ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

WoT on The Extreme Edge (WoTTEE): Enabling W3C Web of Things for Micro-controllers

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Sciullo, L., Castiglione, C., Trotta, A., Di Felice, M. (2022). WoT on The Extreme Edge (WoTTEE): Enabling W3C Web of Things for Micro-controllers. New York : IEEE [10.1109/wf-iot54382.2022.10152179].

Availability:
This version is available at: https://hdl.handle.net/11585/964287 since: 2024-02-29

Published:
DOI: http://doi.org/10.1109/wf-iot54382.2022.10152179

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

13 November 2026

WoT on The Extreme Edge (WoTTEE): Enabling W3C Web of Things for Micro-controllers

Luca Sciullo*, Cristian Castiglione*, Angelo Trotta*, Marco Di Felice*†,
* *Department of Computer Science and Engineering, University of Bologna, Italy*
† *Advanced Research Center on Electronic Systems "Ercole De Castro", University of Bologna, Italy*
Emails: {luca.sciullo, angelo.trotta5, marco.difelice3}@unibo.it, cristian.castiglione@studio.unibo.it

***Abstract*—Edge computing has emerged as a viable approach to minimize the latency of time-critical Internet of Things (IoT) applications. The new frontier of research is to offload tasks directly on micro-controllers, i.e. on the same device generating the sensory data. However, this approach requires to cope with to the limited computational capabilities of the devices and consequently to adapt existing techniques and tools to the extreme edge. In this paper, we contribute to this research direction by investigating how to enable W3C Web of Things (WoT) capabilities on micro-controllers, for increased performance and interoperability purposes. Given the excessive complexity of the original proposal, we propose a revised architecture of the Web Thing (WT) that can fit the limited resources of constrained devices while maintaining the compatibility with the WoT standard. Then, we present the WoTTEE framework, a software suite that supports and facilitates the deployment, installation and monitoring of edge-oriented WTs. Finally, we validate the operations of WoTTEE in a small testbed and demonstrate the capability to support adaptive IoT systems where micro-controllers are able to dynamically switch among different IoT network protocols thanks to the Thing Description (TD) mechanism of the W3C WoT.**

***Index Terms*—Internet of Things, W3C Web of Things, embedded systems, performance evaluation**

I. Introduction

Several novel use-cases of the Internet of Things (IoT) require to acquire and process the sensory data with strict latency constraints [1]. For instance, autonomous robots used in Industry 4.0 scenarios must sense the current environment, detect the presence of possible obstacles and, in real-time, compute an obstacle-free path towards the goal. Edge computing has emerged as a viable approach to reduce the latency in IoT scenarios by offloading the data management tasks nearby the data sources rather than executing them on the cloud [2]. The new frontier of edge computing is how to execute the tasks directly on the micro-controllers, i.e. on the same IoT devices producing the sensory data [3]; for this reason they are also referred as the extreme edge. While minimizing the data acquisition latency, this approach must cope with the limited computational and storage capabilities of the IoT devices. Hence, additional research efforts are required to adapt existing algorithms, techniques and tools to be executed at the extreme edge layer. In this paper, we contribute to this research direction by investigating how to enable W3C Web of Things (WoT) capabilities on micro-controllers, for increased interoperability and performance purposes.

