Deploying W3C Web of Things-Based Interoperable Mash-up Applications for Industry 4.0: A Testbed

Luca Sciullo, Angelo Trotta, Lorenzo Gigli, Marco Di Felice

Deploying W3C Web of Things-based Interoperable Mash-up Applications for Industry 4.0: A Testbed

Luca Sciullo, Angelo Trotta, Lorenzo Gigli, and Marco Di Felice

Department of Computer Science and Engineering
University of Bologna, Italy
{luca.sciullo,angelo.trotta5,marco.difelice3}@unibo.it, lorenzo.gigli@studio.unibo.it

Abstract. In Industry 4.0 scenarios, novel applications are enabled by the capability to gather large amount of data from pervasive sensors and to process them in order to devise the "digital twin" of a physical equipment. The heterogeneity of hardware sensors, communication protocols and data formats constitutes one of the main challenge toward the large-scale adoption of the Internet of Things (IoT) paradigm on industrial environments. To this purpose, the W3C Web of Things (WoT) group is working on the definition of some reference standards intended to describe in a uniform way the software interfaces of IoT devices and services, and hence to achieve the full interoperability among different IoT components regardless of their implementation. At the same time, due also to the recent appearance of the WoT W3C draft, few testbed and real-world deployments of the W3C WoT architecture has been proposed so far in the literature. In this paper, we attempt to fill such gap by describing the realization of a WoT monitoring application of a generic indoor production site: the system is able to orchestrate the sensing operations from three heterogeneous Wireless Sensor Networks (WSNs). We describe how the components of the W3C WoT architecture have been instantiated in our scenario. Moreover, we demonstrate the possibility to decouple the mash-up policies from the network functionalities, and we evaluate the overhead introduced by the WoT approach.

1 Introduction

Recently, the Industry 4.0 has emerged as a new paradigm able to radically transform the organizations' production and business in a myriad of sectors beside the smart manufacturing one [1] [2]. The core of the paradigm that justifies also its generality and viability on different markets is the concept of Cyber-physical Systems (CBSs), i.e. the strict integration between physical elements and computational data enabled by the recent advances on the Internet of Things (IoT) [1]. Hence, the ability to collect, aggregate and analyze sensor data is crucial for the growth of the Industry 4.0 model. At the same time, today's IoT is a chaotic environment characterized by heterogeneous hardware devices, network protocol stacks and data formats. The current fragmentation can significantly increase

the deployment costs, since collected data can remain largely inaccessible in an integrated way unless investing significant manual effort [2]. At the same time, interoperability can represent an opportunity for next-generation IoT applications: the McKinsey report in [3] quantifies in 40% the additional market value that might be provided by achieving full interoperabilty among IoT ecosystems. Among the several approaches proposed so far in order to address IoT interoperability problems, the Web of Things (WoT) has gained considerable attention, thanks to the popularity and well-known unifying nature of the Web [4][5]. Differently from other stack-oriented solutions (e.g. 6LoWPAN), WoT-based approaches propose to achieve system interoperability at the application layer, abstracting from the sensing and communication technologies: in a first approximation, Things are represented as Web resources, and all the interactions toward and between Things are mapped over Representational State Transfer (REST) services [5]. At the same time, given the lack of a reference architecture, several different WoT frameworks have been proposed in the literature (e.g. [6][7][8][9]), introducing further fragmentation and the consequential need of devising ad-hoc solutions for the system integration. Breaking the deadlock, the World Wide Web Consortium (W3C) has recently proposed a reference architecture of the WoT [10] that formally describes the interfaces allowing IoT devices and services to communicate with each other, regardless of their underlying implementation. In the W3C WoT vision, everything can be considered a Thing and to this purpose, each Thing is associated to a Thing Descriptor (TD) providing general metadata as well as the interactions, data model, and security mechanisms of a Thing [10]. In addition, a TD can be serialized and semantically annotated via the JSON-LD language, hence representing a uniform model to enable Machine-to-Machine (M2M) communication toward a Thing and enabling several semantic features, like for instance the Thing Discovery (TD). The generality of the WoT architecture makes it suitable for all those scenarios characterized by the need of aggregating data from multiple, heterogeneous sources, like the Industry 4.0. However, due also to its recent appearance, few implementations and test-bed of the W3C WoT have been described so far in the literature [11][12].
In this paper, we attempt to fill such gap, by describing the design and implementation of a WoT testbed, consisting of a monitoring system of a generic production site that must retrieve and process sensor data from heterogeneous devices using different wireless access technologies (i.e. Wi-Fi, 802.15.4/Zigbee, BLE). The overall goal is to devise mash-up applications able to orchestrate the sensing operations over the target scenario regardless of the network protocols and hardware, hence decoupling the rationale of the monitoring process (e.g. minimal scenario coverage) from its implementation (i.e. the technology used to query the sensor). More specifically, we introduce three main contributions in this study:

