Which IoT Protocol?

Comparing standardized approaches over a common M2M application

Konstantinos Fysarakis, Ioannis Askoxylakis
Institute of Computer Science
Foundation for Research and Technology - Hellas (FORTH)
Heraklion, Crete, Greece
kfysarakis@ics.forth.gr, asko@ics.forth.gr

Othonas Soultatos, Ioannis Papaefstathiou
Dept. of Electronic & Computer Engineering
Technical University of Crete
Chania, Crete, Greece
othonass@gmail.com, ypg@mhl.tuc.gr

Charalampos Manifavas
Dept. of Electrical Engineering and Computing Sciences
Rochester Institute of Technology
Dubai, UAE
cxmcad@rit.edu

Vasilios Katos
Dept. of Computing
Bournemouth University
Poole, UK
vkatos@bournemouth.ac.uk

***Abstract*—Computing devices already permeate working and living environments, while researchers and engineers aim to exploit the potential of pervasive systems in order to introduce new types of services and address inveterate and emerging problems. This process will lead us eventually to the era of urban computing and the Internet of Things (IoT). However, the long-promised improvements require overcoming some significant obstacles introduced by these technological advancements. One such obstacle is the lack of interoperable solutions, to facilitate the use, monitoring and management of the plethora of devices and their services. While seamless machine-to-machine (M2M) and human-to-machine (H2M) interactions are a necessity for secure and truly ubiquitous computing, the current status quo is that of a segregated and incompatible assortment of devices. The resource-constraints of the platforms integrated into smart environments, and their heterogeneity in hardware, network and overlaying technologies, only exacerbate these interoperability issues. Motivated by the above, this paper identifies three promising, standardized protocols, each following a different approach in addressing the above concerns. We evaluate the selected protocols in the context of designing and implementing an application requiring various M2M interactions, namely a policy-based access control framework for IoT devices. Thus, three variants of the application are developed, considering each protocol's intrinsic characteristics and features. Finally, the developed applications are evaluated on a common testbed of embedded devices, allowing us to extract useful conclusions concerning the protocols' performance, their intricacies and their applicability in similar applications.**

Keywords— Internet of things; Ubiquitous computing; M2M; Policy-based access control; Authorization; DPWS; CoAP; MQTT; XACML

I. Introduction

Advances in computing and communication technologies have enabled a new reality where interconnected computing systems, in various forms, are constantly gaining popularity, permeating our environments and aiming to enhance all aspects of our everyday lives. The IP-based connectivity of devices, systems and services, which goes beyond the traditional human-to-machine (H2M) and machine-to-machine (M2M) interactions, is labeled the Internet of Things (IoT). Ubiquitous computing devices, featuring sensors and actuators, are already deployed in a variety of domains (residential/home automation, industrial systems, military, e-textiles, healthcare and automobiles, among others). Nevertheless, while existing networking and security mechanisms are updated and adapted to handle the vast population of IoT devices, higher level, seamless M2M and H2M interactions, are another important requirement in order to effectively monitor and manage the infrastructure, allowing the use of its full potential. End-users typically do not possess the skills to configure and setup the devices that may be found in smart environments; in large-scale deployments, individually setting up devices is not even feasible. From the perspective of implementers, there is a need for rapid development and deployment, while simultaneously tackling issues of scaling and inherent limitations in terms of resources (CPU, memory, power etc.). However, at its current state, the ubiquitous computing landscape is segregated, consisting of numerous proprietary solutions, which are typically incompatible with each other. This makes setting up, managing and securing a smart device ecosystem, significantly challenging.

Various "IoT protocols" aim to address these issues, while standardization initiatives try to guarantee interoperability, through the wide and structured deployment of the proposed mechanisms. Motivated by the above issues and associated efforts, this paper identifies three prominent approaches to providing seamless and lightweight IoT interactions, highlighting a representative, standardized protocol for each of these approaches. Moreover, an application featuring M2M interactions is selected, designing and implementing the required entities and interactions, while adapting them to the intricacies of each protocol. Finally, the developed solutions

are deployed and evaluated on a common testbed, allowing valuable conclusion to be extracted.

This paper is organized as follows: Section II presents the protocols and compares their features; Section III details the application that was used to assess the protocols, along with the intricacies of each protocol-specific implementation; Section IV presents the evaluation results; and Section V closes the paper with some concluding remarks and pointers to future work.

II. The Protocols

Surveying the academic literature and the web, reveals a plethora of protocols aiming to unify IoT devices and applications; some are still in their infancy, some are openly available, some are proprietary, and there are also significant efforts to standardize some protocols. Nevertheless, three main approaches seem to have gained the most traction at this stage, both in terms of research efforts, and in terms of actual applications being already provided to end users. These approaches are detailed in the subsections below.

