Assessment of Different OPC UA Implementations for Industrial IoT-based Measurement Applications

Alberto Morato, Stefano Vitturi, Federico Tramarin and Angelo Cenedese

***Abstract*—The Industrial IoT (IIoT) paradigm represents an attractive opportunity for new generation measurement applications, which are increasingly based on efficient and reliable communication systems to allow the extensive availability of continuous data from instruments and/or sensors, thus enabling real-time measurement analysis. Nevertheless, different communication systems and heterogeneous sensors and acquisition systems may be found in an IIoT-enabled measurement application, so that solutions need to be defined to tackle the issue of seamless, effective, and low-latency interoperability. A significant and appropriate solution is the Open Platform Communications (OPC) Unified Architecture (UA) protocol, thanks to its object–oriented structure that allows a complete contextualization of the information. The intrinsic complexity of OPC UA, however, imposes a meaningful performance assessment to evaluate its suitability in the aforementioned context. To this aim, the paper presents the design of a general yet accurate and reproducible measurement setup that will be exploited to assess the performance of the main open source implementations of OPC UA. The final goal of this work is to provide a characterization of the impact of this protocol stack in a IIoT-enabled Measurement System, in particular in terms of the latency introduced in the measurement process and the power consumption.**

***Index Terms*—Industrial Internet of Things, Distributed sensors, OPC UA, Latency assessment, Performance evaluation, Distributed measurement applications**

I. Introduction

In the last few years, the industrial world embraced the Industry 4.0 paradigm [1], which merges technologies with products, systems, and services, having its own intrinsic networked structures, to realize the Industrial Internet of Things (IIoT) [2], [3]. IIoT is a network of networks that connect industrial equipment, controllers, sensors and actuators, i.e. the "Things", to provide diverse and advanced types of services in manufacturing systems, aiming at improving quality, productivity, efficiency, reliability, safety, and security.

Traditionally, the foundation of manufacturing and process industries has been in the deployment of specific distributed sensors systems, to the purpose of monitoring (and then controlling) the production process, thus leveraging the concept of Distributed Measurements Systems (DMS) [4]. With the increased pervasiveness of the IIoT paradigm, DMSs come even more into focus, since the components of an IIoT systems need an even further level of interaction to integrate instrumentation data, sensors, communication and processing. Moreover, the need of improving production capacity and optimizing the output process, eventually exploiting predictive maintenance and machine learning approaches [5], requires an increased number of sensor devices and sensor swarms to be deployed.

This new IIoT–enabled DMS scenario is based on the support of efficient and reliable communication systems, which have to ensure widespread availability of data gathered from possibly heterogeneous measurement instruments and/or sensors [6], [7]. Overall, the IIoT paradigm may represent the enabler for several enhanced measurement features: continuous and thorough measurements through low-power wireless connections, measurement collection over considerably wide geographic areas, real-time analysis of measurement data collected from the field [8].

Unfortunately, in the highlighted IIoT scenario it is common that components and sensors devices come from different producers and use different formats to represent measurement data, and also it is very likely that they intrinsically operate over heterogeneous networks. Hence, it is of significant importance the provision of ways to enable communication and interoperability among such devices. A key solution toward this goal is the Open Platform Communication (OPC) Unified Architecture (UA) [9]. OPC UA is a protocol defined by the IEC 62541 international standard [10] conceived to implement Machine–to–Machine (M2M) communication over possibly different physical media, while ensuring high level data protection against attacks and threats.

OPC UA represents hence an appealing and advantageous opportunity for the arising IIoT measurement paradigm. Particularly, its object–oriented structure allows a complete contextualization of the information. For instance, an object could be used to store the value of a measurement, the features of the instrument/sensor, the measurement units, possible thresholds and so on. Such important characteristics allow for new generation of measurement instruments to deal with multiple and heterogeneous types of data.

At the same time, the complexity of the OPC UA protocol may also reveal an obstacle to its introduction within measurement systems. Indeed, sensors and actuators, field equipment and measurement instruments in the IIoT scenario are typically realized exploiting devices with limited hardware resources, low computational capabilities (and low costs). As a consequence, the implementation of OPC UA on such devices might be problematic and, furthermore, the performance might result compromised in terms of an increased latency and power consumption, hence impairing the quality and accuracy of measurements.

A. Morato and A. Cenedese are with the Dept. of Information Engineering, University of Padova, Italy.

S. Vitturi is with the CNR–IEIIT, National Research Council of Italy.

F. Tramarin is with the Dept. of Engineering E. Ferrari, University of Modena and Reggio Emilia, Italy.

Motivated by the above considerations, in this paper, which substantially extends [11], we consider IIoT measurement systems based on heterogeneous networks, and specifically we tackle the problem of the assessment of a promising enabling technology. Particularly, we carried out an experimental campaign on three popular open source implementations of the OPC UA protocol stack, namely Open62541, FreeOPC UA C++ and FreeOPC UA Python. The campaign is aimed at investigating the behavior of OPC UA for these three different implementations focusing on i) CPU usage, ii) communication times and iii) power consumption. In order to provide a meaningful and fair assessment, the protocol was implemented on a widespread commercially available Raspberry Pi Model 3B+ board that, thanks to its features, represents a manageable and effective test–bed. The measurement set–up has been designed to be independent from the specific protocol stack and, moreover, to be of general usage and reproducible.

In detail, the paper is organised as follows. Section II provides an essential related work about the adoption of OPC UA in the context of distributed measurement systems. Section III gives a brief description of the OPC UA protocol. Also, this Section outlines the possible structure of distributed measurement systems that rely on OPC UA. Section IV introduces the experimental set–up we built up for the measurement campaign. Section V describes the tests carried out and discusses the obtained results. Finally, Section VI concludes the paper and outlines some future directions of research.