– First, we describe how the scenario can be modeled within the WoT W3C framework. We associate one Thing to each sensing device, and one Thing to the sensor network, by defining the metadata of each. Moreover, we discuss

how the components of the WoT W3C architecture have been made concrete in our application.
- Second, we describe the design and implementation of mash-up applications aimed to orchestrate the sensing operations on the target scenario. We considered four different sensing policies, aimed to balance the coverage of the scenario with the network performance (e.g. delay, packet delivery ratio and energy). All the policies are in charge of dynamically selecting the sensors to query at each instant in order to maximize the policy-specific metric: to this purpose, given the dinamicity of the environment, we employ the Reinforcement Learning (RL) framework [17] to optimally balance the exploration-exploitation tasks.
- Third, we report a subset of the experimental results from the WoT testbed. We investigate the performance of the sensing mash-up applications with respect to the policy goal (e.g. delay), and the convergence over time. Moreover, we show the benefit introduced by the WoT architecture in terms of adaptive design, i.e. the possibility to dynamically switch the sensing policies over time without re-configuring the communication infrastructure, and the overhead introduced by the WoT components.

The rest of the paper is structured as follows. Section 2 reviews the WoT W3C architecture, and its recent applications. Section 3 introduces the test-bed, and the modeling of the network components within the WoT W3C architecture. Section 4 introduces the mash-up policies. Section 5 presents a subset of the experimental results. Conclusions and future works follow in Section VI.

2 Related Works

Since 2007, when the concept of WoT appeared in the literature, several research studies have explored how to interconnect IoT devices through standard Web technologies. This has also lead to a proliferation of WoT frameworks and architectures, which are quite different in terms of WoT-based interaction patterns supported and functional goals addressed. For instance, the authors of [6] review more than twenty WoT frameworks on the basis of twelve elements which are taken as key components of the WoT. Although the REST paradigm is considered the reference solution to implement WoT-oriented services (e.g. [7]), alternatives to the HTTP protocol have been considered: for instance, CoAP-based architectures are proposed, among others, in [8] and [9]. A generic model supporting interoperability and mash-up operations from different hubs is proposed in [4]: here, the authors warn about the proliferation of WoT tools, and advocate for the need of standard solutions.

To this purpose, the W3C WoT group started its activities on 2015 with the goal of defining a reference WoT set of standards, enabling interoperability among different IoT systems. In this paper, we refer to the W3C WoT draft presented in [10]. In brief, the W3C WoT architecture is composed of four main blocks:

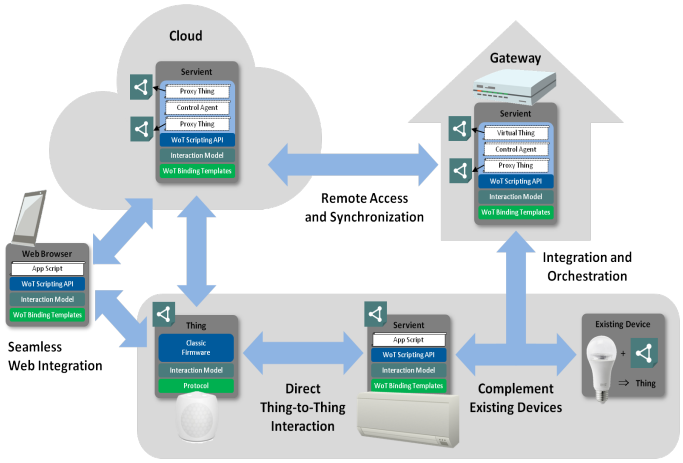


Fig. 1. The abstract architecture of W3C WoT (source: [10]). The image shows the internal blocks and the multiple interactions patterns among the Things.