A. *Service-oriented Approach – The Devices Profile for Web Services Protocol*

Service Oriented Architectures (SOAs) provide an attractive option for IoT node interactions, as web services allow stakeholders to focus on the services themselves, rather than the underlying hardware and network technologies. When examining SOAs, the Devices Profile for Web Services (DPWS, [1]) stands out. It was introduced in 2004 by a consortium led by Microsoft and is now an OASIS open standard (at version 1.1 since July 2009). DPWS was originally conceived and introduced as a successor to UPnP, but nowadays is actively pushed by industry stakeholders as the solution of choice for large-scale enterprise (e.g. industrial) deployments [2]. Like UPnP, DPWS is natively integrated into the various versions of the Windows operating system. The specification defines a minimal set of implementation constraints to enable secure Web Service messaging, including discovery, description, synchronous (via operation invocations) and asynchronous (via subscription and event-driven changes) interactions on resource-constrained devices. The profile's architecture includes hosting and hosted services. A single hosting service is associated with each device while the same device may accommodate various hosted services. The latter represent the device's various functional elements and rely on the hosting service for discovery. DPWS enables the adoption of a SOA approach on embedded and sensor devices with limited resources, allowing system owners to leverage the SOA benefits across heterogeneous systems that may be found in smart environments. The use and benefits of DPWS have been studied extensively in the context of various applications areas, which, other than the ones already mentioned, include automotive and railway systems [3], industrial automation [4], eHealth [5], smart cities [6] and smart homes [7].

B. *Resource-oriented Approach – The Constrained Application Protocol*

Another interesting approach for IoT interactions is the use of protocols following the Representational State Transfer (REST) architecture, which currently dominates the World Wide Web. RESTful implementations typically use the Hypertext Transfer Protocol (HTTP), but the latter is not appropriate for IoT applications, when considering the resource, bandwidth and energy restrictions of the target devices. Thus, the Internet Engineering Task Force (IETF) Constrained RESTful environments (CoRE) Working Group presented the Constrained Application Protocol (CoAP, [8]), now an IETF standard. CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things, aiming to maintain compatibility with the existing Internet infrastructure, through simple proxies. The protocol is often referred to as "the HTTP for the Internet of Things". It follows a request/response model, where a client may interact with the server using a subset of the HTTP methods, namely using GET, PUT, POST and DELETE on the server's resources. The protocol features two layers: the "Transaction layer" responsible for single message exchange between end points and the "Request/Response layer" which is responsible for request/response transmission and resource management; thus providing reliability mechanisms and basic congestion control. Moreover, basic publish/subscribe interactions are also supported, as, by extending the HTTP GET method, a client can "observe" a specific resource. There is significant research interest in CoAP, with numerous efforts to leverage its extremely lightweight interactions in domains such as smart homes [9], mobile IoT deployments [10], cloud services [11], healthcare [12], smart cities [13] and industrial WSNs [14].

C. *Message-oriented Approach – The MQ Telemetry Transport Protocol*

Message-oriented protocols typically focus on providing asynchronous data transfers between distributed devices. Their focus is on reliable messaging, including message buffers and Quality of Service (QoS) facilities, controlled by centralized entities. The MQ Telemetry Transport (MQTT, [15]) is one such message-oriented protocol, introduced by IBM in 1999 and recently standardized by OASIS, as the IoT developments brought it back into the limelight. It is also standardized as ISO/IEC 20922 [1]. MQTT was designed as an extremely lightweight publish/subscribe messaging transport, for small sensors and mobile devices, optimized for high-latency or unreliable networks. An MQTT Broker is responsible for handling and organizing all communications between the various devices. Messages are published with specific "topics", and each client can subscribe to various topics (though the Broker may require username/password authentication before allowing subscription). Topics are organized in a hierarchical manner, like the folder structure in a file system; e.g. "home/kitchen/oven/temperature" could be a topic where a device can subscribe to get updates on the oven's temperature. When a client publishes a message, the Broker then relays this message to all clients subscribed to the message's topic. Thus,

[1] http://www.iso.org/iso/catalogue_detail.htm?csnumber=69466

all interactions are asynchronous and clients only communicate directly with the Broker. As with the previous protocols, researchers have already studied MQTT in a variety of domains, including eHealth applications [16][17], WSNs and smart grid [18], smart homes [19] and also mobile IoT contexts [20], among others.

D. Comparison & Related Work

DPWS, CoAP and MQTT share some important characteristics which make them good candidates for IoT applications, and which motivated us to consider them for this comparison. More specifically, all three protocols: are open standards with significant traction in the research community and the industry; are designed with constrained environments in mind; can offer seamless M2M interactions; run on IP; and have a range of implementations readily available to developers and researchers alike. Nevertheless, as each follows a different approach to provide entity interactions, the protocols are, in many aspects, very different from each other.