II. Related Work

The introduction of IIoT technologies in the context of distributed measurement systems has started to be addressed some years ago. Paper [8] deals with the potential of IIoT for the instrumentation and measurement fields. Notably, the authors provide an accurate assessment that addresses benefits and challenges, including also some useful commercial aspects. In [12], the authors consider the adoption of the widespread Low Power Wide Area Networks (LPWAN) to enable IIoT-based measurement and monitoring applications over large distances. They focused specifically on LoRaWAN and interestingly, the measurements carried out revealed that some commercial devices are able to deliver frame transmission with acceptable uncertainty (below 3 μs). Also, such devices showed a good stability of the clock over time. This allowed the authors to conclude that LoRaWAN networks represent an opportunity for DMS applications such as smart metering, building monitoring and process industry. LoRaWAN is also addressed in [13], where the authors consider the implementation of mesh configurations to implement a distributed measurement application that collects measurement data from sensors distributed over large geographical areas. A different LPWAN, namely, Narrow–Band–IoT (NB–IoT) is considered in [14]. The authors present an access scheme suitable for multiusers with different variable rates. Also, they propose a methodology for real–time analysis of such configurations. Both papers [15] and [16] make use of Wi-Fi to implement measurement systems. In particular, [15] describes a potentiostat for electrochemical biosensors able to transmit measurements to a cloud system via the Wi-Fi interface it uses. A similar communication technique is used in [16], which implements a pH monitoring system. Notably, the examples of data storage and analysis provided in both the papers are based on the same open source cloud service, namely ThingSpeak. Communication aspects of IIoT based measurement systems are addressed also in [17], that presents different implementations of wireless sensors based on both the TCP and UDP protocols (over Wi-Fi) and Bluetooth. A further interesting application of IIoT to measurement systems is described in [18]. The authors propose an Internet of Things system designed to monitor environmental variables gathered from suitable measurement sensors, equipped with Bluetooth Low Energy (BLE) wireless interfaces. The system showed its effectiveness ensuring the transmission of measurements from battery–powered sensors to a cloud system for long period of times.

Moving to OPC UA, in [19] the authors describe a method to achieve synchronization among electrical drives connected via EtherCAT (a widespread real–time Ethernet network) using the OPC UA protocol. Particularly, the paper deals with the significant topic of obtaining a high accuracy synchronization over a geographically distributed system, that is of uttermost importance is distributed measurement applications. In [20], the authors refer to metrology assisted assembly systems, and introduce the optical Large–scale metrology instruments, such as laser trackers and indoor GPS. Then they propose an object–oriented model to formally describe such instruments and investigate the suitability of OPC UA, as well as that of other protocols, to implement such a model. In [21] OPC UA is used to implement a smart sensor system to monitor the behavior of numerical control devices in the Industry 4.0 context. Specifically, OPC UA objects are used to store sensor information such as measurement, threshold, range, product data, etc. Another interesting OPC UA application is proposed in [22]. Here the authors present a system based on the use of the OPC UA protocol to connect and integrate components typical of industrial automation as well as of distributed measurement systems. Examples of applications are provided that include smart microgrids, industrial laboratories and energy systems in general. Finally, paper [23] considers an IIoT environment and focuses on the transfer of plant data to the cloud when OPC UA–based gateways are used to gather data directly at the production level. Notably, the authors implemented a measurement system that allowed to determine the impact of Quality of Service parameters on the communication delays.

III. Brief Introduction to OPC UA

OPC UA is based on the Client–Server model, where the server is the source of information, which in turn is structured in objects, formally referred to as “Nodes”. The OPC UA model defines the Node objects in term of variables, methods and events. A Node is, hence, the fundamental entity of OPC UA and it represents a basic object which has only the attributes necessary to define any kind of information item

(e.g. ID, name, etc.). The Server provides to each attached Client with a set of Services (e.g. read, write, browse, etc.) which can be used to access the information stored on the server itself. The set of nodes made available by an OPC UA server is referred to as the address space [24].

In the context of IIoT-based measurement applications the OPC UA protocol can be profitably exploited to allow seamless and secure interoperability both among the heterogeneous sources of measurement data and among the heterogeneous communication networks. Indeed, measurements can be stored on nodes that belong to one or more servers, so that they can be seamlessly accessed by distributed clients. An illustrative sketch representing the described scenario is reported in Figure 1. As can be seen, measurements stored in different devices, and structured within diverse OPC UA servers, can be remotely accessed by an OPC UA client which provides for their visualization.

IV. Experimental Set–up

The experimental set–up has been redesigned, with respect to [11], to be as much general as possible, with reproducibility in mind in order to be easily reused in different scenarios. In our specific context, the set–up has been used to evaluate the performance figures of three different OPC UA protocol stacks implemented on lightweight embedded systems, that resemble those adopted by intelligent IoT sensors. Particularly, experiments have been carried out using Raspberry Pi Model 3B+ boards, on two different communication systems, namely Ethernet and Wi-Fi, as shown in Fig. 2. Indeed, while both networks are meaningful in distributed measurement systems and IIoT scenarios, the former is much more targeted to high performance, ultra low-latency and local measurement applications, whereas the latter represents its wireless counterpart, allowing for increased mobility and larger installations. More importantly, although Wi-Fi networks are able to provide very high transmission rate and good reliability, the performance figures they provide are clearly different with respect to those

