Scalability of Kura-extended Gateways via MQTT-CoAP Integration and Hierarchical Optimizations

Paolo Bellavista
Dept. Computer Science and Engineering (DISI)
Alma Mater Studiorum – University of Bologna
Viale del Risorgimento, 2 – 40136 Bologna, Italy
paolo.bellavista@unibo.it

Alessandro Zanni
Dept. Computer Science and Engineering (DISI)
Alma Mater Studiorum – University of Bologna
Viale del Risorgimento, 2 – 40136 Bologna, Italy
alessandro.zanni3@unibo.it

ABSTRACT

The introduction of innovative Internet of Things (IoT) gateways, powered by efficient support protocols and tools, has the potential to boost IoT development and related opportunities. Some first open-source platforms and solutions have been developed along that direction, primarily with the goals of targeting device limitations in terms of available resources and of simplifying management operations in heterogeneous environments. This paper originally proposes a multi-protocol IoT gateway solution with the specific and main goals of enabling efficiency and scalability via proper MQTT-CoAP integration. To this purpose, we have extended the Kura framework, which exploits only MQTT for machine-to-machine communication, by implementing a significant CoAP support for scalable hierarchy-based coordination, capable of externally exposing any kind of resource with REST APIs. The paper also reports an extensive set of in-the-field measurements that show the scalability of the proposed prototype that, to achieve the desired performance, also exploits some original optimizations introduced by our work on both Kura and Californium frameworks.

CCS CONCEPTS

• **Computer systems organization** → **Architectures** → **Distributed Architectures** → n-tier *architectures*; • **Networks** → **Network types** → Cyber-physical networks → Sensor networks

KEYWORDS

Internet of Things (IoT), Scalability, Cloud-Sensor Interaction, IoT Gateways, Kura, Californium, MQTT, CoAP

1 INTRODUCTION

The Internet of Things (IoT) has rapidly become a tremendously growing research area of interest for both academia and industries. As many recent research activities have depicted, this exponential growth is expected to continue in the near future, towards a challenging mobile scenario where dynamicity and scalability are pushed to the extreme, thus posing very relevant and open technical challenges. This scenario also entails the relevance of middleware components, e.g., IoT gateways, able to reduce the complexities of IoT system deployment and application development, in particular to seamlessly support interoperability and scalability. Indeed, in the related literature IoT gateways are starting to demonstrate their potential in providing: i) scalability and data aggregations, working as dynamically identified data sinks; ii) interoperability at dynamically determined and most relevant network edges; iii) local support to resource-constrained devices (e.g., in the case of few remaining storage resources or energy power); iv) registration and discovery facilities, dynamically and efficiently, on behalf of their served devices.

Above all, we believe that IoT gateway scalability research is still in its infancy, with many open-source and industry-oriented platform initiatives still leaving open space to improve their scalability via specific optimizations. In particular, this paper proposes a significant scalability-oriented extension of the existing Kura industrial platform that allows hierarchical and efficient device/resource synchronization management in IoT scenarios. Our original solution is based on a dynamically managed tree structure [1] where, to provide scalability, we integrate the usage of standard MQTT (already included in the official Kura release) with the industry-widespread CoAP and the Californium framework.

By delving into finer details, on the one hand, we adopt a hierarchical tree solution to benefit from the topic-based publish/subscribe structure of MQTT that exploits hierarchical namespaces: we use MQTT to subscribe our IoT gateways and entities to multiple topics, to facilitate message management towards all the nodes of the hierarchy, and to add/remove IoT entities without changes in the whole topic

BODYNETS 2016, December 15-16, Turin, Italy

DOI 10.4108/eai.15-12-2016.2267595

hierarchy. On the other hand, MQTT has demonstrated some non-negligible limitations, in particular related to TCP usage and communications hold always open towards the employed brokers; for this reason, we have efficiently and scalably integrated MQTT in IoT Kura gateways with some purpose-specific CoAP usage (connection-less datagrams on top of UDP), especially on behalf of constrained devices, as detailed below. More specifically, we claim that our proposed solution is relevant and with good technical originality for multiple aspects:

- our solution is built on the top of a very promising and powerful open-source initiative, i.e., Kura, with mature ancillary frameworks, i.e., Guava and Kryo, and emerging open-standard protocols, i.e., MQTT and CoAP. In addition, SSL and DTLS integration is achieved for securing machine-to-machine communications;
- we have carefully evaluated MQTT and CoAP characteristics, in particular in terms of system performance and scalability, in order to combine them efficiently and to minimize their specific drawbacks;
- we propose an original IoT gateway extension, based on a dynamic and efficient tree structure to allow IoT resources on different devices to scalably interact even if deployed over a large-scale distributed area;
- we originally describe how we have specifically optimized both Kura MQTT and Californium CoAP supports to improve scalability performance when employed in our extended IoT gateways.

