Journal of
Sensor and Actuator Networks

MDPI

Article

Virtual Replication of IoT Hubs in the Cloud: A Flexible Approach to Smart Object Management

Simone Cirani [1], Gianluigi Ferrari [2], Mirko Mancin [2] and Marco Picone [1,*]

[1] Caligoo S.r.l, Via G. Falcone, 12, 42021 Barco di Bibbiano, Reggio Emilia, Italy; simone.cirani@caligoo.com
[2] IoT Laboratory, Department of Information Engineering, University of Parma, Parco Area delle Scienze 181/A, 43124 Parma, Italy; gianluigi.ferrari@unipr.it (G.F.); mirko.mancin@studenti.unipr.it (M.M.)
* Correspondence: marco.picone@caligoo.com

Received: 27 December 2017; Accepted: 20 March 2018; Published: 26 March 2018

Abstract: In future years, the Internet of Things is expected to interconnect billions of highly heterogeneous devices, denoted as "smart objects", enabling the development of innovative distributed applications. Smart objects are constrained sensor/actuator-equipped devices, in terms of computational power and available memory. In order to cope with the diverse physical connectivity technologies of smart objects, the Internet Protocol is foreseen as the common "language" for full interoperability and as a unifying factor for integration with the Internet. Large-scale platforms for interconnected devices are required to effectively manage resources provided by smart objects. In this work, we present a novel architecture for the management of large numbers of resources in a scalable, seamless, and secure way. The proposed architecture is based on a network element, denoted as IoT Hub, placed at the border of the constrained network, which implements the following functions: service discovery; border router; HTTP/Constrained Application Protocol (CoAP) and CoAP/CoAP proxy; cache; and resource directory. In order to protect smart objects (which cannot, because of their constrained nature, serve a large number of concurrent requests) and the IoT Hub (which serves as a gateway to the constrained network), we introduce the concept of virtual IoT Hub replica: a Cloud-based "entity" replicating all the functions of a physical IoT Hub, which external clients will query to access resources. IoT Hub replicas are constantly synchronized with the physical IoT Hub through a low-overhead protocol based on Message Queue Telemetry Transport (MQTT). An experimental evaluation, proving the feasibility and advantages of the proposed architecture, is presented.

Keywords: Internet of Things; edge computing; cloud computing

1. Introduction

The Internet of Things (IoT) is expected to consist in a worldwide network comprising, by 2020, more than 50 billions devices. This gigantic number of pervasively deployed devices will (i) enable new forms of interaction between things and people and (ii) foster novel applications, denoted as Smart-X Applications. The IoT will be characterized by the extreme heterogeneity of the involved devices, which will range from highly constrained Class 0, Class 1 and Class 2 devices [1] (operating in Low power and Lossy Networks, IP(V6) LLNs), to smartphones, traditional hosts, and the Cloud. IP has been foreseen as the driver for a successful growth of the IoT [2]. An IP-based protocol stack will also allow to maximize the reutilization of the experience derived from the development of Internet architectures and protocols. IP will, in fact, enable maximum interoperability among devices and integration with the existing Internet. Access to LLNs typically occurs through a border router, a network element with multiple network interfaces: one towards the LLN and the others toward the Internet using different protocols, either wired (e.g., IEEE 802.3) or wireless (e.g., IEEE 802.11, low power Wi-Fi, IEEE 802.15.4, BLE). Due to its less limited capabilities, the border router acts as an IPv6 network coordinator for the LLN, typically being the root of a IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) tree [3].

The work carried out by standardization organizations, such as the Internet Engineering Task Force (IETF), and several research projects has focused on defining mechanisms to bring IP connections to constrained devices and to design IoT-oriented application layer protocols, such as the Constrained Application Protocol (CoAP) [4]. CoAP is a lightweight, UDP-based, binary protocol designed to bring the REpresentational State Transfer (REST) paradigm to the IoT, targeting constrained applications. According to the request/response model of the REST paradigm, smart objects expose their resources using CoAP. REST has been selected as a reference communication paradigm for IoT, as it minimizes the coupling between client and server applications and promotes long-term evolution of applications, which is particularly desirable in IoT scenarios, where devices are expected to be deployed and remain operational for years.