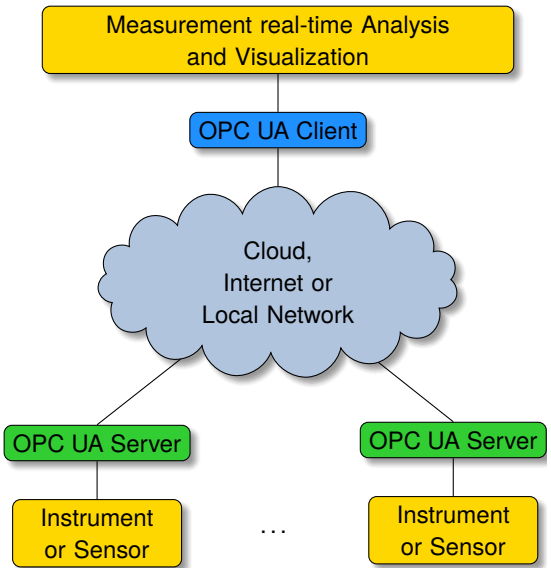


Fig. 1. Example of use of OPC UA in an IIoT-based Measurement System

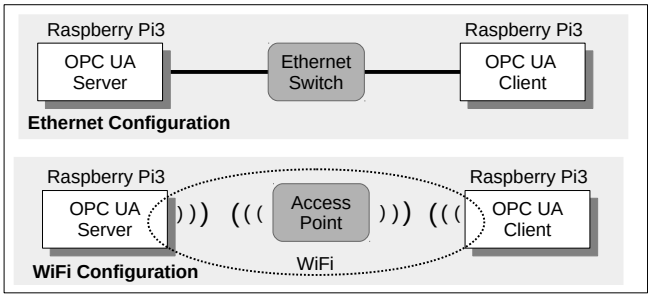


Fig. 2. Experimental set–up

of Ethernet networks, that may considered hence as a benchmark. An important aspect that will be analyzed in this work is hence the comparison between the two network supports in terms of the latency to gather measurement data, which is a crucial parameter for accurate distributed measurements.

As can be seen, the same topology was used for the two networks, with OPC UA client and server implemented on two diverse Raspberry Pi boards. A Netgear WGR 614 Wireless-G router was used to connect the 2 boards that was able to seamlessly implement the packet routing for both the Ethernet and Wi-Fi configurations.

Two different operating system versions have been used for the boards. The first one is represented by the default Raspbian OS (Kernel version 4.14.79), whereas the second one is its real-time version (Kernel version 4.14.74–rt44). The latter one has been obtained from the default kernel version with the introduction of the `RT_PREEMPT` patch set, which enables a real–time behavior of the system allowing non critical parts of the kernel to be preempted in favor of the execution of userspace applications. Furthermore, we exploited a useful feature offered by the Linux operating system, to isolate a group of CPU cores in which a process can be run. Specifically, the `isolcpus` boot parameter in combination with the `taskset` command, allows to isolate one or more cores from the kernel scheduling and to reserve them for the execution of userspace applications without interference from the OS. Furthermore, in order to minimize the external factors that may affect the accuracy of the measurements, the CPU governor (i.e. a kernel–level component responsible of scaling the CPU frequency based on the workload, as will be described in the next Subsection) has been disabled during the experiments and the CPU frequency has been set to its maximum operable value of 1.4 GHz.

The open-source implementations of OPC UA considered in the experiments are reported in Table I, along with the indication of the adopted programming language (PL). In order to ensure reproducibility of the experiments, we also provide the commit hash of the sources (all the implementations are available through the popular Github platform) at the time the experiments have been performed:

All the listed protocol stacks work natively on Raspberry Pi boards, and consequently their setup procedures have not involved any further software adaptation. However, they are conceptually different. In particular, as can be seen, two out of three stacks are implemented by means of compiled languages (C/C++), whereas the other one is implemented in Python,

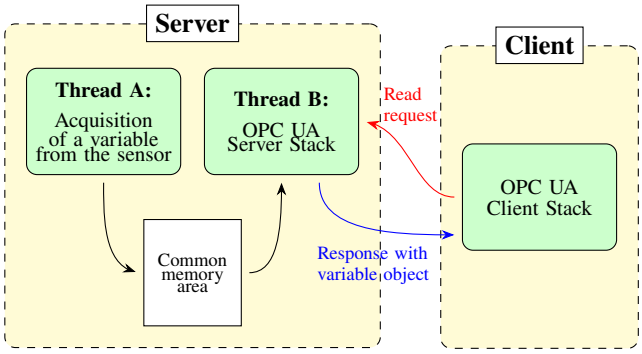


Fig. 3. Test OPC UA Task

which is a high level interpreted language. As a consequence the analysis of their behaviors reveals necessary to provide useful insights for the applications that use them. In this direction, since the outcomes of the experiments also depend on the adopted development environment, the most relevant technical details are summarized below:

- Python version 3.5.3;
- glibc 2.23;
- gcc version 6.3.0;
- gcc optimization option: -O3 -s.

The experiments we carried out were based on a purposely developed test OPC UA task, with which an integer variable stored in the OPC UA Server is read by the OPC UA Client. In this task, the server implements two separate threads as shown in Figure 3. Thread A simulates the acquisition of a new measurement (i.e. a physical quantity) every second, by increasing an integer variable. Thread B is instead devised to manage the whole OPC UA server. The measurement outcome, stored in an OPC UA object, is saved in a memory area common to both threads so that the server can access it. In the test task, the OPC UA Client cyclically reads the value of the variable stored on the Server. This is accomplished by a read request issued by the Client, to which the server answers in agreement with the OPC UA protocol rules.