- *Thing*: this is an entity that can be semantically represented. Using the W3C words: "A Thing is the abstraction of a physical or virtual entity... This entity can be a device, a logical component of a device, a local hardware component, or even a logical entity such as a location" [10].
- *WoT Thing Description (TD)*: this represents the metadata of the Thing, including its interactions, data models, communication protocols and security mechanisms. By default, the TD is serialized with the JSON-LD language and following the Properties, Actions, and Events paradigm.
- *WoT Binding Templates*: this is the metadata describing the communication strategies that the Thing is able to implement. For instance, a possible strategy could be the following: Machine-to-Machine (M2M) over the MQTT protocol, with TLS security mechanism enabled.
- *WoT Scripting API*: this is a WoT interface allowing scripts to perform main operations on a Thing, like adding properties, reading properties, or retrieving its TD.

All the blocks above are implemented within a software runtime named *Servient*, which can indifferently act as a Server or as a Client. In the first case, the Servient is said to host and *expose* Things, i.e. it takes the TD as input and creates a software object serving the requests like accessing the exposed properties, actions and events. In the second case, the Servient is said to *consume* Things, i.e. it creates a runtime resource model that allows accessing the properties, actions and events exposed by the server Thing on a remote device.

Figure 1 depicts the abstract W3C architecture for the WoT, including the blocks within each Thing and their possible interactions. In particular, the W3C working group identified a short list of interaction patterns that are general enough to cover most of the existing IoT deployments, regardless of the ap-

plication domain. The simplest one is the Client-Thing interaction, i.e. a Web application that invokes actions on a remote Thing, after having consumed it. Due to the recent appearance of the W3C WoT standard, few real-world applications and testbed have been proposed so far in the literature. A demo showing the possibility to query a W3C WoT sensor device from a mobile phone is sketched in [11]. In [12], an interesting application of the W3C WoT architecture to the automotive industry is described; more specifically, the authors illustrate how to describe the car signals data with a semantic ontology, and how to make them available to external applications through the W3C WoT interaction patterns. Security risks and vulnerabilities presented by WoT metadata are discussed in [13]. Versioning mechanisms for the TD metadata are proposed in [14]. Finally, in [15], we proposed the WoT Store, a W3C WoT-compliant framework that enables the semantic Thing discovery and the seamless distribution and execution of WoT applications. The WoT Store constitutes the natural execution environment for the mash-up applications considered in this study: we plan to explore such feature in a future work.

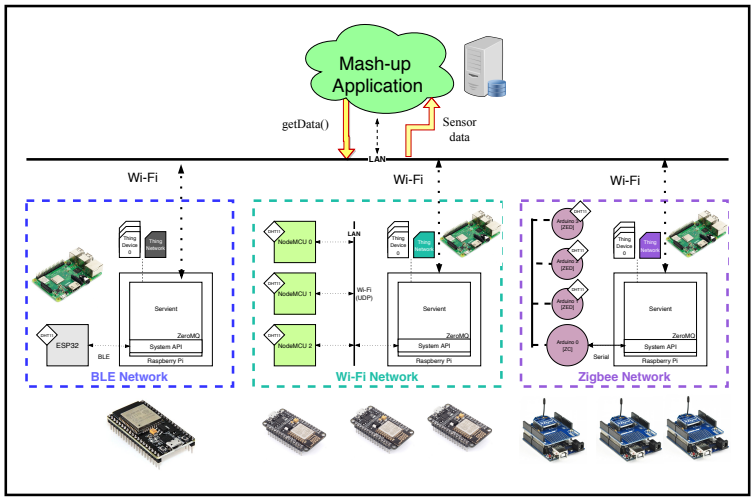


Fig. 2. The IoT/WoT monitoring system deployed in this study.