The Web of Things (WoT) [4] denotes a wide range of approaches aimed at mitigating the device fragmentation issue in the IoT domain by exploiting the unifying power of Web technologies. Among others, the WoT standard promoted by the W3C [5] is becoming the reference solution to enable interoperability across IoT platforms and application domains. Using their words, "*the W3C WoT architecture is designed to describe what exists rather than to prescribe what to implement*" [5]. To this aim, any IoT device of the WoT ecosystem is provided with a WoT Thing Description (TD), i.e. a collection of meta-data describing how to interact with it in terms of properties, actions and events (also called the affordances), the protocols in use, the way to encode/decode data, and the security mechanisms. Despite its recent appearance, several studies have shown the potential of the W3C WoT standard on different IoT markets, from automotive to industry 4.0 [6] [7]. The standard includes the definition of a run-time environment (also called Servient) and an application programming interface (API) to work with the W3C Web Things (WT), i.e. enabling software clients to publish a new WT (WT expose) or to access the affordances of an exposed WT (WT consume) [8]. The current W3C WoT architecture is independent from its implementation and includes many patterns in order to integrate any IoT device into a new WoT ecosystem. Since the run-time functionalities are too demanding for most of the constrained micro-controllers, the W3C WoT architecture envisages the usage of gateways that expose the TD on behalf of the edge device and manage the communication with it. A drawback of this solution is the need for additional software adapters, making the transition of a pre-existing IoT scenario to the WoT paradigm not that immediate. In addition, the usage of the intermediate gateway may affect the system performance, increasing the end-to-end latency.

In this paper, we describe the design, implementation and validation of edge-based solutions for W3C WoT deployments. Our solution enables the main WoT functionalities, such as WT expose and consume operations, directly on the micro-controllers, hence removing the need of any intermediate layer and fitting the requirements of time-critical IoT scenarios. This constitutes a significant advance with respect to the existing WoT Servient implementations [9] [10], which do not support extreme edge devices. As a result, our tool aims at increasing the adoption of the new W3C standard by taking into account

the vertiginous growth of the IoT sensors market size[1]. More in detail, we provide three main contributions in this paper:

- We propose an edge-oriented architecture of the WT in order to reduce the computational resources required by the Servient. The revised architecture is able to support a significant subset of the main operations of a WT and to guarantee WoT-compliant interactions with the WT while taking into account the characteristics of the edge environment in its internal implementation.
- We present WoTTEE, a software framework facilitating the deployment and installation of a WT on a micro-controller. Through a user-friendly graphical interface, the user can insert the behaviour of the WT and select the network protocols in use; the tool is in charge of generating the firmware code that exposes the WT in a network environment, in a guided and semi-automatic way.
- We validate WoTTEE in a small testbed composed of two IoT micro-controllers (ESP32), one exposing a WT related to an ultrasonic sensor, the other working as data collector and consuming the other WT. In addition, we demonstrate the possibility to select the network protocol of the WTs at run-time in order to meet the latency requirements of the IoT application.

In [11], we presented a preliminary study on how to build a WoT Servient for micro-controllers. The new framework includes support for multiple protocol bindings, novel WoT functionalities (e.g. WoT consume) and a completely revised interface and internal implementation.
The rest of this paper is structured as follows: in Section II we provide a brief review of the W3C WoT standard and its components. Section III introduces the WoTTEE architecture, while its implementation is described in Section IV. Section V introduces the testbed and the performance evaluation. Conclusions and future works are discussed in Section VI.

II. Related Work

The W3C Web of Things (WoT) provides a set of standardized technology building blocks that help to tackle the fragmentation issue in the IoT landscape by following the well-known Web paradigm [5]. Figure 1 shows the abstract architecture of a W3C WoT scenario. Several integration patterns such as Thing-to-Thing, Thing-to-Gateway, Thing-to-Cloud, Gateway-to-Cloud are available in order to interconnect IoT devices. The core component of the W3C WoT architecture is the Web Things (WT), which represents "*an abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description*" [5]. The WT architecture is modular and includes five main layers: (*i*) the behaviour, (*ii*) the interaction affordances, (*iii*) the data schemas, (*iv*) the security configuration, and (*v*) the protocol bindings. The first layer defines the behaviour of a

[1]According to BCC research (https://www.bccresearch.com/), the global market for IoT sensors should grow with a compound annual growth rate (CAGR) of 27.4% for the period of 2021-2026.