At the application layer, other protocols have been designed for efficient communications in IoT and Machine-to-Machine (M2M) scenarios. Message Queue Telemetry Transport (MQTT) is a lightweight publish/subscribe (pub/sub) protocol over TCP/IP [5]. MQTT uses topics to "tag" messages, which are sent by publishers to a broker that, according to topic subscriptions, forwards messages to subscribers. Unlike CoAP, MQTT is not suited to highly constrained devices operating in LLNs: TCP connection establishment is a heavy operation, that might not be feasible for devices with limited resources and operating over lossy links. In order to overcome this pitfall, an extension of MQTT for Sensor Network (MQTT-SN) has been introduced. The pub/sub model implemented by MQTT enables reliable communications and minimizes network bandwidth, which make it perfectly suitable for fast transmissions among non-constrained devices.

Billions of IoT sensing/actuating devices will generate an unprecedented amount of data, which will need to be managed properly in order to provide highly available and robust services. Although IP-based IoT would make it possible to address smart objects directly, the following potential drawbacks exist: (i) in order to extend their battery lifetime, smart objects may be duty-cycled devices and, thus not always accessible; (ii) smart objects might be unable, to handle a large number of concurrent requests, thus leading to service disruption and becoming possible targets for Denial-of-Service (DoS) attacks; (iii) under some circumstances (for instance, to minimize memory footprint when the available in-device memory is critically low), smart objects may play the role of clients rather than servers.

In all the above cases, the presence of an intermediate network element operating at the application layer, which may typically coincide with the border router, is desirable. Such a node could integrate several functionalities which may reduce the processing load on smart objects and help overcome some of the problems discussed above (e.g., caching). This resourceful node, denoted as "IoT Hub" and shown in Figure 1, helps moving some of the processing load towards the edge of the network, following the emerging paradigm of "Fog Computing" [6].

Fog Computing, also referred to as Edge Computing, is a novel paradigm, which has been introduced to meet several requirements such as mobility support, low latency, and location awareness [6]. Fog Computing aims at distributing and moving some Cloud-based computation and storage to the edge of the network. While Cloud-only architectures can provide a solution to scalability and flexibility issues by distributing resources among multiple servers, this approach presents some weaknesses, such as: (i) latency; (ii) availability/dependence on Internet connectivity for operations; (iii) flexible networking; (iv) quality of service/experience; (v) security and privacy. Due to its benefits over Cloud-based architectures, especially if time is a critical issue or Internet connectivity is poor or absent, Fog Computing is expected to play a key role in the deployment of IoT applications. The Fog is not intended to replace the Cloud, but rather to complement it, in order to provide location-aware and real-time services, thus enabling new applications that could have not been deployed otherwise. Characteristics features of Fog Computing are the following:

- Geographically distributed, in contrast with a fully Centralized solution associated with Cloud-based architectures;
- Low latency and high performance to support real-time applications;
- Support for mobility.

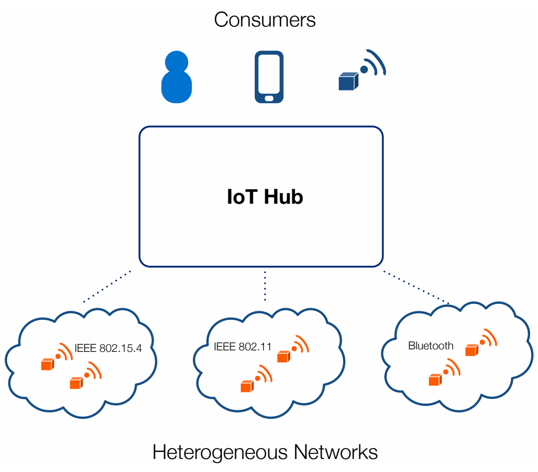


Figure 1. The Internet of Things (IoT) Hub can manage multiple networks of heterogeneous smart objects and enables the access to resources by external consumers that should not be aware of low-level details of communication.

