Bridging IoT Protocols with the Web of Things: A Path to Enhanced Interoperability

Zakaria Benomar, Marco Garofalo, Nikolaos Georgantas, Francesco Longo, Giovanni Merlino, Antonio Puliafito

Bridging IoT Protocols with the Web of Things: A Path to Enhanced Interoperability

Zakaria Benomar*, Marco Garofalo†, Nikolaos Georgantas*, Francesco Longo†‡
Giovanni Merlino†‡, Antonio Puliafito†‡
*National Institute for Research in Digital Science and Technology (INRIA), Paris, France
Emails: {zakaria.benomar, nikolaos.georgantas}@inria.fr
†Department of Engineering, University of Messina, Italy
Emails: marco.garofalo@studenti.unime.it
{flongo,gmerlino,apuliafito}@unime.it
‡ CINI: National Interuniversity Consortium for Informatics, Rome, Italy

***Abstract*—Nowadays, the Internet of Things (IoT) is reshaping numerous application domains. Amidst the different communication protocols currently available (MQTT, CoAP, AMQP, DPWS, etc.) and the abundance of management platforms, the IoT domain has stumbled into vertical silos of proprietary systems hindering interoperability. Within this intricate and fragmented ecosystem, the need for an intelligent and scalable architecture promoting interoperability becomes imperative to maximize the potential of IoT. This paper introduces a system designed to enhance the convergence of IoT protocols (e.g., MQTT, CoAP, AMQP, etc.) with the Web of Things (WoT) paradigm. Our proposed gateway-based solution integrates two systems: Stack4Things (S4T) and Data eXchange Mediator Synthesizer (DeXMS). The role of the latter is to adapt, at the gateway level, the IoT protocols and expose the resources/functionalities of the IoT devices as RESTful APIs using HTTP. Meanwhile, the former (i.e., S4T), leveraging its Dynamic DNS system, ensures that these RESTful resources (exposed at the gateway) are accessible over the Web using publicly routable Uniform Resource Locators (URLs) even when the gateway is deployed behind networking middleboxes (e.g., NATs and firewalls).**

***Index Terms*—IoT, Cloud, interop, Web of Things, DNS.**

I. Introduction

The world we live in is becoming increasingly digital under the widespread emergence of the Internet of Things (IoT). Our surrounding environments are progressively becoming smart, ranging from individual houses and offices to schools, factories, and hospitals. We are now living in the era of the Smart City where a whole city is managed by intelligent systems [1]. This immense growth in terms of the number and type of connected devices calls for new features to use this infrastructure efficiently.

Even though the IoT concept emerged several years ago, enacting IoT-based systems is still raising tremendous challenges for the supporting infrastructure, from networking to programming abstractions. Key challenges relate to [2]: **i)** scale, as we are dealing with systems that may have to coordinate millions of devices; **ii)** deep heterogeneity, since the IoT brings together sensor and actuator networks with cloud-based systems provided by different stakeholders relying upon different protocols and technologies for communication; **iii)** high dynamics, in relation to the unknown topology of the network and further the presence of mobile things, as well as uncertainty about the features of the networked things. The challenges that the IoT is raising in the development of computing systems, along with perspectives on how to address them, have been the focus of numerous papers over the last years [3] [4] [5].

Although the high potential of IoT, the diversity in terms of IoT devices, coupled with the variety of available communication protocols they employ (e.g., MQTT, CoAP, HTTP), poses significant challenges to achieving seamless interoperability. This heterogeneity makes the IoT field a fragmented ecosystem. In most cases, IoT deployments are still disconnected from each other as they use proprietary vertical technology stacks managed by centralized systems (e.g., cloud platforms). These setups typically lack interoperability, hindering communication between devices and systems from different providers. This is a problem leading to vendor lock-in issues, and as a result, users may experience the limitations of IoT [6]. To overcome this issue, providers necessitate tailored integrations for each vendor. However, this task is challenging to achieve due to the complexity of the ecosystem and the absence of a standardized protocol or dominant market player.

Among the plethora of IoT protocols, MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), and HTTP (HyperText Transfer Protocol) have emerged as prominent contenders, each offering unique strengths and capabilities suited for diverse IoT scenarios. While these protocols excel within their respective domains, the fragmentation caused by their disparate implementations poses significant interoperability challenges. A noteworthy method for tackling this challenge is the adoption of the multi-protocol proxy architecture [7]. This approach mitigates protocol disparities by employing gateways to adapt protocol interactions and data semantics.

One of the intriguing approaches to fostering interoperable IoT deployments, the Web of Things (WoT) paradigm stands out prominently. The WoT approach advocates for using Representational State Transfer (REST) with Web protocols (e.g., HTTP) as an appropriate software architectural style for exposing the functionalities of IoT devices, such as sensed

data. This approach enhances productivity by facilitating the integration of IoT with heterogeneous applications and systems, making IoT applications more accessible to conceptualize. Consequently, IoT devices transition from mere devices connected to the Internet to adept communicators capable of interacting with other devices and systems using well-established Web protocols (e.g., HTTP).