3 The W3C WoT testbed: architecture and components

The goal of this study is to investigate the suitability -both in terms of ease of deployment and of performance- of the W3C WoT architecture for Industry 4.0 applications. To this purpose, we consider a generic IoT monitoring system of a production site, characterized by the presence of heterogeneous sensors using different communication technologies. The overall architecture of the testbed, depicted in Figure 2, is structured on three tiers:

- **Edge layer**. This layer is composed of three Wireless Sensor Networks (WSNs), operating over the same environment: an IEEE 802.15.4 WSN network, a IEEE 802.11 Wi-Fi WSN network and a BLE device. The 802.15.4 network includes four devices (*Arduino Xbee* boards), with one Coordinator and three Leaf nodes equipped with sensing units (*ThinkerKit* temperature sensor). The Wi-Fi network includes three devices (two *NodeMCU* and one *Arduino WiFly* board), all provided with a direct link toward the Access Point (AP) and with a *DHT11* temperature/humidity sensor. Finally, the BLE WSN consists of one *ESP32* board, provided with a *DHT11* sensor.
- **Fog layer**. The 802.15.4 coordinator, the BLE and the Wi-Fi devices are connected to the corresponding Fog node, via USB cable links (for the BLE and the 802.15.4 Coordinator) or Wi-Fi links (for the IEEE 802.11 devices). Each fog node is constituted by a *Raspberry PI3B+* board and it is in charge of exposing the corresponding Web avatar (i.e. the Web Thing) for each managed device and WSN.
- **Processing layer**. This layer implements the logic of the monitoring system. It is constituted by a Linux server running the mash-up applications further defined in Section 4, and connected to the Fog nodes via Wi-Fi links. More specifically, the layer is in charge of: (*i*) orchestrating the sensing operations, by properly selecting the devices to query at each time slot according to the policies of Section 4; (*ii*) storing the collected data within a time-series database; (*iii*) processing and analyzing the data in order to implement the Digital Twin model of the monitored site.

In this study, for space reasons, we omit the data analytics process, and also the creation of the Digital Twin model, leaving it to future works. Instead, we detail the data retrieval operations, and specifically the way we implemented the WoT W3C components of the architecture reported in Figure 1, i.e.:

- Edge devices implement low-level communication and sensing operations in the embedded firmware. The implementation as well as the list of operations and the data format used by each device is technology dependent. This layer is part of the IoT, while it is not covered by the WoT architecture.
- Fog nodes run a W3C WoT Servient, by using the JavaScript (JS) framework available at [16]. Each Fog node exposes two types of Web Things, i.e.: multiple (*i*) *Thing Devices*, describing the properties, events and actions of physically managed edge devices, and one (*ii*) *Thing Network*, describing the overall performance of the virtual WSN composed by the list of connected Thing Devices. Moreover, we consider three possible protocol bindings for each Thing, i.e. interaction modes with the Things, based on the HTTP (default choice), the CoAP or the MQTT protocols. The System APIs are implemented in Javascript, and further structured into two layers, i.e.: (*i*) a *Device Query* level, that is in charge of issuing request-response communication with the Edge device, based on the wireless technology and the protocol stack supported by this latter (e.g. UDP socket for the WiFi devices, Serial socket for the Zigbee Coordinator, BLE connected mode for the BLE device), (*ii*) an *Inter-Process Communication* (IPC) level, that makes

the sensor data available to the upper Scripting APIs via IPC facilities (in our case, implemented in the ZeroMQ library[1]).
– Finally, the Processing node interacts with each Fog node/Servient in order to consume Things, e.g. by periodically *invoking* the `getData` *action* from the Things selected according to the actual mash-up policy.

Name	Type	Description
DeviceID	Property	Device identifier in the network.
NetworkID	Property	Network identifier the device belongs to.
Temperature	Property	Last temperature value.
State	Property	Current state of the device.
GetData	Action	Get the temperature data.
Start	Action	Start sending data at each time-slot.
Stop	Action	Stop sending data.
NewData	Event	This event is fired when a new sensor data is produced.
ChangeState	Event	This event is fired when the connection state changes.

Table 1. Example of Properties, Actions, and Events described in a Thing Description of a Device Thing.

Table 1 shows some of the properties, actions, and events described in the Thing Description (TD) for a Device Thing. The TD of a Network Thing includes only properties that are referred to the average network performance (i.e. the delay, the packet delivery ratio and the throughput) and actions that can be invoked from the entire network, like for instance the *getAllData()*. Similarly, the snippet below shows a code fragment of the mash-up application, specifically the way we query a sensor device in order to read its temperature value. We can notice that -through the WoT architecture- the mash-up application is agnostic on the wireless access technology in use, and retrieves data from heterogeneous sensors by means of a common API regardless of the WSN implementation. The rationale of the sensing applications is presented in the Listing 1.1.