While CoAP messages are transported over UDP, MQTT relies on TCP, and DPWS uses a combination of both (TCP for the bulk of the device interactions, and UDP for device discovery and other auxiliary functions); with each protocol inheriting different characteristics from the underlying transport mechanisms. This also affects the available security mechanisms, with DPWS and MQTT deployments supporting the use of TLS, and CoAP applications supporting DTLS. In the case of DPWS, the mechanisms detailed in the WS-Security specification are also applicable, as with any other Web Services deployment. MQTT is suited, by design, for publish/subscribe interactions, CoAP also has support for observing resources, partly covering such functionality, but it is better suited for synchronous interactions, instead of event-based ones. DPWS is more flexible in this regard, as the WS-Eventing specification enables a feature-rich publish/subscribe functionality, including interactions that are triggered at pre-defined intervals and/or when a specific event takes place. Moreover, QoS is an important aspect in MQTT, with the protocol supporting three different modes of message delivery ("Fire and forget", "Delivered at least once" and "Delivered exactly once"), whereas CoAP only offers a rudimentary choice between "Confirmable" and "Non-confirmable" messages. The former have to be acknowledge by the received with an ACK packet, in applications where it is necessary to cater for UDP's unreliable transport. DPWS has no such features built-in, relying solely on TCP's delivery mechanisms. Various extensions enhance the reliability and QoS features of Web Services (e.g. [21]), but these have not been integrated into DPWS yet.

A detailed theoretical comparison of the protocols is beyond the scope of this work, which aims to highlight differences that may not necessarily stand out on paper, but become evident when trying to use the protocols in an actual implementation. This is an aspect missing from the current literature, which mostly focuses on theoretical comparisons of various protocols or detailed comparisons focusing on specific protocol aspects such as packet loss. A high level survey of IoT protocols can be found in [22], though DPWS is not included in the comparison. Authors in [23], have compared DPWS to OPC-UA, a WS-based version of protocols typically used in the industrial domain. A hands-on comparison of CoAP and MQTT, in the context of a smartphone-based application, can be found in [24]. Authors in [25] have also compared the two protocols, using a common middleware which allowed them to assess the protocols' behavior in changing network conditions (as in scenarios with high packet loss). Another lab-based comparison of CoAP and MQTT, along with OPC-UA, in the context of communications over cellular networks, can be found in [26]. Finally, there is a significant body of work in the literature aiming to combine various IoT protocols. A custom Broker that support both CoAP and MQTT, bridging the two protocols, is presented in [27]. Researchers have also tried to bridge DPWS with RESTful architectures [28] and CoAP in specific [29]. A direct comparison of DPWS, CoAP and MQTT is missing altogether from the literature, to the best of our knowledge.

III. The Application - Access Control Framework

The expanded attack surface that results from the integration of the numerous smart devices around us with the Internet, needs new or adapted mechanisms to mitigate these new threats. The Open Web Application Security Project (OWASP) organization includes "Insufficient Authentication/Authorization" in the second place of its list of top ten security problems identified on IoT devices [30], preceded only by the use of "Insecure Web Interfaces". Such negligence in terms of proper authorization is bound to inhibit any efforts made towards using these pervasive devices to handle our personal sensitive data

Aiming to address the above concerns, we have presented a policy-based access control framework for smart ubiquitous devices [6][31]. Based on the OASIS-standardized eXtensible Access Control Markup Language (XACML, [32]), the proposed solution provides the means to control access to the resources of IoT nodes, based on policy constraints centrally managed by the system owner. While XACML defines the structure and content of access requests & responses, it provides no information on the mechanisms for transferring these messages. Typical XACML deployments require the setup of complex infrastructures to enable entities' interaction and policy retrieval (e.g. via LDAP); an approach not suitable in the context of IoT applications and the average user. To overcome this limitation, the policy-based framework could leverage the benefits of IoT protocols, gaining seamless Machine-to-Machine (M2M) discovery and interactions, and allowing the deployment of the framework's entities to any platform, anywhere on the network, with minimal involvement on behalf of the users. Thus, we decided to assess the applicability and performance of the selected protocols in the context of designing and implementing said authorization framework for IoT devices. To achieve this, three different implementations of the framework and its entities were developed, using each of the investigated protocols.

The framework's key entities that had to be implemented, were the following: a **Policy Enforcement Point (PEP)** which makes decision requests and enforces authorization decisions; a **Policy Decision Point (PDP)** which evaluates requests against applicable policies and renders an authorization decision; a

Policy Administration Point (PAP) which creates/manages policies or policy sets; and a **Policy Information Point (PIP)** acts as a source of attribute values.