To deal with protocols' interoperability in IoT, this paper introduces a novel system to address interoperability in IoT environments, focusing on enhancing the alignment of IoT protocols with the WoT paradigm. In particular, we consider the case of bridging, through a smart gateway, resource-constrained IoT devices employing different protocols to the Web (i.e., to be exploitable over HTTP) irrespective of gateway localization and underlying network configurations (e.g., being deployed behind a NAT). To conceive our system, we used two platforms, Stack4Things (S4T) and Data Exchange Mediator Synthesizer (DeXMS). The role of the latter system lies in adapting IoT protocols (e.g., MQTT and CoAP) along with their resources and functionalities (such as sensed data), making them accessible as RESTful HTTP resources (using an HTTP server). To make the latter accessible over the Web, the former system (i.e., S4T) leverages its Dynamic Domain Name System (DDNS) to expose the HTTP resources through publicly routable Uniform Resource Locators (URLs).

The rest of the paper is structured as follows. Section II gives an overview of the current literature; Section III introduce the technologies used; Section IV details the system architecture and its functioning; Section V explains the tests conducted and the measurements obtained; and Section VI concludes the paper and gives hints about future works.

II. Related works

The IoT concept is rapidly expanding across various sectors, driving a surge in market sectors. With a multitude of commercial products available, competition between vendors often leads to compatibility issues among solutions [8]. In today's IoT landscape, numerous protocols are proposed, with their usage evolving over time to accommodate new applications and constraints inherent to IoT deployments [9]. To facilitate communication among IoT devices, frameworks enabling seamless device-to-device connectivity like IoTivity [10] (formerly known as AllJoyn) have emerged. Several applications built on top of these platforms have been proposed [11] [12]. Yet, these solutions operate under the assumption that IoT devices exclusively employ CoAP, which may not always be practical. Additionally, a major limitation of AllJoyn is its lack of scalability, particularly in supporting communication across devices belonging to different broadcast domains. Consequently, AllJoyn's utility is confined primarily to limited-scale deployments like smart homes. Another approach to address protocol interoperability involves implementing agents for protocol translation/adaptation, often employing dedicated gateways [7]. In one such approach [13], authors propose a multi-layer solution employing an intermediate layer to achieve protocol interoperability in smart home environments. However, this solution primarily focuses on MQTT and a specific cloud platform. Similarly, in [14], the focus lies in enabling communication between CoAP and HTTP.

To fully harness the potential of IoT and improve communication among devices and systems, the WoT paradigm emerges as a prominent option [15] [16]. By enabling all devices to communicate using Web protocols such as HTTP, the development of new services becomes easier. In this context, solutions have been proposed to bridge IoT protocols to align with Web protocols. In one such solution outlined in [17], using a centralized system, data is collected from a variety of IoT devices utilizing different protocols and then transmitted to services via HTTP POST requests. However, this solution lacks bidirectional interaction, preventing communication from the services/applications back to the devices, particularly for actuation purposes. Besides, the solution does not offer reachable identifiers (i.e., URIs) associated with the IoT devices. Similarly, in [18], the authors implemented the WoT paradigm using MQTT and CoAP while bridging them to HTTP. Yet, challenges such as data adaptation and the accessibility of the gateway from an external network (i.e., the management of device URIs) have not been addressed.

To achieve an efficient system aligned with the WoT perspective, an S4T-based WoT system have been introduced in [19]. The emphasis was primarily on ensuring the accessibility of Web services hosted on devices deployed behind NATs. This is achieved using a DDNS system built on OpenStack. In particular, the focus was on relatively powerful (MPU-powered) devices like Raspberry Pi. In this paper, we are expanding this approach by using gateways to integrate constrained IoT devices employing different protocols within the Web wisdom. This entails building upon our previous solution and integrating a gateway mechanism to tackle both, reachability concerns and protocols adaptation.

III. Background

In this section, we introduce the systems used to conceive our WoT platform. Besides, we provide an overview about the WoT architectures.

A. Stack4Things (S4T)

S4T [20] is a research effort aiming at integrating IoT and the cloud. The project focuses on extending the widely used open source middleware for the Infrastructure as a Service (IaaS) cloud computing model, that is, OpenStack, to fully support the management of an IoT infrastructure.

The architecture of the S4T middleware is split between the geographically distributed IoT nodes (or gateways) and a centralized cloud datacenter. On the cloud side (see red subsystem in Fig. 1), we have a service named *IoTronic* in charge of managing a set of services dealing with the distributed nodes (e.g., remote access using ssh, remote programmability). As hardware setup of the IoT nodes, we consider relatively powereful MPU-powered embedded systems (e.g., Arduino, Raspberry Pi) capable of hosting a minimal Linux distribution (e.g., OpenWRT). This software configuration makes those