V. Measurement Results and Analysis

The objective of the measurements is to assess the behavior of the three different OPC UA implementations focusing on performance figures that are of interest for IIoT-based measurement instruments and applications. Specifically, we addressed the CPU usage, the power consumption and the task execution times. The latter indicator is of particular significance in the application context of this paper, since it has been defined (see Sec. V-B) to represent the overall latency with which measurement data can be collected at the client, and is hence a meaningful indicator of the intrinsic capability of the system to sustain real-time measurement analysis over networks [8].

TABLE I
OPC UA Implementations

OPC UA Implementation	Programming Language	Commit Hash
Open62541	subsets of C99 and C++98	9f0c73d
FreeOPC UA C++	C++11	da2b76f
FreeOPC UA Python	Python	83fb9ea

TABLE II
Statistics of the CPU Usage

	CPU Usage		
	Mean	In Kernel Space	In User Space
Open62541	17.2%	60.89%	39.11%
FreeOPC UA C++	26.1%	50.63%	49.37%
FreeOPC UA Python	51.2%	–	–

A. CPU Usage

A first set of outcomes is resumed in Table II, which shows the statistics about the CPU usage for the three considered implementations.

As can be seen, Open62541 is the most efficient from the average resources utilization point of view, followed by FreeOPC UA C++ and FreeOPC UA Python. Actually, the latter one highlighted a rather higher utilization compared to the other ones. However, this is not surprising since FreeOPC UA Python is based on an interpreted language, being certainly less efficient. It is interesting to note that, for the Open62541 implementation, the subdivision of the used resources of the CPU is slightly unbalanced towards the Kernel Space, while for FreeOPC UA C++ we have a subdivision almost at 50%. Unfortunately, values for the FreeOPC UA Python implementation are not available, because the tool `Perf`, with which the analysis was performed, does not support measurements of the stack of interpreted languages.

A second set of outcomes is focused on the CPU usage during the execution of the OPC UA test task. They are reported in Table III. Specifically, the table refers to context switches, CPU migrations and total number of CPU cycles to complete the execution of 100.000 consecutive OPC UA test tasks. These outcomes are common indices exploited to determine the efficiency of a program, where high values indicate poor optimization and therefore long execution times. The results are in good agreement with the former observations. Also in this case, both the compiled implementations have comparable values, whereas the Python based one shows much higher values regarding in particular the number of CPU cycles and migrations.

TABLE III
CPU usage for the OPC UA test task

	context switches	CPU migrations	CPU cycles
Open62541	$100 \cdot 10^3$	1	$7.7 \cdot 10^9$
FreeOPC UA C++	$200 \cdot 10^3$	0	$13 \cdot 10^9$
FreeOPC UA Python	$283 \cdot 10^3$	29	$533 \cdot 10^9$

B. Task Execution Time

The *task execution time*, T_s, is defined in this context as the time necessary to complete one instance of the OPC UA test task described in Figure 3. Specifically, T_s represents the

time that elapses between the read request of the client, T_{req}, and the time at which it actually receives the OPC UA object containing the variable, T_{res}.

$$T_s = T_{res} - T_{req} \quad (1)$$

In the experiments, T_s has been measured by a direct access to the content of the *Cycle Counter Register* (CCR) internal CPU register implemented within ARM processors, which is a counter of the processor clock cycles. This design choice is significant to improve the accuracy relevant to the measurement of the task execution time, because accessing the CCR register requires only one CPU cycle [25], hence introducing a negligible impact on the evaluation of the time T_s. The OPC UA test task was run continuously, meaning that a new instance of the task was started immediately after the conclusion of the former one. In the Ethernet configuration, the selected transmission rate was 100 Mbit/s whereas, for the Wi-Fi one, we chose the IEEE 802.11g mode, with transmission rate dynamically selected by the multi–rate support feature provided by such protocol[1]. For each experimental session, $N = 100.000$ measurements of the execution time have been collected and analyzed. Furthermore, all the components of the experimental set–up were located sufficiently close to each other to ensure, particularly for the Wi-Fi configuration, a high success probability in packet delivery. This has been subsequently confirmed by the traffic analysis we carried out, that showed a very low number of packet retransmissions and losses, with an average of about 3.6%.

The statistics of the execution time for the OPC UA test task are reported in Table IV for the case of non – isolated CPU and, respectively, in Table V for the isolated one.

TABLE IV
STATISTICS OF THE EXECUTION TIME FOR THE OPC UA TEST TASK – NON–ISOLATED CPU

		Execution Time T_s [µs]			
		Generic OS		RT OS	
		Mean	Std	Mean	Std
Ethernet					
	Open62541	312.83	12.56	382.48	24.44
	FreeOPC UA C++	377.30	4.54	467.59	15.01
	FreeOPC UA Python	736.79	7.44	778.43	20.63
Wi-Fi					
	Open62541	2036.12	501.60	3765.62	924.52
	FreeOPC UA C++	2063.38	488.10	3869.62	907.36
	FreeOPC UA Python	9274.24	750.59	12463.11	3807.16

At a first glance, the beneficial effect of introducing CPU isolation appears evident. Indeed, as shown in Table V, all mean and standard deviation values decrease when such a feature is used. This is confirmed by the behaviors of the empirical probability density functions, reported for the Ethernet configuration, respectively, in Fig. 4 (generic operating system) and in Fig. 5 (RT operating system). As can be seen, the CPU isolation improves the EPDF shapes in all cases.

The tests carried out on the Wi-Fi configuration revealed similar behaviors, as shown in both Table IV and Table V and in both Fig. 6 and Fig. 7.