As an example of the framework's flow of interactions, consider the case of a person who owns a smart thermostat. The thermostat is a device that hosts a service which supports multiple operations such as setting the target temperature, selecting operation mode, enabling power save, getting the current status or even events such as notifications when the temperature in the room changes, or the target temperature has been reached. As soon as the available device and its service are discovered, a guest in the house can request access to the node that is of particular interest, e.g. in order to extract the latest values from the temperature sensor attached to it. The guest's request is intercepted by the node's PEP module which then forwards the request to the PDP, the latter running on the house owner's trusted device. The PDP has to consider all applicable policies, enriched by any relevant information, from the PIP/PAP. Once all the required information has been collected, the PDP issues a decision which is sent back to the node's PEP. Based on that decision the PEP may or may not allow the guest to access said nodes data of interest.

A. Implementation

Differentiations between the variants were kept to a minimum, where possible, to avoid masking the performance differences during the evaluation phase. Thus, Sun's XACML engine was the basis for all access control implementations; a choice that remains popular among developers and is actually the basis of various current open source and commercial offerings. Moreover, while there is a variety of APIs, based on different programming languages, for all three protocols, the devices were developed using solely open source, Java-based APIs. These were typically the most mature and feature-rich of the available options for each protocol, and, moreover, offer platform-agnostic implementations that can be easily be moved between hardware platforms; an important benefit in the context of heterogeneous IoT platforms. Nevertheless, the exact M2M interactions had to be adapted to the intricacies of each protocol, as will be presented in the subsections below. The description of each implementation will only cover the interactions taking place when a client invokes a resource; the setup process (e.g. device discovery, subscription to events/topics etc.) is not included, for simplicity.

1) DPWS-based Framework

A survey on APIs for DPWS development reveals a plethora of available solutions with diverse characteristics. Still, when focusing on key features such as code portability, support for IPv6 (necessary for IoT applications) and active development and support of the tools, the valid options are actually fewer, with WS4D-JMEDS[2] standing out as the most attractive choice, being constantly updated and improved. The flexibility and feature-rich functionality of the protocol meant there were no significant restrictions when designing the framework and its entities' interactions. Therefore, exploiting the discovery and publish/subscribe features of DPWS, the PDP subscribes to the "SAReq" eventing operation of any