Listing 1.1. Example code for discovering and invoking actions on Things.

```
let type = "http://wots.unibo.it/labWireless/testbed"
let THINGS = []
//get Thing Descriptions from the discovery service
for(var t in discovery.discoverByType(type)) {
    //Consume things
   let thing = await consumer.consumeThing(t);
   //Set http as protocol required
   thing.getClients().set('http', http_client);
   THINGS.push(thing)
}
for(var i = 0; i < lambda; i++) {
    //invoke the getData action for collecting data
    let thing = THINGS[i%THINGS.length];
    var res = await thing.actions['getData'].invoke();
}
```

[1] ZeroMQ Project Website, http://zeromq.org

4 The W3C WoT testbed: the mash-up sensing policies

We implemented multiple mash-up sensing policies, and we tested the capability of switching among them in a seamless way in Section 5. To this purpose, let D be the set of available devices, and $W(d_i)$, $\forall d_i \in D$, be the function describing the WSN type. In our testbed, $W : D \rightarrow \{WiFi, BLE, Zigbee\}$. We assume the time to be divided into discrete time-slot, i.e. $T = \{t_0, t_1, ...\}$, corresponding to sensing events when the mash-up application is issuing `getData` command toward a selected subset of the available devices. Let $t_{interval}$ be the temporal interval between two measurements, i.e. the time difference between t_{i+1} and t_i, assumed constant. Moreover, let $\kappa : D \times T \rightarrow \{0, 1\}$ the function indicating whether device d_i is active, i.e. it is used at time slot t_j (in this case, $\kappa(d_i, t_j) = 1$, otherwise $\kappa(d_i, t_j) = 0$). All sensing policies share a common rationale, i.e.: they keep the area covered higher than a predefined threshold, while maximizing a performance index I. In our case, the area coverage is expressed in terms of number of active devices (M) at each time-slot. More formally, all policies address the optimization problem formally defined below:

$$\begin{aligned} \text{Goal} &: \text{Maximize } I \\ \text{Constraint} &: \sum_{d_i \in D} \kappa(d_i, t_j) = M, \forall t_j \in T \end{aligned} \tag{1}$$

The performance I can vary according to sensing policy in use. We implemented and tested four different metrics:

- Static *Energy-aware* policy (P_0). The mash-up application selects the M active devices at each time-slot according to a pure round-robin scheme, in order to discharge them with the same rate.
- Dynamic *Delay-aware* policy (P_1). The mash-up application takes into account the average delay required to issue a `getData` command and to receive the corresponding reply message. The M devices with the lowest Round Trip Time (RTT) are selected at each time slot.
- Dynamic *PDR-aware* policy (P_2). The mash-up application takes into account the communication reliability of each sensor expressed in terms of average Packet Delivery Ratio (PDR), i.e. the ratio of received replies over the total number of `getData` requests sent toward each d_i. Specifically, the M devices with the highest PDR values are selected at each time slot.
- Dynamic *Delay-PDR-aware* policy (P_3). The mash-up application takes into account both the delay and the PDR, as better explained in the following.

Excluding P_0, all the other policies compute the M sensors to query at each time-slot based on the current traffic loads and network conditions. For this reason, we employ a dynamic, learning-based scheme based on the Reinforcement Learning (RL) framework [2]. In brief, this latter refers to a class of machine learning algorithms where an agent learns over time the optimal sequence of actions

[2] For space shortage, we do not provide an in-depth illustration of the RL framework. Interested readers can refer to [17] for a detailed discussion on the topic.

needed to perform a task, by dynamically interacting with the environment and by receiving a numeric reward at each interaction. More formally, the RL framework can be represented as a Markov Discrete Process (MDP) $< S, A, R, TR >$ where: S is the set of States, A is the set of Actions, $R : \{S, A\} \rightarrow \mathbb{R}$ is the Reward function, expressing a numeric reward received by the agent when executing action $a_j \in A$ in state $s_i \in S$, and $TR : \{S, A\} \rightarrow S$ is the transition function, expressing the next state s_j after performing action a_j from state s_i (a deterministic environment is assumed). The goal of the RL agent is hence to determine the optimal policy function $\tau : S \rightarrow A$ that indicates the optimal action to execute at each state, so that the long-term reward is maximized. In our modeling, we omit the state function S, while the list of action A coincides with the list of devices D. The immediate reward $R(d_i)$ is computed when issuing a `getData` command on sensor d_i, according to the policy in use:

- P_1: this is the RTT for each `getData` command. Only successful requests (i.e. reply messages are received) are considered.
- P_2: this is a positive value (+1) if the `getData` is successful, 0 otherwise.
- P_3: similarly to P_1, however a penalty equal to $t_{timeout}$ is applied in case no reply is sent back after a timeout.