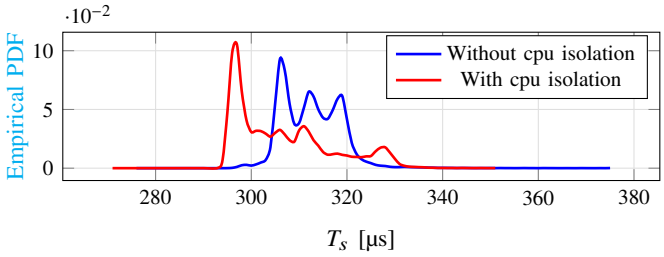
[1]Please note that the multi–rate support feature could not be disabled on Raspberry Pi boards.

TABLE V
STATISTICS OF THE EXECUTION TIME FOR THE OPC UA TEST TASK – ISOLATED CPU

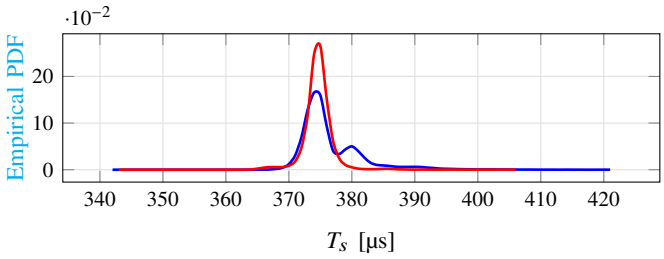
		Execution Time T_s [µs]			
		Generic OS		RT OS	
		Mean	Std	Mean	Std
Ethernet					
	Open62541	306.67	12.76	368.78	10.13
	FreeOPC UA C++	374.74	3.32	457.60	11.32
	FreeOPC UA Python	711.27	6.52	735.84	9.69
Wi-Fi					
	Open62541	1950.29	460.55	3431.68	886.28
	FreeOPC UA C++	2017.71	457.51	3656.67	902.52
	FreeOPC UA Python	9031.62	713.10	12824.94	3901.25

The introduction of the RT operating system, conversely, worsened the behavior of the OPC UA test task execution time, as can be evinced from the statistics as well as the EPDF reported above. Indeed, mean and standard deviation values are higher, for all the OPC UA implementations, with respect to the correspondent cases in which the generic OS is used. Although these results may seem surprising, they may be explained making some considerations about the introduction of the Linux real–time extension. Actually, as can be seen from Table II, all the considered implementations of the OPC UA protocol stack make extensive use of the kernel functions, especially those concerning network connectivity. Nonetheless, the real-time patch makes some parts of the kernel preemptible, thus leaving up more space for executing instructions in the user space. Thus, the stack execution could be interrupted more frequently, resulting in longer execution times and greater jitter. Furthermore, as reported in [26], the application of the Linux real–time extension has negative effects on the throughput of the communication interface. This is confirmed by the traffic analysis that showed, in the worst case, an increment of 2.8 times of packet retransmissions.

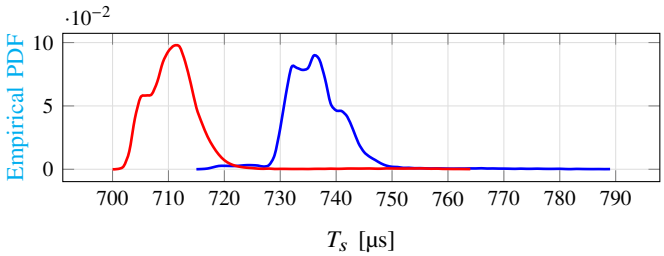
Focusing on the stack implementations, the obtained results show that both the compiled versions, Open62541 and FreeOPC UA C++, are characterized by comparable average values of the OPC UA test task execution time. Conversely, the average T_s is much higher (about doubled) for the FreeOPC UA Python implementation, as it was expected since Python is an interpreted language. This aspect is exacerbated for the Wi-Fi configuration. As far as the standard deviation is concerned, with the Ethernet configuration, Open62541 presents higher values than both FreeOPC UA C++ and FreeOPC UA Python, especially as percentage of the mean, reflecting in a considerable jitter of the execution time. This feature is evident for the generic operating system, whereas it appears more vague for the RT operating system, likely due to the additional randomness introduced by this latter one. A similar consideration can be made for the Wi-Fi configuration. In this case, as can be seen, both the mean and standard deviation are increased with respect to Ethernet. However, the standard deviation values become comparable, especially for Open62541 and FreeOPC UA C++, likely as an effect of the randomness in accessing the physical medium introduced by Wi-Fi.