[2] http://ws4d.org/jmeds

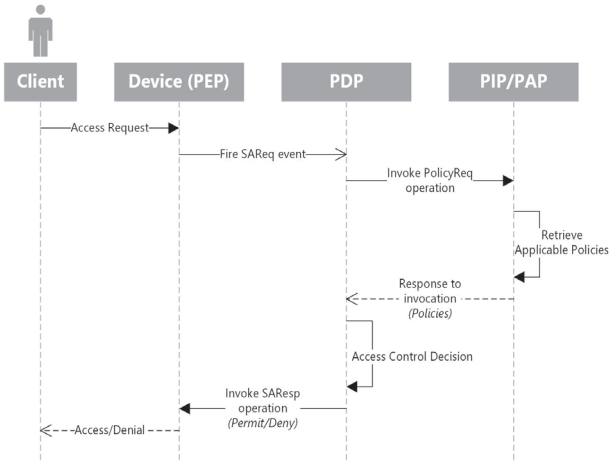

Fig. 1. Sequence Diagram of DPWS Variant

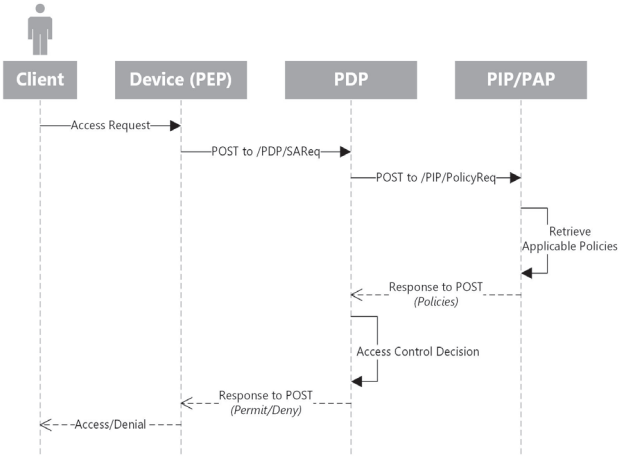

Fig. 2. Sequence Diagram of CoAP Variant

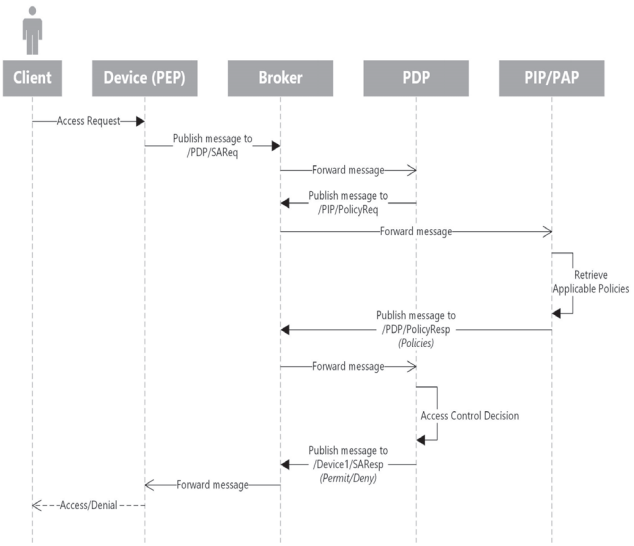

Fig. 3. Sequence Diagram of MQTT Variant

PEP-equipped device that joins the network. This event is fired as soon as a client tries to access a protected resource on the device, relaying the necessary information to the PDP. The PDP then invokes the "PolicyReq" operation on the PIP/PAP, getting the applicable policies in reply. Having all the required information, the PDP evaluates the policies and issues a decision, and then invokes the "SAResp" operation on the PEP to inform it of its decision. Depending on said decision, the device either allows or denies access to the client. This sequence of events appears in Fig. 1.

2) CoAP-based Framework

There are numerous CoAP libraries available to developers, many being platform-specific, i.e. focusing on specific devices. Our implementation was based on the most popular Java-based option, the open-source Californium[3] API, provided by the Eclipse Foundation. There were two obstacles when designing the CoAP-based framework. Firstly, due to the nature of its transport protocols, there are limitations with regard to the reliability of the "observe" mechanism, so we cannot use it inform the PDP of requests coming to the PEP, as in the case of the subscription in the DPWS implementation. As stated in its RFC [8], the mechanism was designed with the "principle of eventual consistency", and a client cannot rely on observing every single state that a resource might go through. This is not tolerable in our application, as we want to be sure all requests reach the PDP for evaluation. Secondly, CoAP's discovery features are restricted (esp. since DTLS does not support multicast UDP messages [33]), prohibiting a consistent automated discovery, as in DPWS above. Thus, for the CoAP implementation of the authorization framework, we focused on using simpler POST-based interactions between the entities, with the access control –related operations being exposed as resources on the devices' endpoints (e.g. "PDP/SAReq"), as depicted in Fig. 2.

3) MQTT-based Framework

For the MQTT implementation we used Paho[4], the Eclipse Foundation's API for MQTT clients, which provides open-source client implementations of open and standard messaging protocols. It has a rich documentation and examples for many languages that the client is implemented. As a Broker, we used Mosquito [5] which is an Open Source MQTT v3.1/v3.1.1 Broker. As in MQTT everything has to go through the Broker, the implementation had to be modified accordingly: all interactions were defined as message posting to the corresponding topics, with the Broker being responsible to deliver these messages to the intended recipient. Each of the framework's operations features its own hierarchical topic (e.g. "/PDP/SAReq"), to ensure that all the entities get the messages intended for them. The sequence of interactions appears in Fig. 3.

IV. Evaluation Results

A common test bed, featuring relatively resource-constrained embedded devices, was used to compare the performance of the three implementation variants of our application (i.e. the access control framework). In more detail, the PEP-equipped target device (i.e. the device providing the actual service to be accessed) was deployed on a Beaglebone, a low-cost credit-card-sized device, running a compact Linux-based operating system. It uses an ARM Cortex-A8 single core CPU running at 720MHz (throttled at 500MHz during testing) with 256MB DDR2 RAM. The test-bed for the Service Orchestrator was a similar, but slightly more powerful and versatile, Beagleboard-xM embedded platform, featuring an 1GHz ARM Cortex-A8 processor (throttled to run at 600MHz during testing) and 512MB DDR2 RAM, also running a minimal Linux-based operating system. The access control infrastructure entities, i.e. the PDP and PIP/PAP, were deployed on a desktop system (Core i5 CPU at 3.3GHz, 8GB DDR3 RAM), as these are expected to run on more resource-rich devices (e.g. the main system used to control and configure our smart home). This system also hosted the Broker needed in the case of the MQTT setup. Finally, the client application was run on a laptop computer, which was programmed to automatically invoke the resources of the device. The client-side response time of 100 consecutive requests to the protected devices was monitored for benchmarking purposes, along with various other parameters on the embedded devices (i.e. the Beaglebone featuring the PEP and the Beagleboard xM assigned with the PDP functionality. Finally, all systems were interconnected via wired Ethernet to minimize the network's impact on the reported performance figures. The setup for all three variants of the framework appears in Fig. 4.