2 RELATED WORK

After seminal activities on scalability through replication and location-based optimizations [2, 3]Some research activities in the literature have recently proposed architectures to address IoT sensors/actuators federation, in particular within large-scale deployment scenarios, with particular focus on how to effectively and efficiently achieve scalability [2]. Even if they explored relevant solution guidelines and were someway inspiring for the community of researchers in the field, we claim that they only partially solved some aspects related to this hard technical challenge, thus leaving still open space for additional, especially industry-mature, solutions.

In particular, [4] proposes an architecture based on the RELOAD base protocol that provides a generic self-organized P2P overlay network service, by using CoAP as the transfer protocol to access RESTful services. Here CoAP is used as the common application layer protocol for heterogeneous and constrained devices, while RELOAD is employed primarily on proxy nodes with higher amount of resources. This work is also relevant to point out the relevance of open standard-based solutions for interoperability and the unsuitability of CoAP alone, e.g., because it is inappropriate to scalably aggregate IoT resources into a hierarchical organization.

[5] proposes a 3-layers architecture for cloud-integrated IoT services based on CoAP. The work proposes a Web integration platform that collects data from IoT devices and can outperform high-performance general-purpose Web servers that are not optimized for the IoT. [5] uses the Californium framework for CoAP integration, by showing its good throughput performance also in high-concurrency situations. Along with CoAP, the solution in [5] employs multi-threaded sockets, but shows that this implementation option is very resource-demanding while growing the number of clients, in particular in terms of memory.

Instead, [6, 7] mainly highlight successful examples of MQTT usage in some application domains. In particular, [6] proposes an architecture for adaptive periodic many-to-one communication in large-scale cyber-physical systems. The paper shows how MQTT can guarantee good flexibility in highly dynamic execution environments where publishers and subscribers can join/leave frequently. [7] reports the experience of a smart home management system where MQTT is used over a hierarchical architecture similar to ours, but with no integrated exploitation of CoAP.

3 AN ORIGINAL KURA EXTENSION FOR SCALABILITY THROUGH COAP

Our original Kura-extended IoT gateway exploits the integration and interworking between MQTT, already fully supported within Kura, and CoAP, with the main purpose of improving scalability in a resource-efficient way. In particular, we have organized IoT devices in a dynamic and hierarchical tree structure, where MQTT is primarily used for tree management and inter-node coordination/communication; instead, CoAP is primarily used for device discovery and external accessibility via REST. For the sake of briefness, we refer to [1] for the general description of our tree structure and for the description of the primary characteristics of the industry-widespread MQTT and CoAP protocols; in this paper, we originally focus on relevant design/implementation insights and performance results about our prototype.

By delving into finer details, we have decided to use MQTT for inter-node coordination/communication inside our tree organization to efficiently manage large-scale deployments via hierarchy- and locality-oriented optimizations. In particular, we exploit the MQTT publish/subscribe model for communications between CoAP servers to retrieve IoT device information and for tree management synchronization. Publishers and subscribers are well decoupled each other, thus also simplifying the dynamic variations in the tree structure via hierarchical topics: our nodes may subscribe to a topic and then observe their whole hierarchy by using wildcards. In addition, we exploit the useful MQTT

characteristics of reliability and seamless NAT traversal in these cases.