(a) open62541

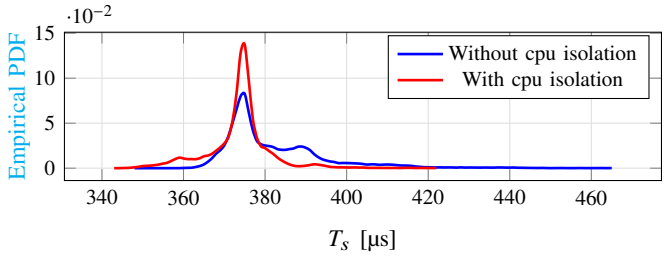


(b) FreeOPC UA C++

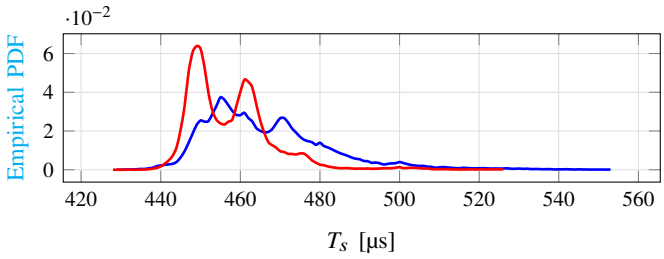


(c) FreeOPC UA Python

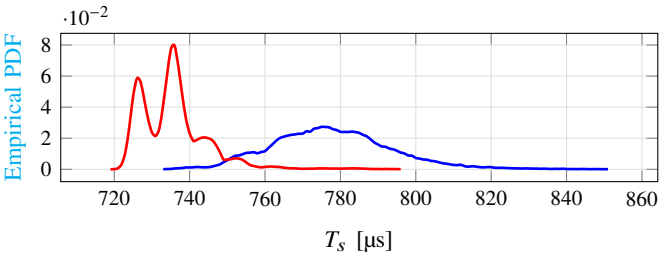
Fig. 4. Generic OS: EPDF of the execution time of the OPC UA test task for the Ethernet configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, where both server and client are forced to run on the isolated CPU.



(a) open62541



(b) FreeOPC UA C++



(c) FreeOPC UA Python

Fig. 5. RT OS: EPDF of the execution time of the OPC UA test task for the Ethernet configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, and where both server and client are forced to run on the isolated CPU.

C. Power consumption

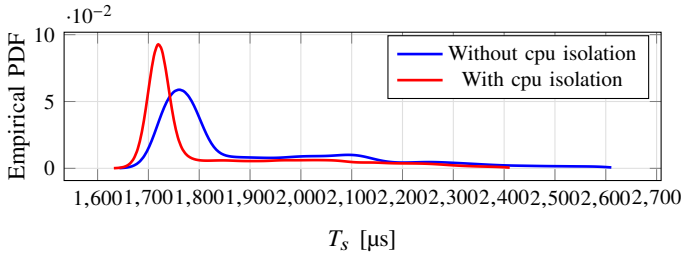
One of the main issues concerned with, possibly mobile, battery powered devices is the autonomy. Indeed, such devices have to ensure a good level of performance for a given amount of time. To meet these requirements, modern processors are capable of Dynamic Voltage and Frequency Scaling (DVFS) to minimize their power consumption and, consequently, extend the battery lifetime [27]. In the Raspberry PI boards used in the experimental setup, the DVFS functionality is driven by a default kernel governor, called `ondemand`, that dynamically adjusts the CPU frequency in agreement with the workload variation. Specifically, if the workload exceeds a predefined threshold for a certain amount of time, then the governor increases the CPU operating frequency to its maximum value. Conversely, if the workload is below the threshold, the operating frequency is switched to the lowest feasible one [28]. Such an approach, clearly, represents an optimal trade-off between performance and power consumption in a generic processing system.

We carried out an analysis of the DVFS impact on both power consumption and performance for the experimental setup considered so far. The tests have been performed using the Open62541 stack, on the Wi-Fi configuration, with the generic operating system and without CPU isolation. In detail, we measured the current consumption on the client side, as well as the time necessary to complete the experimental session described in Subsection V.B, that comprised the execution of $N = 100.000$ OPC UA test tasks.

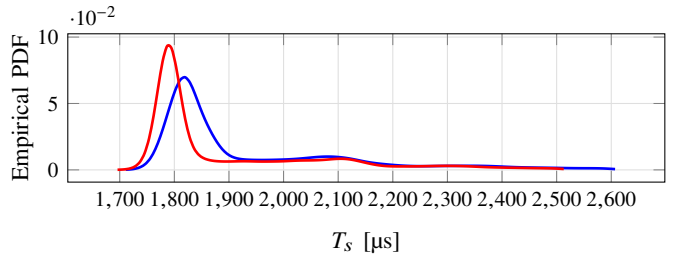
The circuit implemented for current measurement is described in Figure 8. The Raspberry Pi was powered by a stabilized power supply, providing a 5 V continuous voltage. The adsorbed current was measured using a Hall effect sensor whose sensitivity is 185 mV/A. Current measurements have been acquired using an external digital acquisition system equipped with a 12 bit ADC with an input range of [0 , 3.3] V at a sampling rate of 1 kHz. Each time the OPC UA test task is started, the Raspberry Pi rises a signal triggering a new acquisition of the current level, which is also timestamped for further analysis.

TABLE VI
STATISTICS OF THE CURRENT CONSUMPTION WITH CPU GOVERNOR DISABLED AND ENABLED

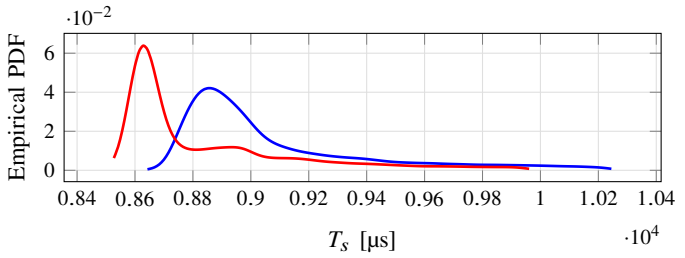
	Current [A]			**Session completion time** [ms]
	Mean	Std	Idle	
Governor Enabled	0.4175	0.0587	0.3928	235924
Governor Disabled	0.4767	0.0760	0.4462	215259