One of the most important aspects compared during benchmarking was the client-side response time, as this refers to the delay a user would experience in each case when trying to access the protected resource (e.g. the temperature of a smart thermostat). The recorded times, averaged over 100 requests, were **130.6ms** for DPWS, **100.3ms** for CoAP and **97.84ms** for MQTT. As is evident from these results, the CoAP- and MQTT- based variants of the framework performed equivalently in this regard, with their DPWS counterpart needing about 30% more time to respond. Further analyzing the response times reveals that in all cases, the bulk (i.e. 60-70%) of the delay can be attributed to the Client-Device (i.e. PEP-PDP) communication, which also includes the time

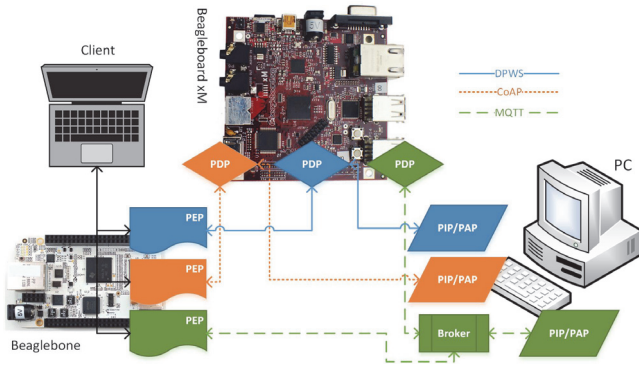

Fig. 4. Evaluation Setup

[3] http://www.eclipse.org/californium

[4] https://eclipse.org/paho

[5] http://mosquitto.org

needed for the PDP to issue a decision on the evaluated access request. The latter gives a bottom barrier to the delay a client can experience, and is irrelevant to the protocol used, as it is inherent to the specific application (more specifically, the policy evaluation mechanism of XACML), and beyond the scope of this work to analyze and improve.

Moving to the embedded devices hosting the PEP and PDP, the CPU load and memory footprint of each application was monitored during the benchmarks. Results from the backend desktop system are omitted, as the presence of the PIP/PAP applications (and the Broker, in the case of the MQTT deployment) had, in the context of its significant computing capabilities, no notable impact on resource utilization. The average CPU load recorded for each protocol and each device appear in Fig. 5, where it is evident that the MQTT and CoAP applications had a bigger impact on the devices' resources during testing. As these variants of the framework have a lower response time compared to DPWS, a higher load is imposed to the devices under test, since they handle more requests (and the associated interactions) per unit of time. The MQTT framework is even more load-intensive than its CoAP counterpart, as it is slightly faster but also involves more complex interactions between the devices. Moreover, the average memory utilization of the applications recorded during tests appears in Fig. 6. The DPWS applications (developed with WS4D-JMEDS) had a bigger memory footprint than their CoAP and MQTT counterparts (developed with Californium and Paho, respectively), with the MQTT variants having a slight advantage over the CoAP ones.