Notwithstanding its many pros, it starts to be recognized that MQTT also presents some weaknesses for highly scalable IoT applications, such as, only to rapidly overview the two most relevant and recognized ones: i) TCP usage, more suitable for resource-full nodes than for constrained IoT devices; and ii) persistent (and expensive) node-broker TCP connections. Therefore, we believe that an MQTT-alone-based IoT gateway is inappropriate in large-scale scenarios and that CoAP is the right protocol to complement MQTT in IoT gateway implementations. In other words, the pervasive usage of MQTT introduces overhead for functions that are not crucial: using a less reliable but also less resource-demanding connection is often more suitable, especially if the sporadic loss of partial data is not critically impacting the overall system behavior. In particular, our original IoT gateway solution exploits CoAP i) to easily add new discoverable devices at runtime (accessible via REST API); and ii) for interactions where we need direct and very responsive lightweight communications, with low reliability constraints, such as for coordination and simple service interaction within single localities, where NAT drawbacks are ineffective and communication reliability is anyway adequate.

4 OUR KURA-EXTENDED IOT GATEWAY: IMPLEMENTATION INSIGHTS

This section presents our implementation work organized into i) internal node structure, replicated into all nodes, and ii) tree-based hierarchy management. The implemented prototype is based on widespread industry-mature solutions and frameworks, such as the MQTT and CoAP protocols, as well as the Kura [1] and Californium frameworks. Californium [8] is currently the most complete open-source implementation of CoAP, with the partial/complete support of many draft extensions. In particular, it provides implementations for: the observe draft [9] to enable CoAP clients to retrieve and update resource representations; the blockwise transfers draft [10] to support stateless behavior while handling transfer blocks separately; and the resource directory draft [11]. The Californium CoAP payload adopts the CoRE Link Format [12], uses the Datagram Transport Layer Security (DTLS) protocol [13], and exploits CoAP-HTTP cross-proxy support through HttpCore-NIO [14] and Guava [15].

4.1 Internal Node Structure

Our CoAP-based extension of the Kura IoT gateway is modularly and flexibly implemented via OSGi bundles, executed by the Kura framework on each IoT node. In addition, Kura allows to exploit a native MQTT support to select differentiated QoS levels: exactly-once semantics can be accessed via the so-called QoS2 mode and we use it for most relevant hierarchy management messages; MQTT QoS1, instead, offers at-least-once semantics that we adopt for more common inter-node communication, with overall performance benefits (see Section 5). For the sake of modularity and loose coupling, any node is structured into multiple components: CoAP server, Californium service, Resource Directory (RD), Remote Query Resource (RQ), and CoAP resource.

The CoAP server is the central component within a node, manages the properties of registered CoAP resources, and uses Californium to offer REST API operations. In particular, CoAP servers use the RD component as a data structure to store endpoints and to enable usual registry operations (e.g., register, maintain, lookup, remove endpoint and resource description) via REST requests. Each CoAP server leverages the RQ bundle for resource lookup inside the hierarchy, by interfacing CoAP server requests with the MQTT broker, with CoAPPublisher and CoAPSubscriber classes. The resource lookup is performed by a set of iterative requests towards other hierarchy nodes, starting from the parent node, that reply with the direct link to the resource or with a list of CoAP servers that may contain the resource for further search (see Figure 1). Finally, we provide the CoAP server implementation with the DTLS support extension, using the existing subproject Scandium [16] inside Californium.

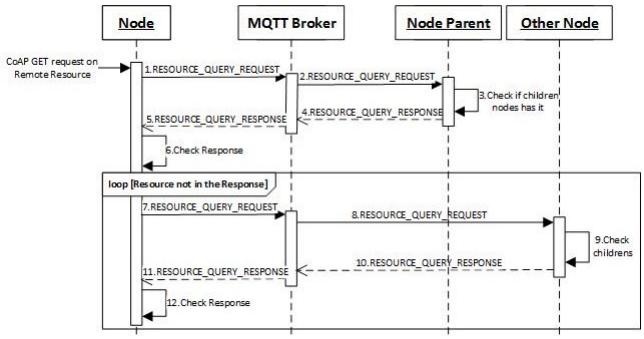

Figure 1: Resource lookup iterations.

4.2 Tree-based Hierarchy Management

Our IoT gateway dynamically joins a multi-layer hierarchy, structured into three levels, where each node specifies domain and group name used for identifying it. Level0 includes the root node and the CoAPTreeHandler (CTH), which enable inter-domain communication and manage the CoAP servers hierarchy, while level1 and level2 include all the nodes belonging, respectively, to a specific domain or domain/sub-group.