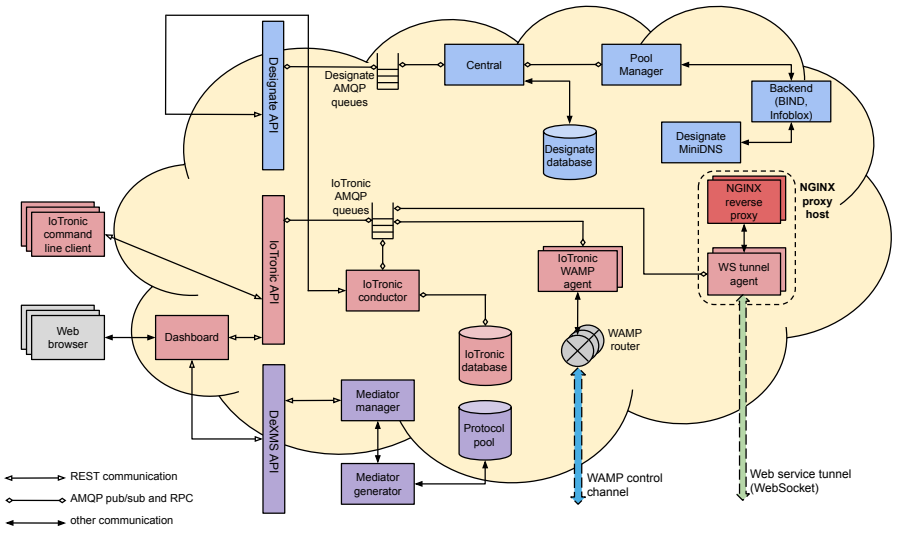


Fig. 1: IoTronic, Designate and DeXMS integration design.

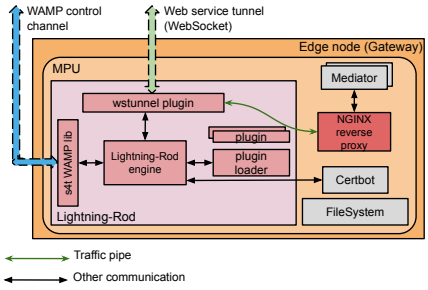


Fig. 2: The S4T device-side architecture.

nodes capable of hosting a set of programming runtimes (e.g., Python, NodeJS) that are required by the S4T device-side agent *Lightning-Rod* (LR) (see Fig. 2). This later componenent is responsible of keeping the nodes connected to the Cloud using a WebSocket-based protocol namely Web Application Messaging Protocol (WAMP[1]) (see blue dashed arrow in Figs. 1 and 2).

In our S4T view, we are intended to delegate as much functionalities as possible to dedicated subsystems from the OpenStack ecosystem. In this context, *IoTronic* has been integrated with a number of OpenStack services for different purposes (e.g., Keystone for users' authentication/authorization).

As one of the main utilities provided by S4T, the Web services system provides the capability of exposing, publicly, RESTful services (i.e., web services) hosted on distributed nodes deployed behind NATs and/or firewals. To achieve this, *IoTronic* has been seamlessly integrated with the DNS subsystem of OpenStack called Designate[2] (see Fig. 1). This integration aims at associating routabe URLs to the services hosted on the distributed nodes by employing a set of mechanisms dealing with the challenges induced by IoT environments (e.g., NATs). In particular, the system uses NGINX reverse proxies and Websocket tunnels to route traffic to nodes deployed at the network edge [21] (see Figs. 1 and 2).

B. Data eXchange Mediator Synthesizer (DeXMS)

The DeXMS [22] framework addresses the need to interconnect IoT devices employing various protocols at the middleware layer (e.g., MQTT [23], CoAP [24], WebSocket [25], HTTP, etc). DeXMS supports the synthesis of *mediators* that seamlessly interconnects IoT devices employing different interaction protocols at the middleware level. The architecture of the proposed mediators is inspired by the Enterprise Service Bus (ESB) paradigm. In this paradigm, a common intermediate bus protocol is used to facilitate the interconnection between applications employing diverse middleware protocols: instead of implementing all possible conversions between the protocols, a developer needs only to implement the conversion of each protocol to the common bus protocol, thus considerably reducing the development effort. This conversion is done by a component associated to each application and its middleware, called a *Binding Component*, as it binds the application to the service bus. Likewise, a mediator binds an IoT device (or a set of devices) to the common protocol of an IoT application. DeXMS framework enables to synthesize mediator for *direct mediation* in IoT environments supporting heterogeneous middleware protocols. As depicted in Fig. 3, a mediator is directly associated to both temperature sensors and an HTTP-based service to bridge the interoperability gap between MQTT and HTTP.

To facilitate the automated synthesis of mediators for interconnecting heterogeneous IoT devices, the DeXMS framework provides a generic interface description for IoT devices, called *Data eXchange Interface Description Language* (DeXIDL). A DeXIDL corresponds to an IoT device that employs any middleware protocol, it enables the definition of operations

[1]https://wamp-proto.org/

[2]https://docs.openstack.org/designate

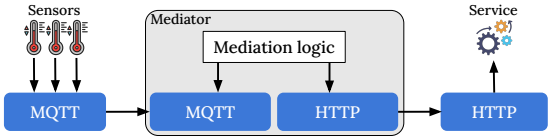


Fig. 3: The DeXMS mediation mechanism.

provided or required by the IoT device and its role (i.e. provider or consumer). Besides an operation's type, the names and data types of its parameters are also specified.