According to the principles of Fog Computing and the fact that it will represents the reference architecture for IoT, the IoT Hub represents a critical building block for the deployment and manageability of complex IoT scenarios. The IoT Hub will allow Smart Objects to interoperate at a local level and to enable the design of new applications and interaction patterns of the Edge (e.g., analytics, machine learning, and human-to-machine interactions). In this context and architectural vision, the role of the IoT Hub becomes central in the foreseen IoT architecture, since it should process all requests providing the following functionalities and benefits:

- Independence from Internet Connectivity allows to guarantee continuous service operation at the local level with no disruption in case of unreachability of Cloud. In case Internet Access is temporarily unavailable, the IoT Hub will keep the state of all Smart Objects and will automatically synchronize with the Cloud as soon as the connection resumes;
- Controlled and secure access to things. Direct access to Smart Object occurs only at the local level and may be filtered by the IoT Hub Border Router: the IoT Hub is the gateway/bridge between one or more constrained networks (e.g., IEEE 802.15.4);
- Service and Resource Discovery: the IoT Hub is able to discover which SOs are available in the network and, subsequently, to discover the resources that they host;
- Resource Directory (RD): the IoT Hub maintains a list of all CoAP resources available in the constrained networks;
- Origin Server (OS): it provides a CoAP server where resources are hosted or are to be created;
- Protocol Translation: (i) CoAP-to-CoAP (C2C) Proxy: it provides proxying capabilities for CoAP requests coming from external clients that should reach internal constrained nodes; and (ii) HTTP-to-CoAP (H2C) Proxy: it provides HTTP-to-CoAP cross-proxying (i.e., protocol translation) in order to enable access to CoAP resources by HTTP clients;
- Cache: in order to avoid unnecessary load on SOs and to minimize latencies, a cache is kept with the representation of most recently accessed resources.

In order to increase the robustness of an IoT Hub-oriented architecture, in this work we propose a Cloud-supported replication mechanism for IoT Hubs in order to efficiently manage CoAP resources in a scalable and secure way. The proposed mechanism relies on the possibility to exploit Cloud platforms to clone and virtualize IoT Hubs. Replicas are full copies of IoT Hubs, thus implementing all their functionalities, and can thus be accessed on behalf of actual physical nodes. The proposed approach introduces several benefits: (i) a common Cloud-based interface for accessing available resources; (ii) the behavior, the logic and the implementation of the IoT Hub are hidden from communication with the aim to protect the IoT Hub and the smart objects behind it; (iii) the introduction of Cloud balancing policies in order to scale up or down the number of replicas according to needs, depending on the number of incoming/estimated requests and managed resources.

This work focuses on the definition, implementation, and evaluation of all functionalities (standardization, communication, synchronization, and caching) that the IoT Hub will provide in order, on the one hand to tackle the complexity of handling heterogenous Smart Objects making them available for applications running on the Edge and, on the other hand to provide an efficient synchronization with the Cloud for remote interactions.

The rest of this paper is organized as follows. In Section 2, an overview of related work, with focus on Fog Computing principles and its integration with the IoT is presented. Section 3 describes the proposed Cloud-based resource management architecture. In Section 4, an extensive experimental analysis of the proposed architecture is presented and the corresponding performance is investigated in a meaningful scenario. Finally, in Section 5 we draw our conclusions.

2. Related Work

The role of Cloud Computing in the IoT is gaining greater and greater attention. Most research has been so far smart object-driven, focusing mainly on the definition of IP-based, low-power, and efficient communication protocols and mechanisms. Many of these aspects have now been significantly addressed. Several IoT solutions have been deployed and brought to the market in several application scenarios, from Home Automation to Smart Cities. Most of these fragmented and vertical solutions rely on the Cloud, in order to provide a centralized access to services exploiting data that are sent uplink from deployed sensors to Cloud storage. Typically, mobile apps "consuming" such services are made available to end-users. However, this approach, which is expedient to disseminate the concept of IoT, does not fully exploit the potential of the Cloud.