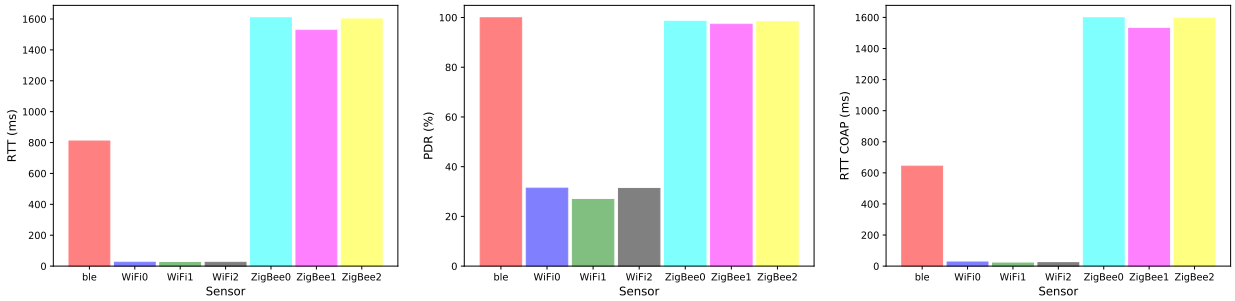
Each time a `getData` is issued on d_i, and the immediate reward $R(d_i)$ is computed, we also update the Q-value entry at time slot t for d_i as follows:

$$Q_t(d_i) = Q_{t-1}(d_i) + \alpha \cdot (R(d_i) - Q_{t-1}(d_i)) \tag{2}$$

where α is a learning rate, set equal to 0.7 in our experiments. Balancing the exploration and exploitation issue is a crucial issue in dynamic environments [17]. For this reason, we consider an ϵ-greedy exploration scheme, i.e.: each time a `getData` is executed, the policy selects with probability $1 - \epsilon$ the sensor with the k-th highest Q-value, and it performs a random selection over D otherwise (avoiding duplicates). We repeat the ϵ-greedy selection M times at each time slot, since all policies need to guarantee an M-coverage of the scenario (in other words, the k above varies between 0 and $M-1$). The ϵ parameter is progressively discounted at each time slot, i.e. $\epsilon_t = \epsilon_{t-1} \cdot \psi$, with $0 < \psi < 1$, in order to reduce the exploration over time. At the same time, the ϵ parameter cannot decrease below a minimal threshold (ϵ_{min}), i.e. a default exploration rate is kept anyway in order to detect any possible change in the scenario, and to adapt the system policy accordingly. We set ϵ=0.8, ψ=0.97, ϵ_{min}=0.1 in our testbed.

5 The W3C WoT testbed: experimental results

In this Section, we report a subset of experimental results collected through the WoT testbed described above. The experimental analysis is divided in three stages: (*i*) first, we characterize the overall performance of different WSNs and sensors; (*ii*) second, we evaluate the four different mash-up policies of Section 4; (*ii*) finally, we demonstrate the possibility of dynamic mash-up policy replacement and quantify the overhead introduced by the W3C WoT architecture.

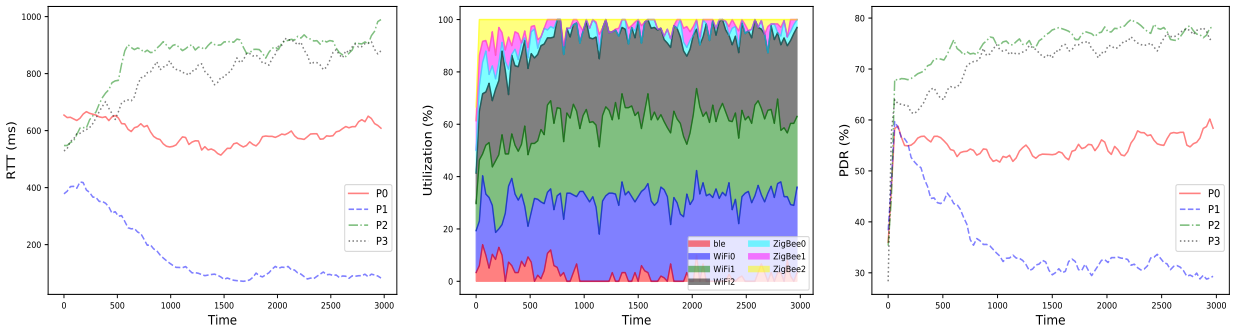


(a) (b) (c)