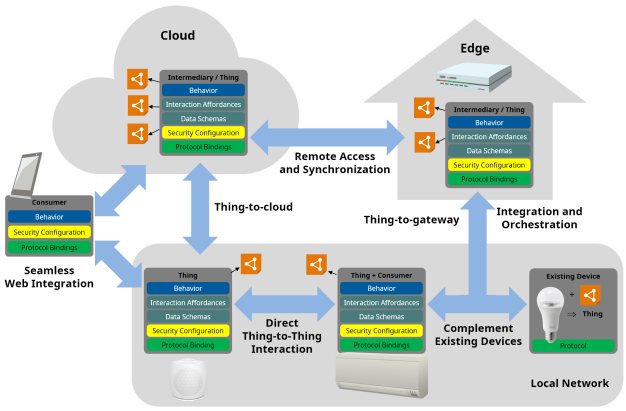


Fig. 1. The W3C WoT abstract architecture proposed in [5].

WT and the handlers for the interaction affordances. The latter specify how clients should interact with the WT based on the Property-Action-Event (PAE) paradigm: a Property represents an internal state variable of the WT, each command that can be invoked on the WT is mapped to an action, while each notification fired by the WT is an event. The third layer contains the Information Model, i.e., the payload structure exchanged between the WT and the client. The fourth layer describes the access control mechanism to the Thing's affordances. Finally, the protocol bindings layer describes the mapping between the interaction affordances and the IoT network protocol in use (e.g. CoAP). The Thing Description (TD) is a collection of metadata describing the capabilities of a WT with respect to the layers aforementioned, excluding the behaviour. By default, the TD is serialized by using the *JSON-LD*[2] language. In addition, the WoT building blocks are implemented by a software stack called Servient. The latter plays the crucial role of turning a TD into a software agents. As the word suggests, the Servient can act both as server and client. In the first case, the Servient is responsible to *expose* the WT, hence enabling the possibility for a client to interact with its properties, actions and events. In the second case, a Servient allows to *consume* an exposed WT i.e., a WT that has been instantiated by someone else and that is ready to be used. *Node-wot* [9] is the most popular Server implementation and is maintained by the W3C working group. The framework is based on *Typescript* and *NodeJS* hence it cannot be executed on micro-controllers with constrained hardware resources. There exists also other Servient implementations. For instance, we cite the *WotPy* [10] tool, an experimental implementation of a W3C WoT Runtime in *Python* and the *Java*-based implementation provided by the Smart Networks for Urban Participation (SANE) project [12]. However, none of them supports extreme edge devices.

III. Software Architecture

The general architecture of a WT previously described in Section II may require excessive computational resources for

[2]https://json-ld.org

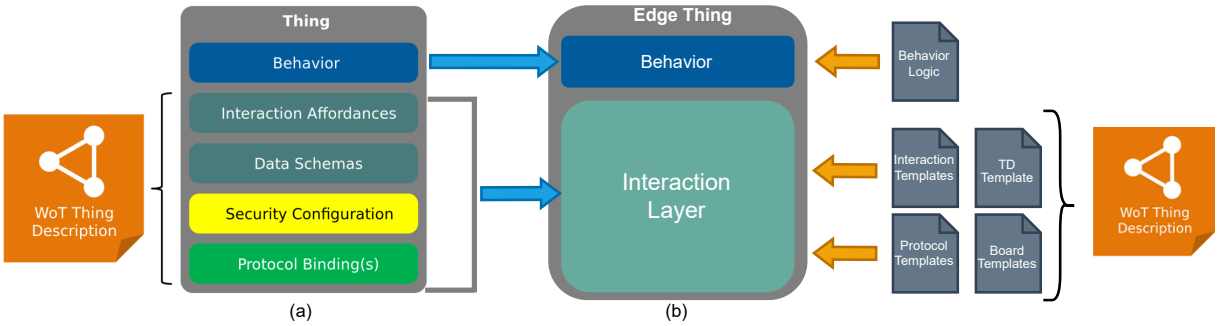


Fig. 2. The edge-oriented architecture of a Web Thing implemented within the WoTTEEE framework.

memory-constrained and single-thread IoT devices. For this reason, our first research contribution was to revise the internal architecture of the WT in order to make its design edge-oriented. The proposed architecture is described in Section III-A; it differs from the reference solution for the internal building blocks and the deployment process however it allows to expose and consume W3C compliant WTs. Then, in Section III-B, we detail the architecture of the WOTTEE framework that helps users in deploying, installing and monitoring the WT on the micro-controller.