4.2.1 CoapTreeHandler. The CTH bundle dynamically manages the hierarchy and dispatch node requests: it provides hierarchy metadata to all nodes, returns the references to create a tree-structure organized with domains/groups; manages multiple node replications; and handles children nodes in case of parent disconnections. Internally, CTH consists of CTHListener and CTHCollections. CTHListener exposes an interface to interact with CTH, triggered by request from nodes that require actions on the hierarchy. The CTHCollections class stores information about the overall hierarchy and consists of two collections optimized for lookup and node substitutions: a Map-extension data structure to store node paths and properties; a data structure to map parent-children associations by using the Guava library in order to simplify collection updates, i.e., dynamic creation of one-to-many associations. To practically exemplify how the hierarchy management works, here we rapidly outline some common use cases:

- New root request. CTH returns the root reference and stores into CTHCollection the node address (root duplicates management). When root disconnects, CTHCollection contains references to suitable new root nodes to replace it and notify level1 nodes;
- Node addition (Figure 2). By providing its path, a node requests root information: if root is present, the node is notified with the root node reference (root_response); if root unavailable, the node is added as root (root_added) or updated (root_not_available). CTH checks the path provided: if new path, the node is added to the hierarchy; otherwise, the node becomes the child of the original node and will able to perform resource lookup or return a local resource;
- Node removal. i) If the node to be removed is the root, CTHCollection provides the reference to the new root, otherwise CTH searches a new root on level1. ii) If leaf, CTH only updates the hierarchy information. iii) Otherwise (Figure 3), children are associated to the root while CTH looks for a new parent, then children are restored;
- CTH failure. During CTH inaccessibility the root is not aware of hierarchy updates. Once CTH reconnects, an emergency MQTT topic (cth_online) triggers a stateless recovery with hierarchy node soft-reboot: i) parents remove all properties attached; ii) children remove parents and resend the request to join the hierarchy; iii) resource collections are sent to the parents and to CTH to be aligned again.

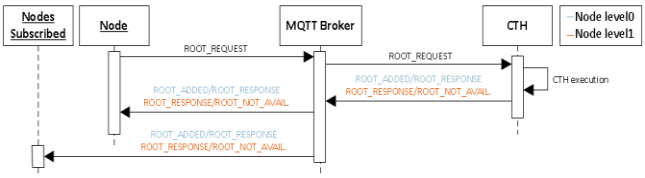

Figure 2: Node Connection iterations.

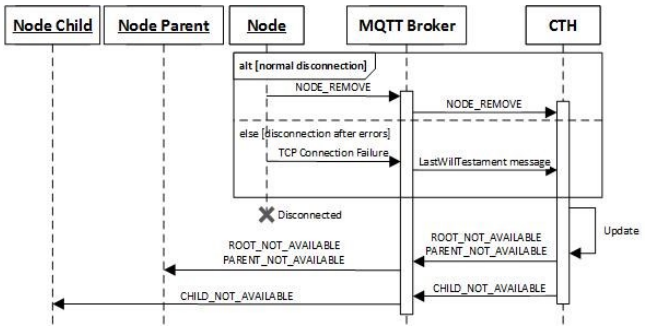

Figure 3: Node removal iterations.

4.3 Performance Optimizations

Although Kura and Californium are widespread and industry-mature frameworks, their default configurations have demonstrated to be not specifically optimized for our hierarchical IoT gateway organization. Thus, we introduce some non-negligible implementation/configuration improvements, focusing on the following primary critical scalability aspects: reducing the number of MQTT messages exchanged among levels and nodes; reducing the number of POST requests to RD.

4.3.1 Object Serialization (OS). MQTT is content-less and only support byte array as payload content. Thus, serialization is performed very often during regular operation: each CoAP message is serialized, then used into the Kura DataService class for MQTT communication, and de-serialized during responses. We have improved performance by integrating serialization based on the efficient Kryo framework. Kryo [17] is an open-source object graph serialization framework for Java.

4.3.2 Data Service (DS) and DataTransportService. The DataService class is the Kura component used to manage MQTT communication and offers several configuration options, delegating to the DataTransportService the implementation of the transport protocol to interact with the associated MQTT broker. When DataService receives a publish request, it stores the message into a DataStore and submits the message on the internal executor of DataTransportService. DataStore is a heavy storage structure that, in case of many messages published concurrently, may even cause

unavailability of the device because of its high CPU consumption. Therefore, to reduce associated overhead, we have decided to exploit only the lower-level DataTransportService class, by removing the support to the features, irrelevant for our solution, of message priority management and message storage on behalf of temporarily disconnected devices. In addition, our DataTransportService performs event propagation by accessing methods and client listeners of the MQTT Paho implementation [18].