Lastly, there is, as expected by studying the corresponding

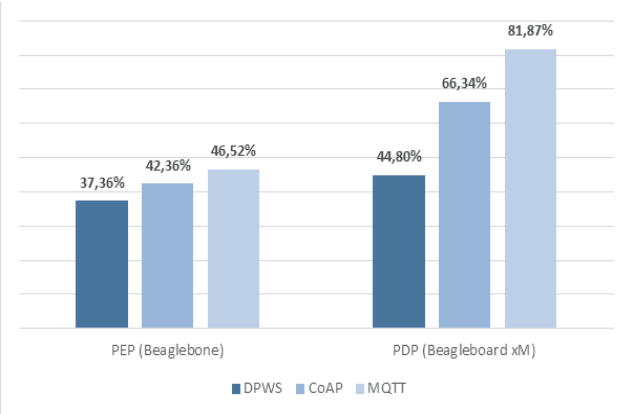

Fig. 5. Average CPU Load (%) on Embedded Devices

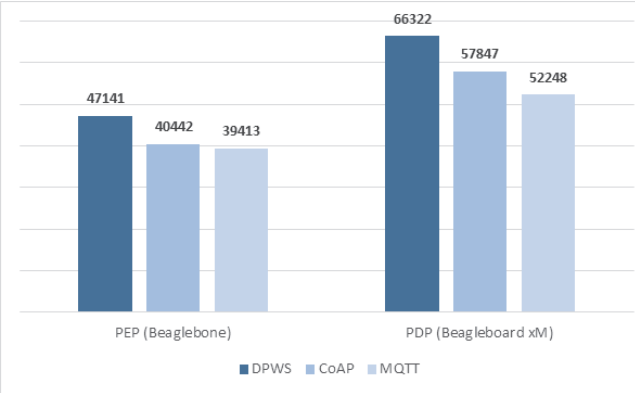

Fig. 6. Average Memory Utilization (kB) on Embedded Devices

specifications, a significant difference in terms of packet size between DPWS and the other two protocols. To assess the exact variance in the context of our application, we focused on a specific interaction with a minimum payload: a "Permit" message that the PDP sends to the PEP, after positively evaluating a request, based on the applicable policies. By capturing the corresponding network packet for each of the protocols, we recorded a total packet size (i.e. including all headers, as it appears "on the wire") of **849 bytes** in the case of DPWS, **58 bytes** for CoAP and **80 bytes** for MQTT. Also note that, in the MQTT application, there are twice as many messages sent compared to the other applications (8, instead of 4), as all interactions have to go through the Broker, who essentially resends the message to the subscribed parties.

V. Discussion & Future Work

Our goal in this work was not to just to assess the performance of each solution – an aspect that can be gathered from studying the specifications and the related work. Instead, our approach aimed to also highlight how the intricacies of each protocol (different protocol design approaches, different supported features etc.) dictate different design decisions (e.g. in terms of architectures, entity interactions, and device deployment.

Thus, performance-aside, DPWS was the benchmark in terms of the ease in designing the framework. Its robust and flexible discovery, subscription and eventing mechanisms meant that the entities and their interactions could be designed in an intuitive manner. This is also true for the end application, as it is the most hassle-free variant from the end users' perspective; minimal setup is required and the entities discover each other and interact seamlessly, no matter where they are deployed on the network. CoAP was intuitive to work with, especially considering that as most developers nowadays have experience with RESTful applications. Still, careful study of the protocol and its limitations (theoretical and/or in terms what is supported in the existing APIs) is needed, as it is not as mature as the other two protocols considered. Lastly, MQTT's lack of synchronous interaction support, meant that we had to follow a not so "elegant" approach in designing the entities' interactions, with too many interactions happening in order to bypass the limitation of only supporting asynchronous interactions that have to be routed through a Broker. In summary, the protocol choice necessitates careful consideration of the target application, as no ideal protocol exists; some protocols have more features and are more mature than the alternatives, while others are more lightweight, some are ideally suited to aggregating data from a variety of sensors, while others are better suited for end-user (e.g. consumer) applications, etc.

Nevertheless, a complex deployment can use more than one protocol. Moreover, the development of a custom protocol could be investigated. An approach to be considered, and one that does not fully sacrifice interoperability with existing solutions, could be to combine one or more protocols, delegating to each one a task that it's more suited for. Thus, for example, CoAP could be used for the lightweight M2M interactions it can provide, MQTT for the cross-domain communications and the SOA-based approach of DPWS could

be used for M2H interactions (to leverage the ubiquity of web service support in all human-operated devices) – an approach we intend to explore in future work.

ACKNOWLEDGMENT

Part of this work was funded by the European Union's Horizon 2020 research and innovation programme Virtuwind, Grant Agreement No: 671648.

REFERENCES

[1] D. Driscoll, A. Mensch, T. Nixon, and A. Regnier, "Devices profile for web services, version 1.1," OASIS, 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.pdf.

[2] T. Nixon, "UPnP Forum and DPWS Standardization Status," 2008. [Online]. Available: http://download.microsoft.com/download/f/0/5/f05a42ce-575b-4c60-82d6-208d3754b2d6/UPnP_DPWS_RS08.pptx.

[3] V. Venkatesh, V. Vaithayana, P. Raj, K. Gopalan, and R. Amirtharaj, "A Smart Train Using the DPWS-based Sensor Integration," Res. J. Inf. Technol., vol. 5, no. 3, pp. 352–362, Mar. 2013.

[4] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A Real-Time Service-Oriented Architecture for Industrial Automation," IEEE Trans. Ind. Informatics, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[5] S. Pöhlsen, S. Schlichting, M. Strähle, F. Franz, and C. Werner, "A DPWS-Based Architecture for Medical Device Interoperability," in World Congress on Medical Physics and Biomedical Engineering, September 7 - 12, 2009, Munich, Germany SE - 22, vol. 25/5, O. Dössel and W. Schlegel, Eds. Springer Berlin Heidelberg, 2009, pp. 82–85.

[6] K. Fysarakis, I. Papaefstathiou, C. Manifavas, K. Rantos, and O. Sultatos, "Policy-based access control for DPWS-enabled ubiquitous devices," in Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), 2014, pp. 1–8.

[7] R. R. Igorevich, P. Park, J. Choi, and D. Min, "iVision based Context-Aware Smart Home system," in The 1st IEEE Global Conference on Consumer Electronics 2012, 2012, pp. 542–546.

[8] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)", 2014. https://tools.ietf.org/html/rfc7252.