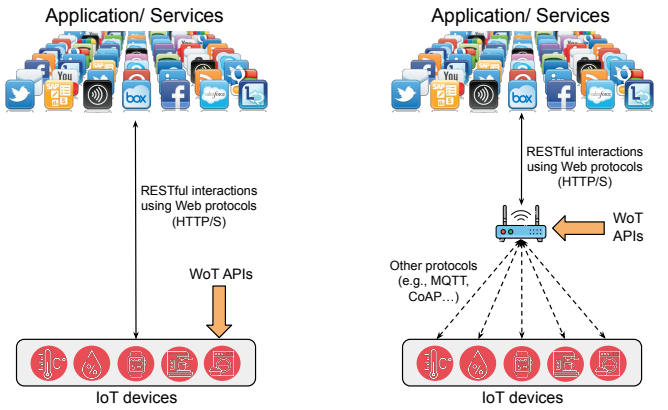
C. Integrating IoT devices into the Web

An overarching perspective of WoT architectures is illustrated in Fig.4. By harnessing the conventional client/server communication model, robust IP-enabled IoT devices can operate as HTTP/S RESTful entities, enabling their seamless integration into the web through direct API access (see Fig.4(a)). Alternatively, in scenarios where IoT devices face resource constraints and cannot support protocols such as HTTP/S, the WoT framework utilizes smart gateways to bridge these devices to the Web (see Fig. 4(b)). We delve into these two integration patterns further in the following:

1) **Direct Integration:** In this integration model, devices must possess sufficient capabilities to handle TCP/IP and HTTP/S. Additionally, embedding web servers is essential to enable smart devices to collaborate using Web standards. While this integration approach may seem straightforward, it presents challenges. Overcoming constraints imposed by IoT deployments, such as NATs and firewalls, requires intricate features. In this scenario, smart devices must remain accessible and expose their resources via public URLs, even when deployed behind networking middleboxes. This integration pattern has been the focus of previous works [21] [19] .
2) **Indirect Integration:** Since not all IoT devices can support the requirements of the WoT paradigm due to resource limitations, this integration approach aims to connect such devices to the Web using gateways. A network designer might opt for this design to address various constraints like energy consumption and security. In this setup, an intermediary smart gateway positioned between the Web and the IoT devices abstracts the communication protocols or APIs of these devices, afterwards it exposes the devices' resources (e.g., sensed data) to the Web using RESTful APIs (i.e., HTTP/S).

IV. System Description and Scenarios

This section delves into a comprehensive system analysis, providing intricate details of its architecture, components, and functionalities. Additionally, we present detailed workflows illustrating how the system operates in various scenarios, offering insights into its functioning in different use cases.



(a) Direct integration. (b) Gateway-based integration.

Fig. 4: WoT integration patterns.

A. OpenStack and DeXMS integration

In today's IoT ecosystem, different IoT elements can be broadly classified into three main categories: IoT nodes, gateway nodes, and IoT services/applications. IoT nodes typically include household appliances, sensors, and actuators. These nodes have limited computational power, strict energy constraints, and restricted communication capabilities. Gateway nodes, on the other hand, serve as aggregators of sensor data and facilitate connections among IoT nodes and service providers. Gateway nodes have greater computing resources than IoT nodes and sometimes serve as replacements for IoT nodes. Finally, IoT services send commands to or gather data from various gateway nodes and offer user-specific or event-specific services through graphical interfaces, notifications, or applications.

To enhance interoperability in IoT environments, our WoT proposed architecture is depicted in Fig 5. In our design, we expect to have heterogeneous IoT nodes with distinct constraints, employing different communication protocols. To deal with the complexity of such an environment, our IoT gateways adapt those communication protocols and expose the devices' resources (data and functionalities) as RESTful resources exploitable using web standards (e.g., using HTTP/S).

To deploy the mediated WoT system on the remote gateways, we integrated DeXMS within a cloud-based OpenStack environment as shown in Fig. 1. The figure depicts the architecture of the two OpenStack subsystems Designate (blue subsystem) dealing with DNS domain and subdomains management, IoTronic (red subsystem) in charge of managing the IoT infrastructure (e.g., reachability and remote management), and DeXMS (purple subsystem) that deals with protocols mediation. Our system comprises two primary functions:

(i) Adapt the different IoT protocols (e.g., MQTT, AMQP, CoAP) and expose the devices' data and functionalities as HTTP exploitable resources. This is seamlessly achieved using the protocols mediators/adaptors generated by DeXMS.

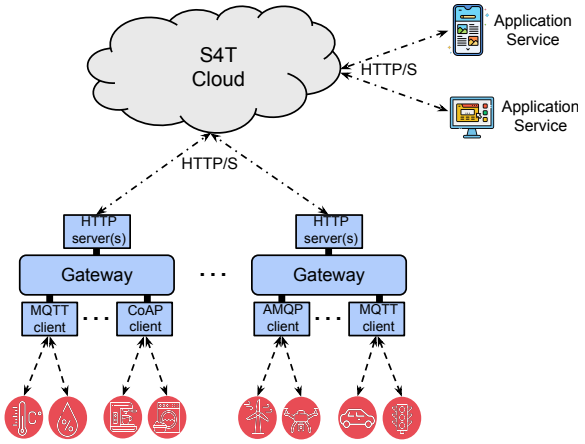


Fig. 5: Proposed WoT architecture.

(ii) Expose DeXMS mediators/adaptors using publicly routable URLs. This is achieved even when the mediators are deployed on gateways behind NATs and/or firewalls. It is worth mentioning that our system uses a cloud-based routing mechanism based on NGINX reverse proxies and Websocket tunnels to overcome the restrictions imposed by networking middleboxes [21].