Our experimental work has pointed out that the default configuration of the Californium framework exhibits some performance drawbacks for highly scalable scenarios, in particular in terms of efficient message parsing. We have optimized the related aspects by providing original extensions for i) efficient data structures for CoAP payload parsing, ii) efficient support to most frequent regular expressions to identify message sections, and iii) exploitation of message setting patterns available into the CoRE Link Format.

4.3.3 RD Parsing (RP) and String Refactoring (SR). To optimize regular expression management and to speed up parsing, we have decided to represent any CoRE link with resource path in the first position of the RD attribute list (fixed static position). In addition, we adopt Guava libraries for efficient string management; for instance, we have replaced the Java Scanner class with the specialized Guava Splitter and used the optimized StringBuilder for string declaration.

4.3.4 Request Aggregation (RA). Since every node can send POST messages, it is likely that some nodes may send information about multiple resources. Thus, we optimize message management whenever possible (in absence of strict latency requirements) by aggregating multiple resources together into a single CoRE link format composition.

5 IN-THE-FIELD PERFORMANCE RESULTS

To quantitatively evaluate the scalability of our tree-based IoT gateway organization and the efficiency of our combined usage of MQTT and CoAP, we have extensively tested our solution in realistic application/deployment scenarios: the results reported in this section are average values over multiple runs and also compare the performance of MQTT and Californium default configurations with our original optimizations described above. In particular, in our tests we have used growing numbers of sequential MQTT requests and CoAP POST ones, distributed over a realistic testbed environment consisting of 10 nodes organized in a 4-layer tree, 20 service bundles per node, 2 devices per service bundle, and 10 sensors per device. Each regular (non-root) node is equipped with RaspberryPi 1 model B+, with Raspbian Whezzy, Kura version 1.1.1; Parallels VM, with 512 MB Ram, single core CPU, Xubuntu OS. The root node is 2.2 GHz Intel i7 CPU, 16 GB RAM, 1600 MHz DDR3. The integrated MQTT Broker is Mosquitto. Additional details about the exact setup of our testbed deployment and further performance results, e.g., on DTLS overhead, are not reported here for the sake of briefness and are available at our Web site http://lia.disi.unibo.it/research/MQTT_CoAP_Integration.

MQTT-related evaluation tests have started by sending a peak of 1000 concurrent requests between two RaspberryPi. Without any of our optimizations described above, our prototype is not able to handle them; exceptions related to too many messages stored and too many messages waiting the ACK are observed. Thanks to our object serialization optimization, we have observed a significant decrease of the total time needed to serve all the requests (around 49%), but anyway this is still insufficient to manage such a large peak with no exception occurrences. By adding also our DataService optimization, we have experienced another significant performance improvement, as well as the ability not to enter in exception situations due to overload.

After that preliminary experimentation, we have evaluated the behavior of our prototype while further increasing the number of MQTT exchanged messages. Table 1 reports the total time to complete MQTT transmissions in relation to the optimizations introduced. Note that, due to their limited capabilities and the high number of operations to perform, sometimes RaspberryPi nodes are affected by connection lost errors with the MQTT broker; this has shown to be prevalently due to the inability of sending MQTT heartbeat messages in time to keep the associated connection alive.

Table 1: MQTT Performance Results

Sender/ Receiver	Optimization	MQTT Req.	Time (s)	Errors
Raspberry/ Raspberry	-	1000	211	KuraStoreCapacityReached KuraTooManyInflightMessages
Raspberry/ Raspberry	OS	1000	109	KuraStoreCapacityReached KuraTooManyInflightMessages
Raspberry/ Raspberry	OS - DS	1000	14	-
VM/ Raspberry	OS - DS	1000	5	-
VM/VM	OS - DS	1000	1	-
Raspberry/ Raspberry	OS - DS	10000	120	Lost Connection

VM/ Raspberry	OS - DS	10000	60	-
VM/VM	OS - DS	10000	7	-
Raspberry/ Raspberry	OS - DS	60000	723	Lost Connection
VM/ Raspberry	OS - DS	60000	377	-
VM/VM	OS - DS	60000	34	-

By passing to some relevant results about RD (CoAP-centered) evaluation, we have started RD tests by sending 500 POST requests: already with this number of requests, the default Californium configuration has shown non-negligible performance issues; for instance, when working on top of a limited gateway such as the RaspberryPi one, this load peak may frequently generate errors. Therefore, we have applied our original RD optimizations, also in different partial subsets; Table 2 reports the related performance results in terms of total time to complete the POST requests on RD, with the different possible subsets of optimizations introduced.