As billions of smart objects are expected to be deployed pervasively, efficient data processing has highlighted the need to rely on the Cloud. The Cloud of Things (CoT) refers to the interaction between IoT and the Cloud [7]. In [8], an architecture for integrating Cloud/IoT is proposed, based on a network element, denoted as "Smart Gateway", which is intended to act as intermediary between heterogeneous networks and the Cloud. The role of the Smart Gateway is similar to that of the IoT Hub, in terms of supporting several heterogeneous networks. However, the role of the Cloud is mainly envisioned as a data storage and aggregator, which can be used by end-users to access data. According to this approach, data are sent uplink, making it impossible to directly address and act on smart objects, as the IoT is supposed to do. At the opposite, in the current paper we envision that the Cloud, by hosting replicas of the IoT Hub, is also used as an enabler for direct and efficient access to resources, while providing desirable features, such as: seamless access by external clients; security; and high availability. In [9] the concept of Smart Service Proxy is presented and described. It propose an interesting solution combining the use of IoT protocols and technologies with Semantic Web solutions. The proxy provides a unified access to resources/nodes through Proxying and Caching solution and introduce a dedicated Directory Service dedicated for the management of object representations, queries and ontology.

Fog Computing brings a new approach to Internet access networks by making computation, storage, and networking resources available at the edge of access networks. This improves the performance, by minimizing latency and availability, since resources are accessible even if Internet access is not available [10]. Fog-based solutions aim at introducing an intermediate architectural layer where resources and applications are made available in the proximity of end devices, thus avoiding continuous access to the Cloud.

Fog-based access networks are based on the presence of highly specialized nodes, denoted as Fog Nodes, able to run distributed applications at the edge of the network. In particular, the deployment of computing resources on Internet access networks allows to dinamically activate Virtual Machines (VMs) dynamically on Fog Nodes. For this reason, the cloning and synchronization techniques of VMs at the core of this work fit perfectly into Fog-based infrastructures, as will be discussed in more detail in Section 3. The proposed architecture can protect local resources by providing remote access to their replicas in a transparent way. Local resources are kept synchronized by multiple clones of the same machine, thus achieving a high level of reliability and load balancing. Smart management of the activation/deactivation of replicas and choice of the most appropriate Fog Node to run the clone allows to optimize the usage of CPU and memory available on the infrastructure, according to the specific real-time resources requirements by running applications.

The adoption of Virtualization Technologies for Wireless Sensor Networks has been studied and evaluated in several aspects during last years. In [11] the researchers present the implementation of a testbed where physical, simulated, and emulated sensor nodes cooperate and interact in real time extending the experimentation capabilities of a deployment. With the arrival of the IoT revolution, this technologies have been enriched with new features and functionalities in order to augment the proposed platforms and solutions. For example the authors of [12] introduce an interesting and innovative platform supporting different aspects of Wireless Sensor Network virtualization in order to handle the heterogenous requirements typical of IoT applications.

In [13] the authors present a scalable virtual sensor framework to support the build of logical data flows related to either physical sensors or custom virtual sensors. Although the presented approach is interesting, the presented technology is not applied on the Edge to handle synchronization between local and remote environment. The authors of [14] propose the use Virtual IoT Devices on the Edge for local data processing, management of physical devices, and quick actuation. The Cloud is envisioned only as a way to access the Edge infrastructure remotely but without the introduction of Gateway and/or Objects replicas and the related synchronization procedures.

A suitable lightweight alternative to VMs is represented by containers, which provide a more flexible environment for "disposable applications", like the IoT Hub. Container platforms like Docker [15–17] are gaining increasing attention also for edge and Fog computing applications. In this challenging scenario, the possibility of moving from centralized to decentralized paradigm to offload the processing to the edge reducing application response time and improving overall user experience will play a fundamental role in Internet of Things. In [18,19], the authors present how a container-based architecture could be efficiently used for dynamic networking application. In [20], an interesting comparison about existing lightweight and hypervisor-based approaches for edge computing and efficient networking is presented. Furthermore, in [21] a novel approach for the application of a lightweight virtualization technology (such as Docker) to constrained devices with a negligible overhead is presented. In this work, a containerized version of the IoT Hub, based on Docker, will be considered.