(a) open62541

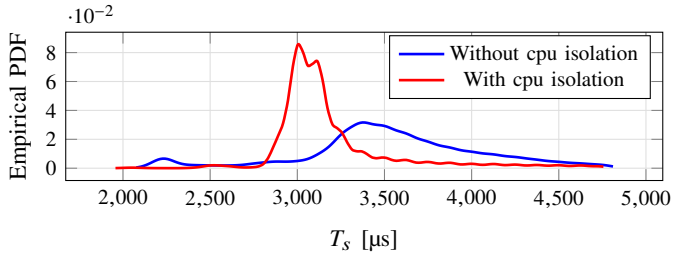


(b) FreeOPC UA C++

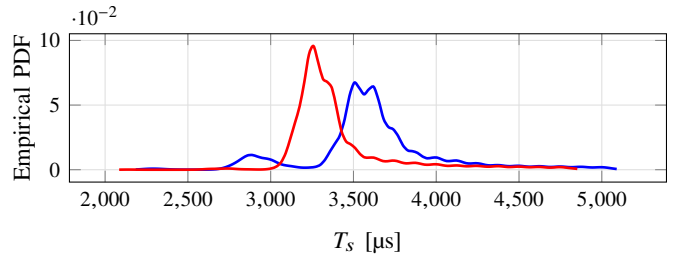


(c) FreeOPC UA Python

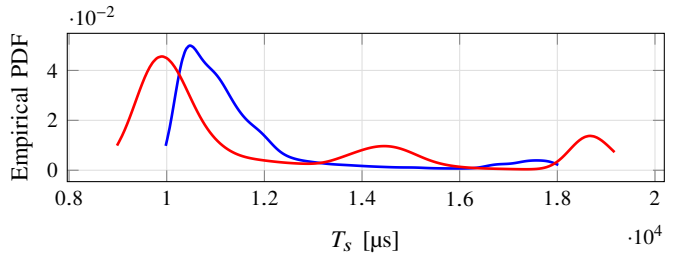
Fig. 6. Generic OS: EPDF of the execution time of the OPC UA test task for the Wi-Fi configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, where both server and client are forced to run on the isolated CPU.



(a) open62541



(b) FreeOPC UA C++



(c) FreeOPC UA Python

Fig. 7. RT OS: EPDF of the execution time of the OPC UA test task for the Wi-Fi configuration. Blue line: configuration without CPU isolation. Red line: configuration with CPU isolation enabled, and where both server and client are forced to run on the isolated CPU.

The results of this new set of experiments are summarized in Table VI, which reports the statistics of the current consumption for the two cases in which the governor was, respectively, enabled and disabled. The table also shows the current consumption in idle state (i.e. while the experimental session was not carried out), for comparison.

As can be seen, enabling the governor leads to a slight decrease of the current consumption, for both average and standard deviation values. However, the time necessary to complete the experimental session increases, of almost 10%, as a result of the continuous CPU frequency adjustments caused by the workload variations. Thus, at a first glance, it might be not worth to maintain the governor enabled, since the benefits achieved in term of power savings may result vanished by the performance degradation. However, a decision in this direction has to take into consideration other aspects, such as the specific devices adopted and the performance requirements.

A further observation can be made with respect to the current consumption in idle state. Table VI clearly shows only a limited increase of the current consumption, when moving from this state to that in which experiments were executed, regardless of the governor status (enabled or disabled). This may be explained considering that Open62541 uses very low CPU resources, as confirmed by the results presented in

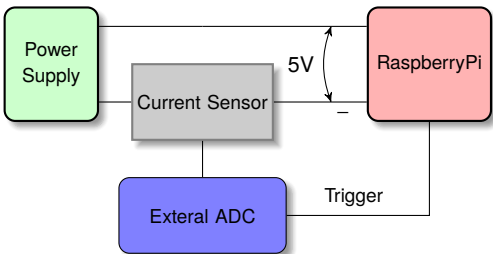


Fig. 8. Setup adopted for current measurements.

Table II that, evidently, are not sufficient to imply a remarkable variation in current consumption.

VI. Conclusion and Future works

In this paper we considered the case of IIoT measurement applications and proposed the adoption of OPC UA protocol to enable a seamless interoperability when heterogeneous sources of measurement data coexist sharing information over the network. We focused on three widespread open source implementation of the protocol, to analyze their impact on a networked measurement system, mostly in terms of power consumption and latency. A reproducible and effective measurement setup has been designed that allowed to carry out a thorough assessment, where the latency has been analyzed in

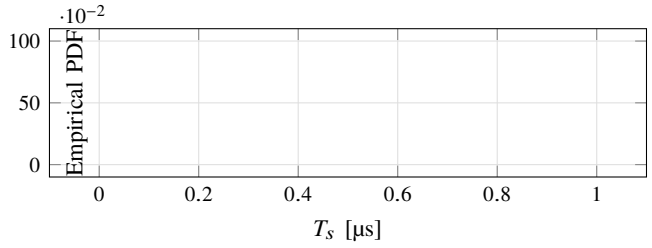


Fig. 9. EPDF of the execution time of the OPC UA test task for the Prosys Java implementation on Ethernet with Generic OS and CPU isolation.

different operating conditions, thus providing a rather complete characterization of the suitability of the OPC UA protocol for network-based measurement instrumentation.

Meaningful results have been obtained, allowing also to provide some interesting implementation guidelines. For instance, it has been verified that the use of a real-time operating system does not bring specific advantages whereas, in general, the best performances are achieved with a generic operating system exploiting the CPU isolation for the measurement application. Moreover, it has been shown that Open62541 is the most efficient among the examined implementations.

Furthermore, in compliance with modern IIoT-based applications, we considered the case of battery powered wireless measurement systems, thus providing some valuable insights about the expected power consumption in some selected and relevant cases. Indeed, the measurement campaign highlighted that the DVFS feature should be enabled, allowing for lower power consumption without compromising the performance.

