP can accommodate a maximum number of clients (used as a variable in our experiments, as well). Each client requests a randomly selected sensor measurement. Clients’ request interval follow a normal distribution with average 1 min. The maximum number of entries that responses table may contain equals to the number of clients connected to a NAP. This represents the extreme case that all clients have made a CoAP request and no response has been received. The size of each entry in this table is 24 bytes (16 bytes for client IPv6 address and 8 bytes for the CoAP token–this is the maximum value specified by [1]).

We now measure the number of entries in the request forwarding table of a NAP, as well as the number of requests a NAP sends to a controller. Each experiment begins with a warming up period equal to the path identifier timeout. Then we continue to run the experiment for 1 h. Fig. 2 shows the maximum number of entries of the request forwarding table and the total number of requests sent to the controller as function of the path identifier timeout. In this experiment the number of CoAP clients per NAP is set to 10. The same experiment has been performed with the number of clients varying from 5 to 20 and the results follow a similar pattern.

Fig. 3 shows the maximum number of entries of the request forwarding table and the total number of requests sent to the controller as function of the number of clients per NAP. In this experiment the path expiration time is set to 10 min.

```
<Policy ID='Policy 1'>
    <Target>
        URI='coap://city/lights'
    </Target>
    <Match>
        of_eth_src='00:04:06:08:05:06'
    </Match>
    <Predicate>
        TIME > 2000, METHOD='PUT'
    </Predicate>
    <Actions>
        <Path Id='Path1'>
            <src>NAP_A</src>
            <dst>NAP_B,NAP_C</dst>
        <Path>
        <Rule>
            nx.ipv6_dst = $Path1,
            nx_action_resubmit,
            timeout = 3600
        </Rule>
    </Actions>
</Policy>
```

Fig. 3. Number of entries of the request forwarding table, and number of messages to the controller, as a function of the number of clients of a NAP.

5.1. Security evaluation

Our solution does not consider network properties of the clients (e.g., currently it is not possible to define an access control policy based on a CoAP client network address), neither considers the context of the client applications (e.g., it does not perform user authorization and access control.) Furthermore, our solution requires that NAPs are trusted, since a misbehaving PEP (located in a NAP) may ignore a PDP decision (or a path identifier timeout) and use a path identifier that is already known.

An interesting security-performance tradeoff of our solution is related to the path identifier timeout. As shown in the previous subsection, large timeouts result in significantly less messages sent to the controller. This not only decreases the overhead imposed to the controller, but it also results in CoAP clients experiencing faster responses to their requests (since a request is forwarded by the NAP without communicating with the controller). On the other hand, while a path identifier is valid a CoAP client may access a resource for which is not anymore authorized. For this reason, path identifiers timeouts must be carefully selected.

5.2. Discussion

Our system takes full advantage of the forwarding solution presented in [19]. All OpenFlow rules required for implementing this approach are installed in all SDN switches during setup: once a packet leaves a NAP, it is forwarded directly to its destination(s) without any further communication with the controller. Furthermore, and as discussed in more detail in [15], the number of forwarding rules in only related to the number of NAPs and not to the number of the IoT devices.

Another aspect that affects the performance of our system is the number and the complexity of access control policies. Of course, this is a concern that all access control systems share. For this reason, in this paper we decided to not focus on this aspect. However, it should be noted that the only entity affected by the complexity of the access control policies is the (centralized) PDS: all other entities, including IoT devices, are oblivious to the policies.

6. Conclusions and future work

In this work we proposed an SDN-based solution for enforcing access control in CoAP-based IoT applications. Our solution does not require any modification to the IoT endpoints, and it is built using standard

OpenFlow functionality. Our solution can be deployed using existing infrastructure and imposes minimal overhead.

A strong security requirement of our solution is that the network attachments points (NAPs), which also act as the policy enforcement points (PEPs) to be trusted and to not use path identifiers that have expired. Our approach can be protected against misbehaving NAPs by using mechanisms that render a path identifier useless after the timeout period, e.g., link identifiers may not be constant, instead they may change after some time related to the path identifier timeout. It is in our future work plans to explore solutions towards this direction.

Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This project was funded by the Deanship of Scientific Research (DSR) at King Abdulaziz University, Jeddah, under grant no. G-235-611-1440. The author, therefore, acknowledge with thanks DSR for technical and financial support.