A. *Edge-oriented WT Architecture*

The original WT architecture considers a strict separation of functionalities into five independent building blocks: from the implementation perspective, they translate into different software modules that interact each others. To reduce such complexity, the proposed edge-oriented WT architecture shown in Figure 2 follows a flattened design, in which the five original layers have been reduced to two only. More in detail, the *Behaviour* layer has been preserved with the original functionality. Vice versa, the other four layers of the original WT architecture have been grouped into a single *Interaction layer* that deals with the networking capabilities of the board, including security, data schemas, and affordances representation. At the same time, in order to guarantee compatibility in terms of functionalities supported by the WT, we reduced the number of software-generated behaviors that have been coded and turned into static functionalities. For example, each affordance has been coded into a separate endpoint/function for each different IoT protocol. This is a major difference with respect to the original W3C standard, where there is a single service capable of handling different protocols for the same affordance. While this approach increases the size of the source code, it drastically reduces the complexity of the internal messaging process and hence the computational resources in use. The overall firmware generation is made via source templates, i.e. software modules that are available in the WOTTEE platform (see details later) and that are merged with the user's code in order to create the final sketch running on the IoT device. The user's code is referred as *Behaviour logic* and includes the business logic of the IoT application and the implementation of the affordances, e.g. the body of the actions. We considered different kinds of source templates:

- *Board templates* contain the instructions required by the sketch for being instantiated on a specific board (e.g. include directives at the beginning of the sketch);
- *Protocol templates* contain the implementation of a specific network protocol (HTTP, CoAP, etc);
- *Interaction templates* help building the functions for the affordances and mainly consist of the skeleton for producing getters and setters operations for the properties and actions of the WT;
- *TD template* provides the skeleton of the TD that will be exposed by the edge device.

B. *WOTTEE Framework Architecture*

The WOTTEE framework is the stand-alone software that supports the user in developing an edge-oriented WT, installing it on a micro-controller and monitoring its data in real-time. The architecture has been designed according to a micro-service pattern. As shown in Figure 3, the framework is based on two layers, i.e., the *Edge layer* and the *Application layer*; in addition, we distinguish between *Front-end* and *Back-end* functionalities. The *Application layer* includes all the services that can be deployed also in the *cloud* since they not require a direct communication with the physical device: they are the *Graphical User Interface (GUI)*, the *Dashboard*, the *Core* and the storage services, namely the *Database (DB)* and the *Object Storage (OS)*. The *GUI* allows the user to create a new project, modify an existing sketch through editor or HTML form, and select the commands to run, such as start the compilation process. Through the *Dashboard* service, the user can retrieve the data stored in the Database and display them through plots. The *DB* and the *OS* services are in charge of storing all the data produced and consumed by the application: in the first case, the *DB* stores the information required by the GUI, like the user data, the sketches metadata, and the board settings, while in the second case the *OS* service stores the sketches, hence the source code, produced by the user. The *Servient Builder (SB)* service is in charge of generating the sketches for the boards, according to the protocol bindings and the settings chosen by the user through the GUI and that are stored in the *DB*. The