The authors of [22] present the use of virtualization techniques in the context of IoT Gateway management. In particular, they introduce the concept of Gateway-as-a-Service, a lightweight instance that can be shared between different users thanks to the use of virtualization techniques. This interesting work is focused on the adoption of virtualization techniques (e.g., Virtual Machines and Containers) on the Edge but does not take into account the virtualization and synchronization of gateways and Smart Objects with the Cloud and how this procedure can be seamless for users and services both on local and remote environments.

3. Architecture

The purpose of the work is to propose an architecture based of a locally deployed network element (the IoT Hub) for the seamless and ubiquitous access to IoT resources: (i) within the same local network where Smart Objects are deployed; or (ii) externally through a Cloud service with high availability (Smart Object virtual replicas). This work provides the details also about the synchronization mechanisms between the state of resources maintained by the IoT Hub and their Cloud replicas. This solution will enables remote access to resources in networks in a fully transparent and standardized way. In order to achieve our goals, a new application layer for IoT networks is designed and developed. In particular, we focus on the design of an architecture that allows the access by external clients, by virtualizing the functionalities of an IoT network. The details of the IoT network, such as its location or its actual implementation, are kept hidden from users and external clients willing to access resources. In other words, resource access by remote clients will be mediated by the Cloud platform, which provides a standard and secure front-end.

3.1. The IoT Hub

As anticipated in Section 1, the proposed architecture is based on a network element component, denoted as "IoT Hub". The IoT Hub does not have the same strict requirements on energy consumption and processing capabilities as other smart objects and is thus expedient to provide relevant features to a constrained network. The IoT Hub is placed at the edge of the constrained network and plays a fundamental role by implementing the following functions, as summarized in the functional plane, mapped in the protocol stack, shown in Figure 2.

- Service and Resource Discovery: the IoT Hub is able to discover which SOs are available in the network and, subsequently, to discover the resources that they host;
- LoWPAN Border Router: at the network layer, the IoT Hub is the gateway between one or more constrained networks (e.g., IEEE 802.15.4) it belongs to (through some radio interfaces it is equipped with).
- CoAP/CoAP (C2C) Proxy: at the application layer, it provides proxying capabilities for CoAP requests coming from external clients that should reach internal constrained nodes.
- HTTP/CoAP (H2C) Proxy: at the application layer, it provides cross-proxying (protocol translation) between HTTP and CoAP in order to let external HTTP clients access CoAP resources hosted by smart objects.
- Resource Directory: the IoT Hub maintains a list of all CoAP resources available in the constrained network. These resources may have been gathered through several mechanisms, such as those described in [23].
- Cache: in order to avoid unnecessary load on smart objects and to minimize latencies, a cache is kept with the representation of most recently accessed resources.
- Replica Manager: a software module responsible for the coordination and the synchronization between the IoT Hub and its replicas.

Moreover, due to the constrained nature of smart objects, they cannot typically implement strong security mechanisms and access policies, for instance, those related to authorization, whereby the IoT Hub may act as a filter for incoming requests in order to limit access to specific resources. Because of all the functionalities outlined above, the IoT Hub may suffer from the following critical issues, which may undermine the lifecycle of a constrained IoT network:

- the IoT Hub is a bottleneck of the architecture, since all traffic must pass through it even if it is not a communication endpoint;
- failure of the IoT Hub would make the resources hosted by smart objects temporarily or permanently unavailable.