The current work opens up to future analysis. For instance, power consumption depends also on intrinsic parameters of the accumulator and on environmental conditions, hence requiring a more extensive experimental campaign focused on mobile battery powered measurement instruments. In addition, the proposed experimental setup for latency analysis seems to be overkill for small integrated smart sensors. Hence, we plan to test the framework on low power embedded devices, like widespread microcontrollers with no operating systems.

VII. Prosys Java

TABLE VII
Statistics of the Execution Time for the OPC UA Test Task for the Prosys Java implementation on Ethernet with Generic OS and CPU isolation

	Execution Time T_s [µs]	
	Mean	Std
Prosys Java	2648.31	348.50

References

[1] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, Jun. 2017.

[2] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, Nov 2018.

[3] S. Vitturi, C. Zunino, and T. Sauter, "Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 944–961, Jun. 2019.

[4] D. Grimaldi and M. Marinov, "Distributed measurement systems," *Measurement*, vol. 30, no. 4, pp. 279–287, Dec. 2001.

[5] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*, 3rd ed. Boston: Morgan Kaufmann, 2011.

[6] G. Y. Tian, "Design and implementation of distributed measurement systems using fieldbus-based intelligent sensors," *IEEE Trans. on Instr. and Measurement*, vol. 50, no. 5, pp. 1197–1202, Oct 2001.

[7] L. Skrzypczak, D. Grimaldi, and R. Rak, "Analysis of the different wireless transmission technologies in distributed measurement systems," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009. IEEE International Workshop on*, Sept 2009, pp. 673–678.

[8] B. Ooi and S. Shirmohammadi, "The potential of IoT for instrumentation and measurement," *IEEE Instrumentation Measurement Magazine*, vol. 23, no. 3, pp. 21–26, 2020.

[9] D. Bruckner, M.-P. Stanica, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, "An Introduction to OPC UA TSN for Industrial Communication Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, Jun. 2019.

[10] International Electrotechnical Commission, *IEC 62541: OPC unified architecture - Part 1: Overview and concepts*. IEC, 2016.

[11] A. Morato, S. Vitturi, F. Tramarin, and A. Cenedese, "Assessment of different opc ua industrial iot solutions for distributed measurement applications," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2020, pp. 1–6.

[12] M. Rizzi, P. Ferrari, A. Flammini, and E. Sisinni, "Evaluation of the IoT LoRaWAN Solution for Distributed Measurement Applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 12, pp. 3340–3349, 2017.

[13] H. Lee and K. Ke, "Monitoring of large-area iot sensors using a lora wireless mesh network system: Design and evaluation," *IEEE Transactions on Instrumentation and Measurement*, vol. 67, no. 9, pp. 2177–2187, 2018.

[14] R. Palisetty and K. C. Ray, "FPGA Prototype and Real Time Analysis of Multiuser Variable Rate CI-GO-OFDMA," *IEEE Transactions on Instrumentation and Measurement*, vol. 67, no. 3, pp. 538–546, 2018.

[15] V. Bianchi, A. Boni, S. Fortunati, M. Giannetto, M. Careri, and I. De Munari, "A Wi-Fi Cloud-Based Portable Potentiostat for Electrochemical Biosensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 6, pp. 3232–3240, 2020.

[16] Y. Liao and H. Lai, "Investigation of a wireless real-time ph monitoring system based on ruthenium dioxide membrane ph sensor," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 2, pp. 479–487, 2020.

[17] G. Mois, S. Folea, and T. Sanislav, "Analysis of Three IoT-Based Wireless Sensors for Environmental Monitoring," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 8, pp. 2056–2064, 2017.

[18] S. C. Folea and G. D. Mois, "Lessons Learned From the Development of Wireless Environmental Sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 6, pp. 3470–3480, 2020.

[19] J. Cavalaglio Camargo Molano, A. Lahrache, R. Rubini, and M. Cocconcelli, "A new method for motion synchronization among multivendor's programmable controllers," *Measurement*, vol. 126, pp. 202 – 214, 2018.

[20] B. Montavon, M. Peterek, and R. Schmitt, "Model–based interfacing of large–scale metrology instruments," *Multimodal Sensing: Technologies and Applications*, June 2019.

[21] S. Lee, C. Kim, and J. Lee, "Development of a Smart Sensor System Using OPC UA," in *Proceedings of the 15th International Conference on Advances in Mobile Computing & Multimedia*, 2017, pp. 220–225.

[22] I. González, A. J. Calderón, A. J. Barragán, and J. M. Andújar, "Integration of Sensors, Controllers and Instruments Using a Novel OPC Architecture," *Sensors*, vol. 17, no. 7, 2017.

[23] P. Ferrari, A. Flammini, S. Rinaldi, E. Sisinni, D. Maffei, and M. Malara, "Impact of Quality of Service on Cloud Based Industrial IoT Applications with OPC UA," *Electronics*, vol. 7, no. 7, p. 109, July 2018.

[24] M. Damm, S.-H. Leitner, and W. Mahnke, *OPC Unified Architecture*. Springer-Verlag Berlin Heidelberg, 2009.

[25] Arm architecture reference manual armv8, for armv8-a architecture profile. [Online]. Available: https://developer.arm.com/documentation/ddi0487/fb/

[26] "Raspberry Pi: The N-queens Problem (benchmark) Preempt-RT vs. Standard Kernel," https://lemariva.com/blog/2018/04/raspberry-pi-the-n-queens-problem-performance-test, (accessed 2020-07-17).

[27] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart, "A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, Jan. 2011.

[28] M. P. Karpowicz, "Energy-efficient CPU frequency control for the Linux system," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 2, pp. 420–437, 2016.