References

[1] Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), RFC, IETF, 2014.
[2] W. Xia, Y. Wen, C.H. Foh, D. Niyato, H. Xie, A survey on software-defined networking, IEEE Commun. Surv. Tutor. 17 (1) (2015) 27–51, doi:10.1109/COMST.2014.2330903.
[3] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: a complete survey, J. Syst. Archit. 98 (2019) 289–330.
[4] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using openflow: a survey, IEEE Commun. Surv. Tutor. 16 (1) (2014) 493–512, doi:10.1109/SURV.2013.081313.00105.
[5] N. Bizanis, F.A. Kuipers, Sdn and virtualization solutions for the internet of things: a survey, IEEE Access 4 (2016) 5591–5606.
[6] S. Bera, S. Misra, A.V. Vasilakos, Software-defined networking for internet of things: asurvey, IEEE Internet Things J. 4 (6) (2017) 1994–2008.
[7] M. Wazid, A.K. Das, R. Hussain, G. Succi, J.J. Rodrigues, Authentication in cloud–driven IoT-based big data environment: survey and outlook, J. Syst. Archit. 97 (2019) 185–196.
[8] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, G. Gu, Towards SDN-Defined Programmable BYOD (Bring Your Own Device) security, 23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21–24, 2016, 2016.
[9] J. Sonchack, J.M. Smith, A.J. Aviv, E. Keller, Enabling practical software-defined networking security applications with OFX, 23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21–24, 2016, 2016.
[10] A. Voellmy, H. Kim, N. Feamster, Procera: A language for high-level reactive network control, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48.
[11] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, FRESCO: modular composable security services for software-defined networks, Internet Society NDSS., 2013.
[12] A. Lara, B. Ramamurthy, Opensec: policy-based security using software-defined networking, IEEE Trans. Netw. Serv. Manage. 13 (1) (2016) 30–42.
[13] S. Rivera, S. Lagraa, C. Nita-Rotaru, S. Becker, R. State, Ros-defender: Sdn-based security policy enforcement for robotic applications, in: 2019 IEEE Security and Privacy Workshops (SPW), 2019, pp. 114–119.
[14] K. Papachristou, T. Theodorou, S. Papadopoulos, A. Protogerou, A. Drosou, D. Tzovaras, Runtime and routing security policy verification for enhanced quality of service of IoT networks, in: 2019 Global IoT Summit (GIoTS), 2019, pp. 1–6.
[15] N. Fotiou, V.A. Siris, G. Xylomenos, G.C. Polyzos, K.V. Katsaros, G. Petropoulos, Edge-icn and its application to the internet of things, in: 2017 IFIP Networking Conference (IFIP Networking) and Workshops, 2017, pp. 1–6, doi:10.23919/IFIPNetworking.2017.8264880.
[16] N. Fotiou, D. Mendrinos, G.C. Polyzos, Edge-assisted traffic engineering and applications in the IoT, in: Proceedings of the 2018 Workshop on Mobile Edge Communications, in: MECOMM'18, ACM, New York, NY, USA, 2018, pp. 37–42, doi:10.1145/3229556.3229561.
[17] Z. Shelby, K. Koster, C. Bormann, P. van der Stok, C. Amsuess, CoRE Resource Directory, Internet-draft, IETF, 2019.
[18] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun ACM 13 (7) (1970) 422–426.
[19] M.J. Reed, M. Al-Naday, N. Thomos, D. Trossen, G. Petropoulos, S. Spirou, Stateless multicast switching in software defined networks, in: 2016 IEEE International Conference on Communications (ICC), 2016, pp. 1–7, doi:10.1109/ICC.2016.7511036.
[20] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, The design and implementation of open vswitch, in: 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), USENIX Association, Oakland, CA, 2015, pp. 117–130.
[21] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, NOX: Towards an operating system for networks, SIGCOMM Computer Communications Review 38 (3) (2008) 105–110.
[22] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, in: Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 19:1–19:6.
[23] V.A. Siris, N. Fotiou, A. Mertzianis, G.C. Polyzos, Smart application-aware IoT data collection, J. Reliab. Intell. Environ. 5 (1) (2019) 17–28.

Bander A Alzahrani is an associate professor at King Abdulaziz University, Saudi Arabia. He completed his M.Sc. in Computer Security (2010), and his Ph.D. in Computer Science (2015), both from University of Essex , United Kingdom. His research interests include Wireless sensor networks, Information centric networks, Bloom filter data structure and its applications, secure content routing, authentication protocols in IoT. Bander has published more than 37 research papers in International Journals and conferences.

NIKOS FOTIOU received the Dipl. in information and communication systems engineering from the University of the Aegean, Samos, Greece in 2005, the M.Sc. degree in Internetworking from the Royal Institute of Technology (KTH), Stockholm, Sweden in 2007, and the Ph.D. degree in computer science, for the Athens University of Economics and Business (AUEB), Athens, Greece, in 2014. Since 2014, he has been a Researcher in the Mobile Multimedia Laboratory at AUEB. His research interests include future Internet architectures, security and privacy, IoT systems, and applications of the distributed ledgers technology.