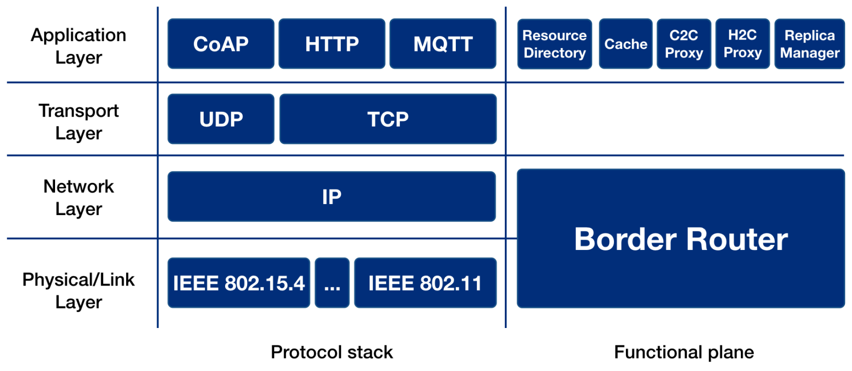


Figure 2. Protocol stack and functional modules implemented by IoT Hub.

It is necessary to relieve the IoT Hub from some of this load in order to guarantee that resources can be accessed with high availability in a secure and seamless way.

In this work, we propose to rely on the Cloud by providing virtual replicas of the IoT Hub. Replicas of the IoT Hub are fully functional clones, which may be used by any external client to access resources in the same way as they would do with the actual IoT Hub. Interacting with resources through the replicas allows to provide an access point different from the real (physical) IoT Hub, thus decoupling the constrained network management function and granting access to external clients. Replicas of the IoT Hub are synchronized through a dedicated MQTT-based protocol, which is used to transfer copies of the resources from the IoT Hub to the replicas in a pub/sub model. Replicas can be instantiated on-the-fly, according to particular needs, thus also providing load balancing (according to policies which may depend on the number of connected clients and smart objects) and acting as recovery facilities, in the presence of temporary failure of the real IoT Hub.

3.2. Operational Scenarios

Resource access through the proposed Cloud-based platform can occur according to the following three operational models implemented by a smart object: (i) CoAP server; (ii) observable CoAP server; and (iii) CoAP client.

3.2.1. Polling Resources

If the smart object is a CoAP server, it can receive requests to access its hosted resources. In this case, shown in Figure 3, the message flow is as follows: (1) the external client sends a HTTP or CoAP request to the Cloud platform front-end, which (2) forwards the request to one selected replica of the IoT Hub.

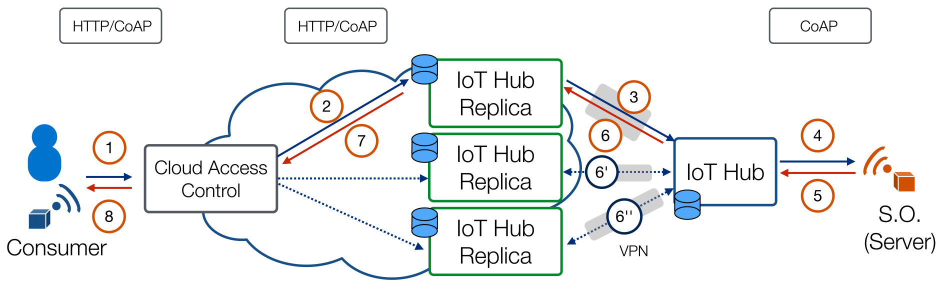


Figure 3. Message flow for the Polling scenario: a HTTP/Constrained Application Protocol (CoAP) client requests a resource to the Cloud platform, which internally selects a suitable IoT Hub replica and forwards the request. The request reaches the smart object only if neither the replica nor the IoT Hub have stored a fresh cached representation of the resource. Other replicas (which are not in the path of the request) are kept in sync using the synchronization protocol.

The CoAP protocol [4] defines the Max-Age option to indicate the maximum time a response may be cached before it is considered not fresh. Since Smart Objects, the IoT Hub, and Virtual Replicas all implement CoAP they are able invalidate their caches autonomously based on this information.