Table 2: RD Performance Results

Sender/ Receiver	Optimization	POST Req.	Time (s)	Errors
Raspberry/ Raspberry	-	500	99	POST timeout
VM/ Raspberry	-	500	12	-
VM/VM	-	500	6	-
Raspberry/ Raspberry	RP	500	68	Possible POST timeout
VM/ Raspberry	RP	500	7	-
VM/VM	RP	500	4	-
Raspberry/ Raspberry	RP – RA	500	55	-
VM/ Raspberry	RP – RA	500	5	-
VM/VM	RP – RA	500	3	-
Raspberry/ Raspberry	RP – RA – SR	500	19	-
VM/ Raspberry	RP – RA – SR	500	4	-
VM/VM	RP – RA – SR	500	0.8	-
Raspberry/ Raspberry	RP – RA – SR	1000	51	-
VM/ Raspberry	RP – RA – SR	1000	11	-
VM/VM	RP – RA – SR	1000	1.8	-
Raspberry/ Raspberry	RP – RA – SR	5000	255	POST timeout, not responsive device, Californium server stops
VM/ Raspberry	RP – RA – SR	5000	54	-
VM/VM	RP – RA – SR	5000	3.3	-
Raspberry/ Raspberry	RP – RA – SR	10000	491	POST timeout, not responsive device, Californium server stops
VM/ Raspberry	RP – RA – SR	10000	104	-
VM/VM	RP – RA – SR	10000	6.9	-

In addition to the above results for communication performance, we have thoroughly investigated resource consumption on IoT gateways, which is crucial also to maintain full responsiveness of the participating nodes. CPU and RAM usage on RaspberryPi nodes, when 500 resources are registered (four description attributes each, on average) via POST REST invocations, is respectively 92% and 80%, with around 550 active threads. In fact, every time RD stores a resource, the Californium-based CoAP implementation associates a new thread to it in order to monitor its validity time and delete the registration once it expires. With a high amount of concurrent requests, this multi-threaded support is clearly not adequate. Therefore, we have decided to change the associated multi-threaded support via an optimized set of worker threads, each of them handling a partition of resources (the cardinality of the partition is dynamically determined and can change over time – self-adapting monitoring).

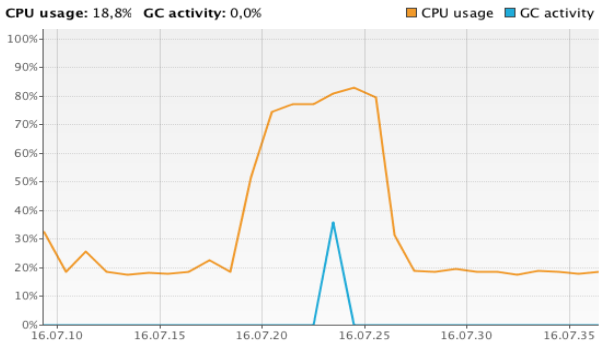

Figure 4: CPU usage

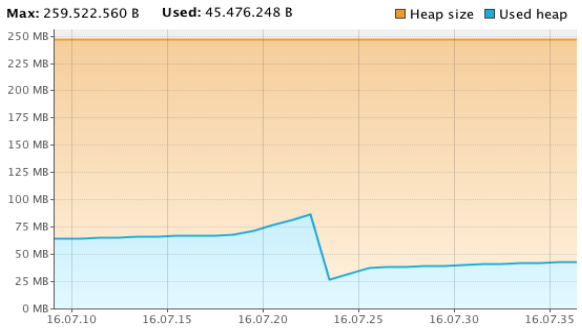

Figure 5: RAM usage

In particular, registration update periodicity may vary depending on desired responsiveness and available resources at gateways; consider also that in our application scenario, a false positive in resource registrations only generates the waste of a useless lightweight CoAP request that will not receive any response. Figures 4 and 5 show the performance results about resource consumption in the thread-optimized case (only 35 active threads on average).