In our proposed WoT architecture, the network designer has to remotely manage the gateway by deploying the necessary protocols' adaptors/mediators according to his/her specific needs and deployment requirements (e.g., the protocols to bridge: MQTT, CoAP, etc.). For this purpose and to make the solution customizable, we adopted the as-a-Service approach. In our approach, the designer should have total control over the gateway by deploying any kind of mediator regardless of the gateway location. Note that through the authentication system, *Keystone*, integrated with *IoTronic*, our system has mechanisms to authenticate and authorize users to have specific privileges in managing the mediators/adaptors. In the following, we detail the workflow for implementing our WoT subsystem on a remote gateway:

1) Using a web GUI, the user generates the configuration file for the mediator (i.e., the DeXIDL file) by specifying the involved protocols (e.g., MQTT and HTTP, CoAP and HTTP, etc.), the data structure, etc. This file is then generated and stored in the cloud.
2) The user interacts with *IoTronic* via the dashboard to deploy the mediator in a specific gateway. *IoTronic*, using the WAMP control channel, sends a Remote Procedure Call (RPC) to the gateway so as to instantiate the mediator. As parameters, the file ID stored in the cloud and the required ports to use are transmitted.
3) The gateway sends a request to the DeXMS service to generate the mediator. DeXMS generates the mediator in the form of a Java JAR file and sends it back (as a Bytestream) to the gateway.
4) On the gateway, the LR agent generates a Dockerfile and creates the mediator image.
5) Using the Docker API, LR runs the image with the necessary configurations (e.g., port mapping between the host and the container).
6) After generating and running the container, S4T publicly exposes the mediator's service (in this case, the HTTP server running on the gateway. See Fig. 5) through a specific URL.

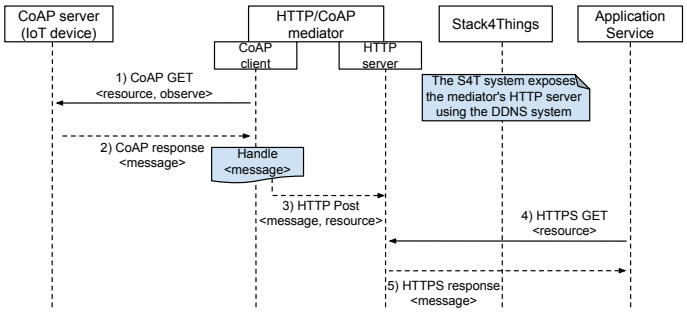


Fig. 6: Message translation from CoAP to HTTP.

While steps 4 and 5 are optional, an alternative approach involves direct execution of the mediator JAR file on the gateway. However, we advocate for containerization to bolster sandboxing mechanisms. This strategic choice enhances isolation and ensures seamless compatibility with orchestration engines like Kubernetes (K8s) and K3s. This architectural decision enables our solution to dynamically adapt to the fluctuating workloads inherent in IoT environments.

B. Integration scenarios

In this subsection, to better grasp system's inner workings, we present two scenarios of exposing data collected using CoAP and MQTT protocols to the Web.

In a nutshell, CoAP is an optimized REST protocol for sensor applications, which supports request/response and resource/observer architecture. MQTT, on the other hand, is a telemetry protocol and uses the publisher/subscriber (pub-sub) model, where the publisher manages a list of resources, also known as "topics", and subscribers can register to topics to obtain information when an event occurs.

Our WoT architecture provides, at the gateway level, interfaces to the IoT devices by supporting their protocols via the mediators' clients (see Fig. 5). On the other side (i.e., the top part), the mediator deploys an HTTP service (i.e., a server) to expose the device data/functionalities as RESTful resources: the Web-facing component of our system.

Fig. 6 illustrates the process of exposing data generated by an IoT device using CoAP. In the initial phase, we assume that the S4T process has already facilitated the exposure of the mediator's HTTP server to the Web, as detailed in [21]. As mentioned above, the mediator has an interface to the Web (i.e., the HTTP server) and another interface for communication with the IoT device, in this case a CoAP client.

The process unfolds as follows: first, the mediator's CoAP client initiates a GET request to the IoT device, utilizing the

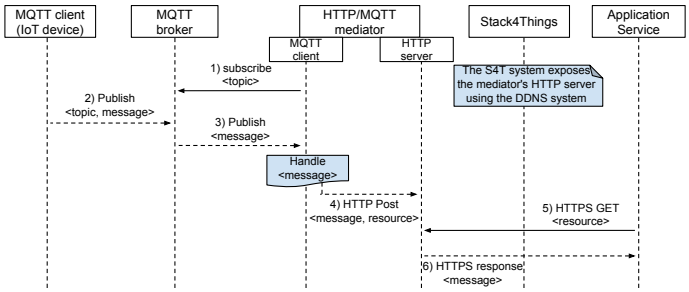


Fig. 7: Message translation from MQTT to HTTP.

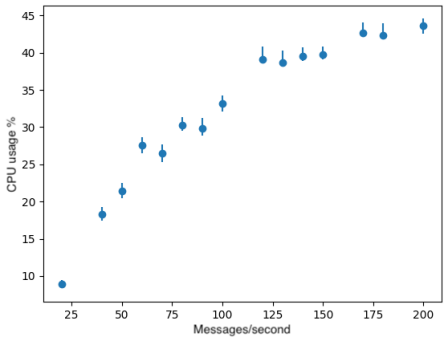


Fig. 8: CPU usage of the MQTT/HTTP mediator on the Raspberry Pi 3 with 95% confidence intervals.