Fig. 3. The average per-device RTT and PDR is shown in Figures 3(a) and Figure 3(b), respectively. The per-device RTT for the CoAP protocol is shown in Figure 3(c).

Figures 3(a), 3(b) and 3(c) refer to the first analysis. Specifically, Figure 3(a) and 3(b) show respectively the average RTT and PDR for each device and WSN type, when the HTTP protocol is used to interact with each Web Thing. It is easy to notice that the Wi-Fi devices are producing the lowest RTT values. The PDR original results demonstrated that the Wi-Fi WSN is also the most reliable technology. However, in order to differentiate the mash-up policies, we introduced a probabilistic packet filter on the Wi-Fi Serviant, discarding the sensor data messages with a loss rate equal to 70% to emulate a congested access point. As a result, comparing Figures 3(a) and 3(b), we can notice that the sets of M=3 nodes maximizing the RTT or the PDR depends on the selected performance index. Finally, Figure 3(c) shows the per-device RTT when the CoAP protocol is used for data gathering. Only minimal differences can be noticed compared to the HTTP case (Figure 3(a)).

In Figures 4(a)-5(a), we evaluate the performance of different mash-up policies. Figure 4(a) shows the RTT values of P_0, P_1, P_2, P_3 algorithms over time-slots; as expected, P_1 produces the lowest delay since it takes into account the per-packet RTT as immediate reward. Also, we can appreciate the learning phases of P_1: the RTT is high during the exploration phase and it is progressively reduced when increasing the amount of exploitation. After time-slot 1000, the RL algorithm has discovered the optimal set of sensors, however it keeps performing random actions for continuous, minimal exploration. This justifies the jagged shape of the plot. In Figure 4(b) we depict the per-device ratio of utilization over time for the policy P_1. While during exploration all the devices are equally used, after time-slot 1000 the mash-up policy is mostly exploiting the three Wi-Fi devices since -in accordance with Figure 3(a)- they are associated to the lowest RTT values. Figure 4(c) compares the policies in terms of PDR. Here, the optimal policy is P_2; form Figure 5(a) we can notice that, after the exploration phase, the three Zigbee devices are maximally used, hence conversely to Figure 4(b) but again in accordance with Figure 3(b). We tested the dynamic policy replacement in Figure 5(b); i.e. from time-slot 1 to 3000, policy P_1 is used



(a) (b) (c)

Fig. 4. The RTT an PDR values for the four mash-up policies are shown in Figures 4(a) and 4(c). The device utilization ratio for the P_1 policy is shown in Figure 4(b).

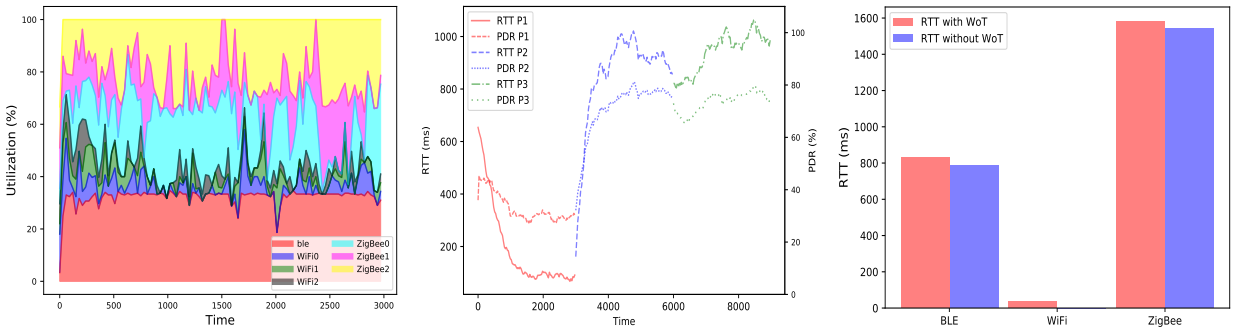
(delay minimization), then P_2 from 3001 to 6000 (PDR maximization), finally we switch to P_3 (delay-PDR trade-off) from instant 6001. We remark that the policy replacement is simply implemented as the shut-down of a JS process and the execution of a new one, thanks to the abstraction provided by the W3C WoT architecture; no hardware or software re-configuration of the WSNs is required. Finally, we evaluate in Figure 5(c) the overhead introduced by the W3C WoT deployment, and specifically by the WoT servient: to this aim, we compute the RTT required to perform a sensor request directly at the System API level. We can notice that most of the overhead is due to the channel access and the processing at the firmware level, while the overhead introduced by the Servient and by the additional communication with the Web Thing is negligible.