6 CONCLUSIVE REMARKS

The paper has presented most significant implementation insights and in-the-field performance results about our CoAP-integrated Kura-extended IoT gateway. The reported results show that our solution can preserve scalability also in high-load scenarios, with overall latency that is proportional to the number of incoming requests and limited overhead. In addition, we have described some implementation/configuration optimizations that significantly improve the performance of both the standard Kura gateway and the exploited Californium framework for CoAP. Moreover, we were able to achieve good overall performance also with IoT devices with very strict resource constraints, like the used RaspberryPis.

ACKNOWLEDGMENTS

The Authors would like to take the opportunity to thank all the MSc students who participated to this project, with their extension and experimentation activities during course projects and MSc graduation works. A special thank goes to Gianvito Morena, whose contribution was substantial and essential for the results reported in this paper.

REFERENCES

[1] P. Bellavista, A. Zanni, "Towards Better Scalability for IoT-Cloud Interactions via Combined Exploitation of MQTT and CoAP", IEEE 2nd International Forum on Research and Technologies for Society and Industry (RTSI), 2016.
[2] P. Bellavista, A. Corradi, C. Giannelli. "Efficiently Managing Location Information with Privacy Requirements in Wi-Fi Networks: a Middleware Approach", 2nd Int. Symp. Wireless Communication Systems. IEEE, 2005.
[3] P. Bellavista, A. Corradi, E. Magistretti. "REDMAN: an Optimistic Replication Middleware for Read-only Resources in Dense MANETs", Elsevier Pervasive and Mobile Computing Journal, Vol. 1, No. 3, pp. 279-310, 2005.
[4] J. Mäenpää, J. Jiménez Bolonio, S. Loreto, "Using RELOAD and CoAP for Wide Area Sensor and Actuator Networking", Eurasip J. Wireless Communications and Networking, Vol. 121, pp. 1-22, 2012.
[5] M. Kovatsch, M. Lanter, Z. Shelby, "Californium: Scalable Cloud Services for the Internet of Things with CoAP", IEEE 4th International Conference on the Internet of Things, 2014.
[6] H. Jo, H. Jin, "Adaptive Periodic Communication over MQTT for Large-Scale Cyber-Physical Systems", IEEE 3rd Int. Conference on Cyber-Physical Systems, Networks, Applications, 2015.
[7] Y.T. Lee, et al., "An Integrated Cloud-Based Smart Home Management System with Community Hierarchy", IEEE Transactions on Consumer Electronics, vol. 62, issue 1, 2016.
[8] Californium. Available online at: https://eclipse.org/californium.
[9] K. Hartke, Observing Resources in CoAP, IETF Internet Draft, 2014. Available online at: https://tools.ietf.org/html/draft-ietf-core-observe-16.
[10] C. Bormann, "Block-wise transfers in CoAP draft-ietf-core-block-17", IETF Internet Draft, 2015. Available online at: https://tools.ietf.org/html/draft-ietf-core-block-17.
[11] Z. Shelby, C. Bormann, "CoRE Resource Directory draft-ietf-core-resource-directory-02", IETF Internet Draft, 2014. Available online at: http://tools.ietf.org/html/draft-ietf-core-resource-directory-02.
[12] Z. Shelby, "Constrained RESTful Environments (CoRE) Link Format", IETF Internet Draft, 2012. Available online at: https://tools.ietf.org/html/rfc6690.
[13] E. Rescorla, N. Modadugu, "Datagram Transport Layer Security Version 1.2", IETF Internet Draft, 2012. Available online at: https://tools.ietf.org/html/rfc6347.
[14] HttpCore-NIO. Available online at: http://hc.apache.org/httpcomponents-core-ga/index.html.
[15] K. Boumillion, J. Levy, "Guava: Google core libraries for Java 1.5+". Available online at: https://github.com/google/guava.
[16] Scandium. Available online at: https://github.com/eclipse/californium.scandium.
[17] Kryo. Available online at: https://github.com/EsotericSoftware/kryo.
[18] Paho. Available online at: https://eclipse.org/paho.