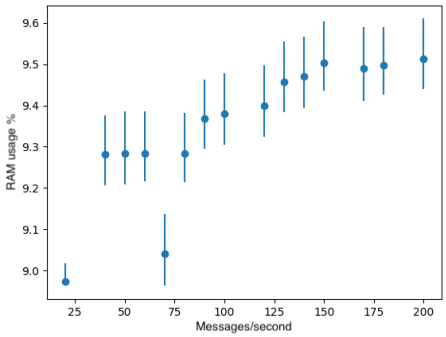


Fig. 9: RAM usage of the MQTT/HTTP mediator on the Raspberry Pi 3 with 95% confidence intervals.

observe option of CoAP. Upon data generation or a sensor change in state (step 2), the IoT device transmits the data to the mediator (i.e., the CoAP client). Subsequently, the mediator processes the data syntax, adapts its content, and sends it to the HTTP server using a POST request (step 3). Finally, the HTTP server receives and exposes the data within a designated resource (specified in the DeXIDL configuration file). As mentioned previously, the mediator's HTTP server is already accessible via an S4T-managed URL, including TLS certificate management. Consequently, the resource exposing data is accessible over the Web using that URL. An application or service can then access the HTTP server using GET requests (step 4) to retrieve the data generated by the IoT device.

Similarly, Fig. 7 illustrates the adaptation of messages between an MQTT publisher and a REST interface. Here, the mediator utilizes an MQTT client on the IoT device interface (left) to receive data from the broker. Initially, the process involves subscribing to the specific topic on which the IoT device publishes data (this configuration is specified in the DeXIDL). Once the broker receives data from an IoT device with the designated topic, it forwards it to the mediator (step 3). Subsequently, the mediator processes the data and issues an HTTP POST request to its HTTP server (step 4). The data can be retrieved (step 5) afterwards using the URL whose domain is initially associated by S4T to the mediator's HTTP server and whose path is specified in the DeXIDL. Consequently, applications and services can retrieve data generated by the IoT device as if interacting with an HTTP-based service.

In these two examples, we demonstrated the system's capability to expose data retrieved from IoT devices using CoAP and MQTT protocols to the Web. However, it is essential to highlight that our system offers more than just data retrieval; it also facilitates interactions with IoT devices for further purposes, including actuation. Essentially, our system supports bidirectional communication, allowing not only the retrieval of data but also the initiation of actions or changes on IoT devices. This bidirectional interaction capability significantly enhances the versatility and utility of the system, enabling comprehensive management and control of IoT devices and their associated functionalities.

V. System Evaluation

In this section, we undertake a series of tests aimed at evaluating the performance and resource utilization of our system. We specifically utilized a variety of instances emulating IoT devices to send traffic to a gateway where the protocol mediators are deployed. Notably, the proposed system does not impact the IoT devices themselves, as it is designed to be implemented on the gateways.

The gateway in our setup is a Raspberry Pi 3 Model B+ equipped with a Quad-Core 1.2GHz Broadcom BCM2837 64-bit CPU and 1 GB of RAM. As discussed in the preceding sections, our gateway hosts the S4T agent, LR, and an NGINX reverse proxy (see Fig. 2). In addition, the gateway hosts mediators responsible for protocol adaptation, enabling the exposure of data from protocols such as CoAP, MQTT, and AMQP as HTTP resources. Although our tests primarily utilize MQTT and CoAP, it is worth noting that other protocols, such as AMQP and DPWS, are also supported. In this context, our system is adept at managing multiple protocols, including secure ones when required. Notably, DeXMS has recently undergone enhancements to support protocols like CoAPS (CoAP over DTLS), MQTTS (MQTT over TLS), HTTPS (HTTP over TLS) and Secure WebSockets. The mediators within our system are proficient in handling encrypted communications, further bolstering security measures when needed.

In our testing phase, we specifically focused on evaluating the performance of an MQTT to HTTP and CoAP to HTTP mediators. For MQTT, we deployed an MQTT client (representing an IoT device) to publish messages according to a Poisson distribution, with a predetermined sending rate

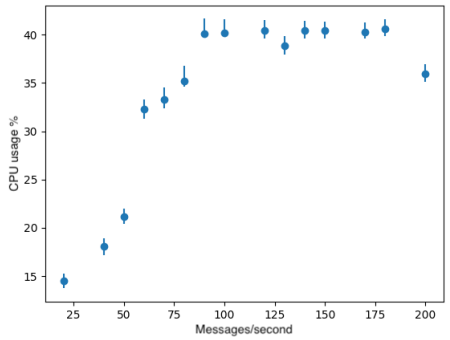


Fig. 10: CPU usage of the CoAP/HTTP mediator on the Raspberry Pi 3 with 95% confidence intervals.