6 Conclusions and Future Works

In this paper, we have described the deployment and evaluation of a W3C-based WoT system for generic site monitoring applications. In order to stress the interoperability issue, we have realized a testbed composed of three heterogeneous WSNs and mash-up applications orchestrating the sensing operations over the monitored area. We have discussed how the different WoT components have been instantiated on the target scenario, and we have demonstrated the capability of decoupling the sensing policy from the low-level networking operations, thanks to the Thing meta-data description. Future works include: the digital twin model implementation, the evaluation of additional mash-up policies, the integration with the WoT Store [15] framework.

References

1. E. Sisinni, A. Saifullah, et al. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics*, 2018.



(a) (b) (c)

Fig. 5. The device utilization ratio for the P_2 policy is shown in Figure 5(a). The RTT and PDR values when replacing the active policy at runtime in shown in Figure 5(b). The RTT when enabling/disabling the WoT approach is shown in Figure 5(c).

2. P. Patel, et al. From raw data to smart manufacturing: AI and semantic Web of Things for Industry 4.0. *IEEE Intelligent Systems*, 33(4), pp. 79-86, 2018.
3. McKInsey Global Institute. The Internet of Things: Mapping the value beyond the hype. Executive Summary. 2015.
4. M. Blackstock and R. Lea Toward interoperability in a Web of Things. *Proc. of IEEE UbiComp*, Zurich, Switzerland, 2013.
5. D. Guinard and V. Trifa Building the Web of Things. *Manning Editions*, 2016.
6. A. Kamilaris and M. I. Ali. Do "Web of Things platforms" truly follow the Web of Things?. *Proc. of IEEE WF-IoT*, Reston, USA, 2016.
7. F. Paganelli, et al. A Web of Things framework for RESTful applications and its experimentation in a smart city. *IEEE Systems*, 10(4), pp. 1412-1423, 2016.
8. L. Mainetti, V. Mighali and L. Patrono. A software architecture enabling the Web of Things. *IEEE IoT Journal*, 2(6), pp. 445-454, 2015.
9. E. Mingozzi, G. Tanganelli and C. Vallati. CoAP proxy virtualization for the Web of Things. *Proc. of IEEE CloudCom*, Singapore, 2014.
10. WoT W3C Architecture. http://www.w3.org/TR/wot-architecture/
11. Y. Ji, K. Ok and W. Suk Choi. Demo Abstract: Web of Things based IoT standard interworking test case. *Proc. of ACM BuildSys*, Shenzen, China, 2018.
12. B. Klotz, S. K. Datta, D. Wilms, et al. A car as a semantic Web Thing: motivation and demonstration. *Proc. of IEEE GIoTS*, Bilbao, Spain, 2018.
13. M. McCool and E. Reshetova. Distributed Security risks and opportunities in the W3C Web of Things. *Proc. of IEEE DISS*, San Diego, USA, 2018.
14. M. Blank, S. Kaebisch, H. Lahbaiel, and H. Kosch Role models and lifecycles in IoT and their impact on the W3C WoT Thing Description. *Proc. of IEEE IoT*, Santa Barbara, USA, 2018.
15. L. Sciullo, C. Aguzzi, M. Di Felice, T. S. Cinotti WoT Store: Enabling things and applications discovery for the W3C Web of Things. *Proc. of IEEE CCNC*, Las Vegas, USA, 2019.
16. Eclipse ThingWeb https://projects.eclipse.org/proposals/eclipse-thingweb
17. A. Barto and R. S. Sutton Reinforcement Learning: An Introduction MIT Press, 1998