[9] O. Bergmann, K. T. Hillmann and S. Gerdes, "A CoAP-gateway for smart homes," Computing, Networking and Communications (ICNC), 2012 International Conference on, Maui, HI, 2012, pp. 446-450.

[10] S. M. Chun and J. T. Park, "Mobile CoAP for IoT mobility management," Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE, Las Vegas, NV, 2015, pp. 283-289.

[11] A. Betzler, C. Gomez, I. Demirkol and M. Kovatsch, "Congestion control for CoAP cloud services," Emerging Technology and Factory Automation (ETFA), 2014 IEEE, Barcelona, 2014, pp. 1-6.

[12] H. A. Khattak, M. Ruta and E. Di Sciascio, "CoAP-based healthcare sensor networks: A survey," Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on, Islamabad, 2014, pp. 499-503.

[13] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," in IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22-32, Feb. 2014.

[14] C. P. Kruger and G. P. Hancke, "Implementing the Internet of Things vision in industrial wireless sensor networks," Industrial Informatics (INDIN), 2014 12th IEEE International Conference on, Porto Alegre, 2014, pp. 627-632.

[15] A. Banks, R. Gupta, OASIS Message Queuing Telemetry Transport (MQTT), version 3.1.1, OASIS. (2014) 1-81. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf.

[16] D. Barata, G. Louzada, A. Carreiro, and A. Damasceno, "System of acquisition, transmission, storage and visualization of Pulse Oximeter and ECG data using Android and MQTT". Procedia Technology, 9, 1265-1272, 2013.

[17] Y. F. Gomes, D. F. S. Santos, H. O. Almeida and A. Perkusich, "Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things," Consumer Electronics (ICCE), 2015 IEEE International Conference on, Las Vegas, NV, 2015, pp. 200-201.

[18] P. Papageorgas, D. Piromalis, T. Iliopoulou, K. Agavanakis, M. Barbarosou, K. Prekas, K. Antonakoglou, "Wireless Sensor Networking Architecture of Polytropon: An Open Source Scalable Platform for the Smart Grid", Energy Procedia, Volume 50, Pages 270-276, 2014.

[19] Seong-Min Kim, Hoan-Suk Choi and Woo-Seop Rhee, "IoT home gateway for auto-configuration and management of MQTT devices," Wireless Sensors (ICWiSe), 2015 IEEE Conference on, Melaka, 2015, pp. 12-17.

[20] J. E. Luzuriaga, J. C. Cano, C. Calafate, P. Manzoni, M. Perez and P. Boronat, "Handling mobility in IoT applications using the MQTT protocol," Internet Technologies and Applications (ITA), 2015, Wrexham, 2015, pp. 245-250.

[21] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, and U. Yalcinalp, "Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2," Oasis Standard, 2009. [Online]. Available: http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.pdf.

[22] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A Survey on Application Layer Protocols for the Internet of Things," *Trans. IoT Cloud Comput.*, vol. 3, no. 1, pp. 11–17, 2015.

[23] G. Cândido, F. Jammesy, J. B. De Oliveira, and A. W. Colomboz, "SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications," in *IEEE International Conference on Industrial Informatics (INDIN)*, 2010, pp. 598–603.

[24] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, 2013, pp. 1–6.

[25] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, 2014, pp. 1–6.

[26] L. Durkop, B. Czybik, and J. Jasperneite, "Performance evaluation of M2M protocols over cellular networks in a lab environment," in *2015 18th International Conference on Intelligence in Next Generation Networks*, 2015, pp. 70–75.

[27] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, 2012, pp. 36–41.

[28] S. Han, S. Park, G. M. Lee, and N. Crespi, "Extending the Device Profile for Web Services (DPWS) standard using a REST Proxy," *IEEE Internet Comput.*, pp. 1–1, 2014.

[29] G. Moritz, F. Golatowski, and D. Timmermann, "A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks," in 2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems, 2011, pp. 861–866

[30] OWASP, "Internet of Things Top Ten Project," Open Web Application Security Project (OWASP). [Online]. Available: https://www.owasp.org/index.php/OWASP_Internet_of_Things_Top_Ten_Project.

[31] Fysarakis, K., Konstantourakis, C., Rantos, K., Manifavas, C., & Papaefstathiou, I. (2015). WSACd-A Usable Access Control Framework for Smart Home Devices. In Information Security Theory and Practice (pp. 120-133). Springer International Publishing.

[32] B. Parducci, H. Lockhart, and E. Rissanen, "eXtensible Access Control Markup Language (XACML) Version 3.0," OASIS Standard, 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf.

[33] T. A. Alghamdi, A. Lasebae, and M. Aiash, "Security analysis of the constrained application protocol in the Internet of Things," in Second International Conference on Future Generation Communication Technologies (FGCT 2013), 2013, pp. 163–168