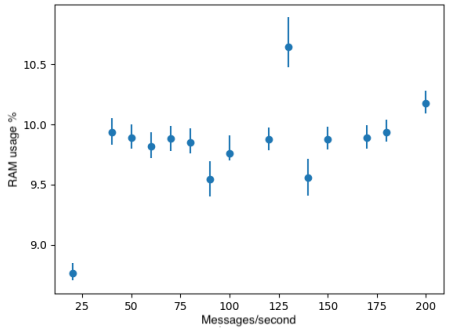


Fig. 11: RAM usage of the CoAP/HTTP mediator on the Raspberry Pi 3 with 95% confidence intervals.

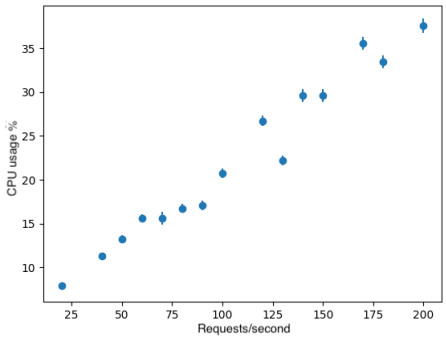


Fig. 12: CPU usage of the mediator's HTTP server running on the Raspberry Pi 3 with 95% confidence intervals.

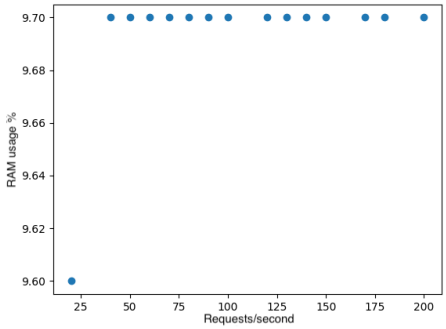


Fig. 13: RAM usage of the mediator's HTTP server running on the Raspberry Pi 3 with 95% confidence intervals.

(i.e., the number of messages per second). As for the broker, we opted to deploy it on the same gateway. It is important to note that the broker can be deployed elsewhere and the mediator will subscribe to the specified topic as configured in the DeXIDL file.

In Figure 8, we present the CPU usage of the mediator: it is evident from this graph that CPU consumption increases nearly linearly with the rise in the number of messages per second. We mention that for each sending rate value, we run the test for 5 minutes. The CPU usage reaches around 43% when generating 200 messages per second. We observed some message drops when reaching around 175 messages per second. Regarding RAM usage (see Fig. 9), it remained relatively stable, hovering around 9.4% regardless of the sending rate.

To address the issue of message drops, a suitable approach involves leveraging Kubernetes (K8s) or K3s. By configuring the environment accordingly, additional instances (i.e., containers) can be provisioned, and a load balancing strategy between the mediators can be established. This setup mitigates message drops instead of flooding a single mediator with messages.

During the testing phase of the CoAP to HTTP mediator, we employed a CoAP client to emulate an IoT device. This client generated data by altering the value of a resource, adhering to a Poisson distribution with a predetermined average rate. Similarly to the MQTT scenario, we observed an increase in CPU usage as the rate of resource variation per second increased (see Fig. 10). However, in this case, we observed message drops occurring at approximately 125 messages per second, which is less than the 175 observed for MQTT. This difference may be attributed to distinct message handling mechanisms, as employed by the mediator for each protocol. Specifically, while a Java queue (i.e., *Interface BlockingQueue*) is utilized to store incoming MQTT messages, CoAP messages, in contrast, are directly forwarded to the HTTP server without intermediate storage. Regarding RAM usage (illustrated in Fig. 11), we observed consistent resource consumption levels, maintaining stability at approximately 9.7% across all sending rates.

Another critical aspect concerning the mediator is its resource utilization on the Web-facing interface (i.e., the HTTP server). To evaluate this aspect, we needed to gauge the performance of the server that we use, which is, in this case, an instance based on the *Jakarta Servlet* Java library. As depicted in Fig. 13, the CPU usage of the HTTP server is showcased as it receives and responds to various HTTP GET requests. The values show a linear increase as the number of requests per second increases, reaching approximately 35% when handling 200 GET requests per second. For RAM usage (see Fig. 13), the mediator's HTTP server uses around 9.7% of the available RAM regardless of the number of requests received.

Regarding other elements of the WoT system, particularly the mechanisms employed by S4T (such as the NGINX reverse proxy and WS tunnels), comprehensive testing has already been conducted in previous work [21].

VI. Conclusion and future work

In this paper, we introduced our gateway-based WoT system, which addresses the challenges of protocol adaptation and resource reachability over the Web. Throughout the paper, we elaborated on how our solution effectively deals with the typical constraints encountered in IoT environments. Looking ahead, our future focus will be on enhancing the semantic aspect of the system. Specifically, we aim to ensure full compatibility with the WoT W3C Thing Description [26], thereby improving our solution's interoperability and semantic richness. This effort will enable seamless integration with other WoT-compliant systems and further enhance the usability and effectiveness of the system.

References

[1] S. Rani, P. Bhambri, A. Kataria, and A. Khang, "Smart City Ecosystem: Concept, Sustainability, Design Principles, and Technologies," in *AI-Centric Smart City Ecosystems*. CRC Press, 2023, pp. 1–20.

[2] T. Teixeira, S. Hachem, V. Issarny, and N. Georgantas, "Service oriented middleware for the internet of things: A perspective," in *Towards a Service-Based Internet: 4th European Conference, ServiceWave 2011, Poznan, Poland, October 26-28, 2011. Proceedings 4*. Springer, 2011, pp. 220–229.