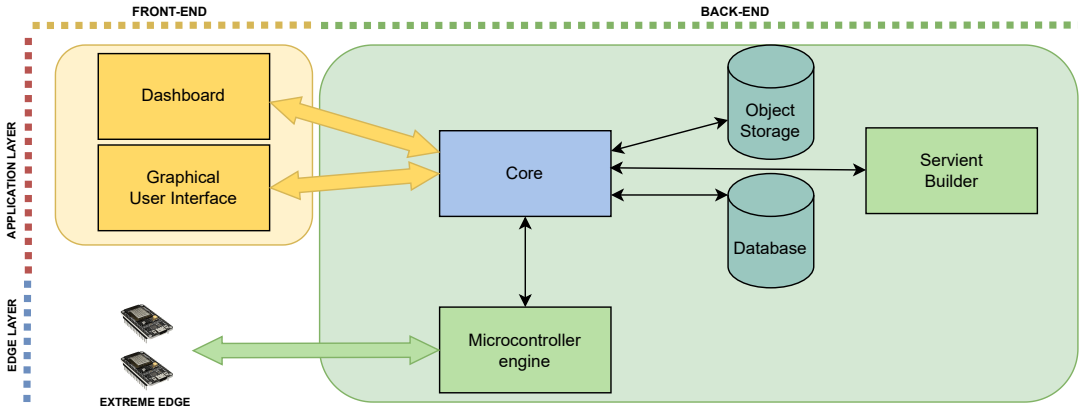


Fig. 3. The WoTTEE framework architecture with the edge/cloud layers and the back-end/front-end components.

Core service enables the communication between the back-end services and the front-end ones, translating each request coming from the user into proper commands for the other services. Finally, at the *Edge layer*, the *Microcontroller Engine (ME)* manages the communication between the physical device and the rest of the framework: it is in charge of compiling and uploading the final sketch on the target board.

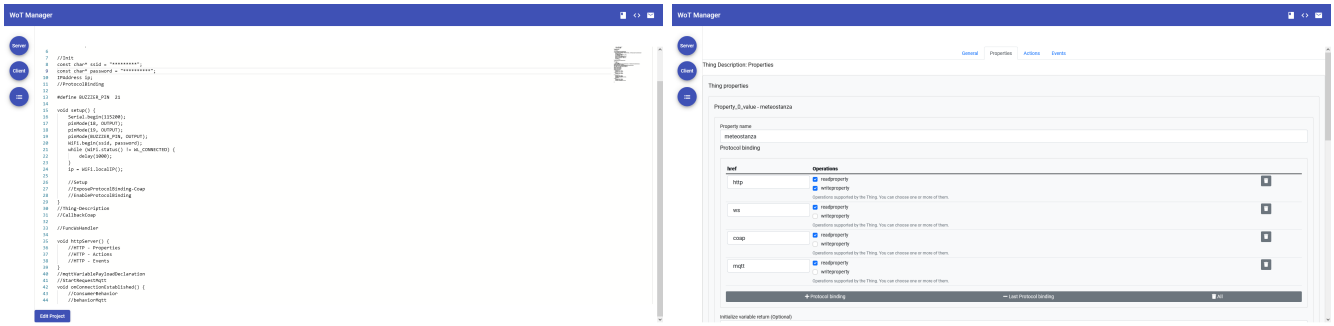
IV. Implementation

In the following we detail the technologies that have been used to implement the software architecture presented in the previous Section. The GUI service is a Web application written in *Typescript* and using the high-level *Angular*[3] framework. All settings are encoded in JSON format and handled through the *JSON editor* open-source library, that automatically generates an HTML form starting from a *JSON Schema* and can be easily integrated with the most popular *CSS* frameworks (like Bootstrap, Spectre, Tailwind). The generated source code can be manually modified through a dedicated editor in the GUI, implemented with the *Monaco Editor*[4] library; the latter includes also the syntax highlighting and syntax checking for several programming languages, including the *C* one. Figures 4(a) and 4(b) show two screenshots of the GUI, one related to code editing the other related to the TD-PAE definition (more specifically, the definition of properties) with the selection of the active protocol bindings. The Dashboard is a Grafana[5] instance, while the *Core* and the *SB* services are *Typescript* programs that runs over the *NodeJS* engine and written using the *NestJS* framework[6]: they expose a REST API for interacting with the GUI and the *Core* respectively. While the *Core* is mainly a proxy for the requests coming from/to other services, the *SB* service implements a specific procedure for generating the sketches or part of them. This is achieved by combining the settings and the resources -retrieved by the storage services- contained within the requests from the core with source templates, that are stored in the service as plain text files. For the storage of the sketches source code in the *OS*, we use *MinIO*[7], an open source version of the Amazon S3 object storage, while all the other data handled by the framework are stored in the *DB* service through a *MongoDB*[8] instance. Finally, the *ME* service is written in *Python* and implements a REST API through the usage of the *Flask*[9] framework. The main task of this module is to compile and upload the sketches on the selected board, and for this reason we rely on the *Pyduinocli*[10] library for the interaction with the physical devices and the download of the necessary additional board libraries required by the sketch. Each microservice has been containerized using *Docker*, forming a stack that can be easily orchestrated through *Docker Swarm* or *Kubernates*. The deployment of the stack can be obtained in two ways: *locally* -with all the services deployed on the same machine to which the micro-controller is attached to-, or *hybrid-locally* -with all the services deployed in the cloud but the *ME* hosted on the same machine where physical device is available. In this case, the host machine needs to be added to the swarm, or alternatively, the *ME* can be deployed as a standalone service.

V. Performance Evaluation

In this Section we investigate the performance of the WOTTEE framework and in particular its ability to support multi-protocol data acquisition in WoT-enabled edge devices. The scenario for the experiments is depicted in Figure 5. Our system is composed of two IoT devices and a workstation. The extreme edge devices are constituted by ESP32 boards[11]; the first device (on the left in Figure 5) is equipped with an ultrasonic distance sensor and is used as a sensor node, while

[3]https://angular.io/
[4]https://microsoft.github.io/monaco-editor/
[5]https://grafana.com/
[6]https://nestjs.com/
[7]https://min.io/
[8]https://www.mongodb.com
[9]https://flask.palletsprojects.com/en/2.1.x/
[10]https://github.com/Renaud11232/pyduinocli
[11]https://www.espressif.com/en/products/socs/esp32



(a) (b)

Fig. 4. Two screenshots of the WoTTEE GUI: the Figure on the left shows the code generation process, with the user-defined code and the code imported from the source templates. The Figure on the right shows the process of TD construction, and more specifically, the definition of the WT properties with the active protocol bindings.

the second device is used as a proxy data collector and it is responsible for the data transmission to a server (on the right in Figure 5). The experimental setup models a generic Wireless Sensor Network (WSN) scenario, with extreme edge devices sensing the environment and a network cluster-head (CH) collecting the measurements and transmitting them to a central database.

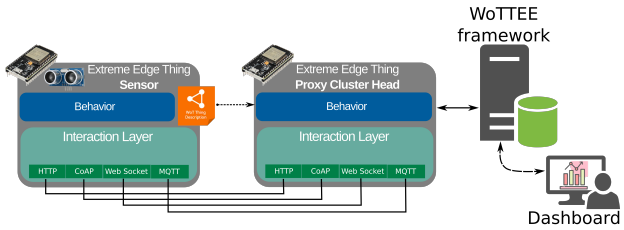


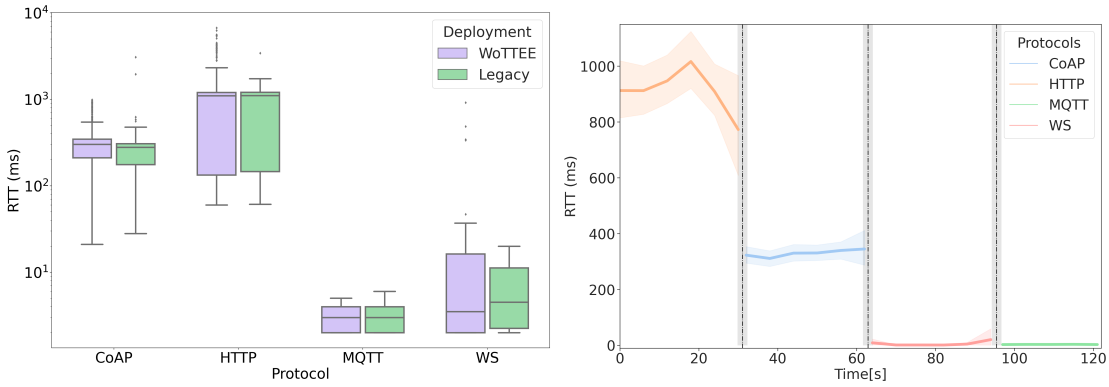
Fig. 5. The experimental scenario considered in our tests.