[3] N. Srivastava and P. Pandey, "Internet of things (IoT): Applications, trends, issues and challenges," *Materials Today: Proceedings*, 2022.

[4] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, and Y. Wang, "A survey of network virtualization techniques for internet of things using sdn and nfv," *ACM Computing Surveys (CSUR)*, vol. 53, no. 2, pp. 1–40, 2020.

[5] W. H. Hassan *et al.*, "Current research on Internet of Things (IoT) security: A survey," *Computer networks*, vol. 148, pp. 283–294, 2019.

[6] A. van der Zeeuw, A. J. van Deursen, and G. Jansen, "The orchestrated digital inequalities of the IoT: How vendor lock-in hinders and playfulness creates IoT benefits in every life," *new media & society*, p. 14614448221138075, 2022.

[7] P. Desai, A. Sheth, and P. Anantharam, "Semantic gateway as a service architecture for iot interoperability," in *2015 IEEE international conference on mobile services*. IEEE, 2015, pp. 313–319.

[8] G. Colangelo and O. Borgogno, "Shaping Interoperability for the Internet of Things: The case for ecosystem-tailored standardisation," *European Journal of Risk Regulation*, vol. 15, no. 1, pp. 137–152, 2024.

[9] C. Bayılmış, M. A. Ebleme, Ü. Çavuşoğlu, K. Küçük, and A. Sevin, "A survey on communication protocols and performance evaluations for Internet of Things," *Digital Communications and Networks*, vol. 8, no. 6, pp. 1094–1104, 2022.

[10] A. Subash, "Iotivity–connecting things in iot," *TIZEN development summit*, 2015.

[11] D. Bruneo, S. Distefano, F. Longo, G. Merlino, A. Puliafito, V. D'Amico, M. Sapienza, and G. Torrisi, "Stack4Things as a fog computing platform for Smart City applications," vol. 2016-September, 2016, p. 848 – 853.

[12] S. M. Kala, V. Sathya, S. S. Magdum, T. V. K. Buyakar, H. Lokhandwala, and B. R. Tamma, "Designing infrastructure-less disaster networks by leveraging the alljoyn framework," in *Proceedings of the 20th international conference on distributed computing and networking*, 2019, pp. 417–420.

[13] M. Cristian Marin, M. Cerutti, S. Batista, and M. Brambilla, "A Multi-Protocol IoT Platform for Enhanced Interoperability and Standardization in Smart Home," in *2024 IEEE 21st Consumer Communications Networking Conference (CCNC)*, 2024, pp. 1–6.

[14] C. Lerche, N. Laum, F. Golatowski, D. Timmermann, and C. Niedermeier, "Connecting the web with the web of things: lessons learned from implementing a CoAP-HTTP proxy," in *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*. IEEE, 2012, pp. 1–8.

[15] M. Younan, S. Khattab, and R. Bahgat, "From the wireless sensor networks (WSNs) to the Web of Things (WoT): an overview," *J. Intell. Syst. Internet Things*, vol. 4, no. 2, pp. 56–68, 2021.

[16] M. Wazid, A. K. Das, K.-K. R. Choo, and Y. Park, "SCS-WoT: Secure communication scheme for web of things deployment," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 10 411–10 423, 2021.

[17] V. A. Barros, S. A. Junior, S. M. Bruschi, F. J. Monaco, and J. C. Estrella, "An IoT multi-protocol strategy for the interoperability of distinct communication protocols applied to web of things," in *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*, 2019, pp. 81–88.

[18] V. Thirupathi and K. Sagar, "Web of Things an intelligent approach to solve interoperability issues of Internet of Things communication protocols," in *IOP Conference Series: Materials Science and Engineering*, vol. 981, no. 3. IOP Publishing, 2020, p. 032094.

[19] Z. Benomar, F. Longo, G. Merlino, and A. Puliafito, "A Stack4Things-based web of things architecture," in *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*. IEEE, 2020, pp. 113–120.

[20] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito, "Stack4Things: An OpenStack-based framework for IoT," in *2015 3rd International Conference on Future Internet of Things and Cloud*. IEEE, 2015, pp. 204–211.

[21] Z. Benomar, F. Longo, G. Merlino, and A. Puliafito, "A Cloud-Based and Dynamic DNS Approach to Enable the Web of Things," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 6, pp. 3968–3978, 2022.

[22] G. Bouloukakis, N. Georgantas, P. Ntumba, and V. Issarny, "Automated synthesis of mediators for middleware-layer protocol interoperability in the IoT," *Future Generation Computer Systems*, vol. 101, pp. 1271–1294, 2019.

[23] "MQTT version 3.1.1," https://goo.gl/1WdTPZ.

[24] Z. Shelby and K. Hartke, "The Constrained Application Protocol (CoAP)," Tech. Rep., 2014.

[25] I. Fette and A. Melnikov, "The WebSocket Protocol," Tech. Rep., 2011.

[26] V. Charpenay, S. Käbisch, and H. Kosch, "Introducing Thing Descriptions and Interactions: An Ontology for the Web of Things." in *SR+ SWIT@ ISWC*, 2016, pp. 55–66.