The local workstation shown in Figure 5 is used to host the full WoTTEE framework with all the modules described in Section III. Hence, it contains the database with the sensing measurements and gives access to the dashboard through which the user can visualize the collected time-series.

In the first experiment, we measure the overhead introduced by the WoT standard to the device's activities. We consider the Round-Trip Time (RTT), computed as the time interval, measured in milliseconds, from when the CH sends a data request to when it receives a response from the sensor node. We considered two alternative deployments: (*i*) a WoT deployment using the WoTTEE framework and (*ii*) a legacy deployment in which we develop the firmware without the WoT paradigm. In both cases we tested four different network protocols: *HTTP*, *CoAP*, *Web Sockets (WS)*, and *MQTT*. Figure 5 details the WoT-enabled use-case: when the CH needs to retrieve the sensor data from the distance sensor, it needs to *consume* first the TD (shown in orange in Figure 5). As a result, the CH is aware of the communication protocols supported by the other device. Vice versa, in the *legacy* deployment, there is no initial setup and the CH queries the sensor device directly. The result of the comparison is depicted in Figure 6(a), measuring the RTT index for the four communication protocols. It is easy to notice that the overhead introduced by the WoTTEE framework is negligible, while the choice of the IoT protocol has a considerable impact on the latency. Based on this result, we demonstrate the ability of a WoT-enabled device to support multiple protocols thanks to the TD and to dynamically switch to the preferred communication method decided by the CH. For pure testing purposes, we used a round-robin policy to switch among all the available IoT network protocols. However, it is worth mentioning that any policy addressing specific Quality of Service (QoS) requirements can be loaded into the CH behavior. Figure 6(b) shows the outcome of the dynamic experiment; each protocol is used for 30 seconds. From the Figure, we can notice that the switch time from two consecutive protocols is in the order of 2 seconds. Finally, Figure 7 shows the Dashboard that allows the user to check and analyze the correct operations of the deployed WSN. More specifically, the figure shows the sensor values over time when using a round-robin policy over the available communication protocols.

VI. Conclusions and Future Works

In this paper, we presented WoTTEE, a software framework enabling the execution of W3C Web Things (WT) on micro-controllers. The tool assists the user in the WT creation and code generation processes thanks to the usage of source templates. Consequently, it supports the main operations of the legacy W3C WoT standard while taking into account the constrained resources at the extreme edge. We validate the operations of WoTTEE in a small IoT testbed. On the one side, the experimental results demonstrate that our framework does not introduce any performance bottleneck compared to a legacy, non WoT-based implementation. On the other side, it demonstrates the advantage of an edge-oriented WT solution, such as the possibility to switch at run-time the IoT acquisition protocol thanks to the protocol-agnostic nature of the TD. We believe that the success of the W3C WoT initiative depends on its wide-spread adoption from the industry and research community, and, to this aim, the presence of a wide Software ECO-system (SECO) of supporting tools is of paramount



(a) (b)

Fig. 6. The boxplot of the RTT for the WoTTEE-legacy comparison is shown in Figure 6(a). The average RTT when switching among the four IoT protocols supported by WoTTEE is shown in Figure 6(b).

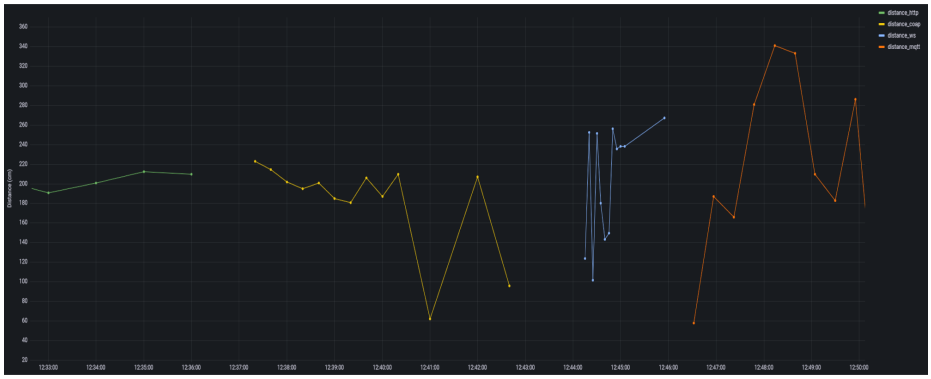


Fig. 7. A screenshot of the *Dashboard* module with the sensor data transmitted by using different IoT network protocols over time.

importance. Our research contribution goes in this direction by facilitating the integration to the WoT scenario of micro-controllers, whose market is one of the driving factors of the IoT growth. There are several future research activities related to WoTTEE. First of all, we plan to expand the list of source templates, considering additional IoT boards (at present, we support ESP32 and Arduino boards). Second, we plan to extend the evaluation by considering the heterogeneous IoT condition monitoring scenarios developed by the Arrowhead Tools project.

Acknowledgements

This work has been funded by by the EU ECSEL Joint Undertaking under grant agreement No 826452 (Arrowhead Tools), within the EU Horizon 2020 research and innovation programme.

References

[1] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards low-latency service delivery in a continuum of virtual resources: State-of-the-art and research directions," *IEEE Communications Surveys Tutorials*, vol. 23, no. 4, pp. 2557–2589, 2021.

[2] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, and D. O. Wu, "Edge computing in industrial internet of things: Architecture, advances and challenges," *IEEE Communications Surveys Tutorials*, vol. 22, no. 4, pp. 2462–2488, 2020.

[3] A. Ghibellini, L. Bononi, and M. Di Felice, "Intelligence at the iot edge: Activity recognition with low-power microcontrollers and convolutional neural networks," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, 2022, pp. 707–710.

[4] L. Sciullo, L. Gigli, F. Montori, A. Trotta, and M. D. Felice, "A survey on the web of things," *IEEE Access*, vol. 10, pp. 47 570–47 596, 2022.

[5] W3C Working Group. (2021) WoT Reference Architecture (W3C Recommendation 9 April 2020). [Online]. Available: http://www.w3.org/TR/wot-architecture/

[6] B. Klotz, S. K. Datta, D. Wilms, R. Troncy, and C. Bonnet, "A car as a semantic web thing: Motivation and demonstration," in *2018 Global Internet of Things Summit (GIoTS)*, 2018, pp. 1–6.

[7] L. Sciullo, S. Bhattacharjee, and M. Kovatsch, "Bringing deterministic industrial networking to the w3c web of things with tsn and opc ua," in *Proceedings of the 10th International Conference on the Internet of Things*, ser. IoT '20. New York, NY, USA: Association for Computing Machinery, 2020.

[8] Z. Kis, D. Peintner, C. Aguzzi, J. Hund, and K. Nimura, "Web of things (wot) scripting api," W3C Recommendation, Nov. 2020, https://www.w3.org/TR/wot-scripting-api/.

[9] W3C Working Group. (2019) Eclipse Thingweb node-wot. Source repository. [Online]. Available: https://github.com/eclipse/thingweb.node-wot

[10] A. G. Mangas and F. J. S. Alonso, "Wotpy: A framework for web of things applications," *Computer Communications*, vol. 147, pp. 235–251, 2019.

[11] L. Sciullo, I. D. R. Zyrianoff, A. Trotta, and M. D. Felice, "Wot micro servient: Bringing the w3c web of things to resource constrained edge devices," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2021, pp. 161–168.

[12] "Smart Networks for Urban Participation (SANE)," 2021. [Online]. Available: https://sane.city/