If the replica has a matching cached "fresh" resource, it can return it immediately; otherwise, (3) the replica forwards the request to the actual IoT Hub (which is securely connected through a VPN tunnel): if the IoT Hub has a matching cached fresh resource, it can return it immediately; otherwise, (4) it acts as a reverse proxy and forwards a (and, if needed, translated) CoAP request to the CoAP server, which (5) returns the resource. At this point, the returned resource is cached by the IoT Hub, and (6) returned to the replica, which, in turn, (7, 8) sends it back to the client. The resource cached at the IoT Hub is then ($6'$ and $6''$) synchronized with its replicas, in order to speed up and efficiently manage subsequent requests targeting the same resource.

3.2.2. Observing Resources

The Observe option [24] is a CoAP option which allows resource observing. According to this specification, a CoAP client can send a single request, including an Observe option with a value of 0 (register), in order to register its interest in receiving updates related to the targeted resource. The CoAP server sends a response each time the resource value is updated. In this case, multiple responses are sent after a single request. The Observer option allows to implement a notification-based communication model, thus reducing network traffic. The observing scenario is shown in Figure 4. If the targeted smart object is a CoAP server which implements the Observe option, it can receive requests to access its hosted resources, which may be either observable or not. In the latter case, requests are handled as in the polling case. In the former case, instead, the message flow resembles that of the polling scenario, but (i) the IoT Hub observes the resource and (ii) the external client observes the cached resource on the replica. When the resource is updated, (i) the smart object will send a notification to the IoT Hub; (ii) the IoT Hub will synchronize the resource with its replica; and, finally, (iii) the replica will send a notification to the external observing client.

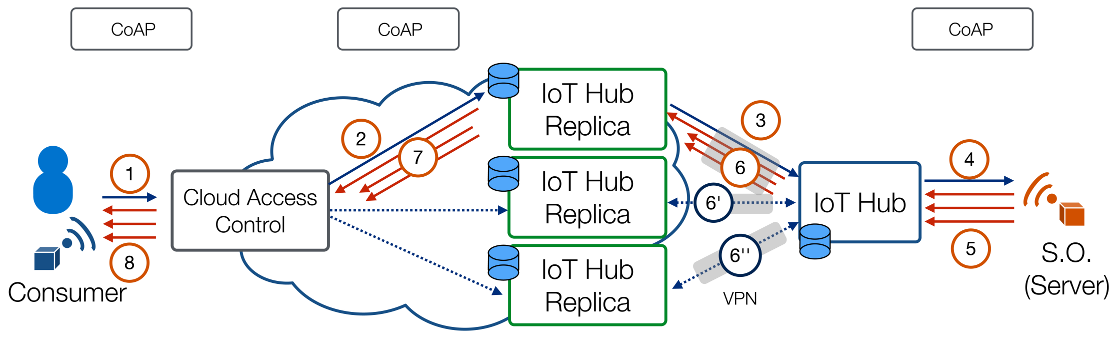


Figure 4. Message flow for the Observing scenario: a CoAP client requests a resource to the Cloud platform using the CoAP Observe option. The Cloud platform internally selects a suitable IoT Hub replica and forwards the request. The observe request is then forwarded to the IoT Hub and to the smart object thus creating an "observe chain". Resource updates are then sent from the smart object back to the IoT Hub, then to the replica, and finally to the CoAP client. Other replicas (which are not in the path of the request) are kept in sync using the synchronization protocol.

3.2.3. Pushing Resources

Sometimes, the memory constraints of smart objects make it unfeasible to let them act as CoAP servers. In this case, described in Figure 5, the smart object acts as a CoAP client and sends CoAP requests (POST and PUT) to the IoT Hub, which plays the role of origin server, by maintaining resources on behalf of the clients. If the smart object is a CoAP client, according to the semantics of CoAP methods, the following message flow takes place: (1) the smart object sends CoAP POST requests to the IoT Hub in order to create resources and CoAP PUT requests in order to change their value. When the IoT Hub receives POST and PUT requests, after properly handling these requests it accordingly stores them and (2, 2′ and 2″) synchronizes them on the replica. When (3) an external client sends an HTTP or a CoAP request to the Cloud platform front-end, it (4) the latter forwards the request to one selected replica of the IoT Hub. Since the replica is synchronized with the IoT Hub, (5, 6) the replica can respond immediately with the stored representation of the resource.

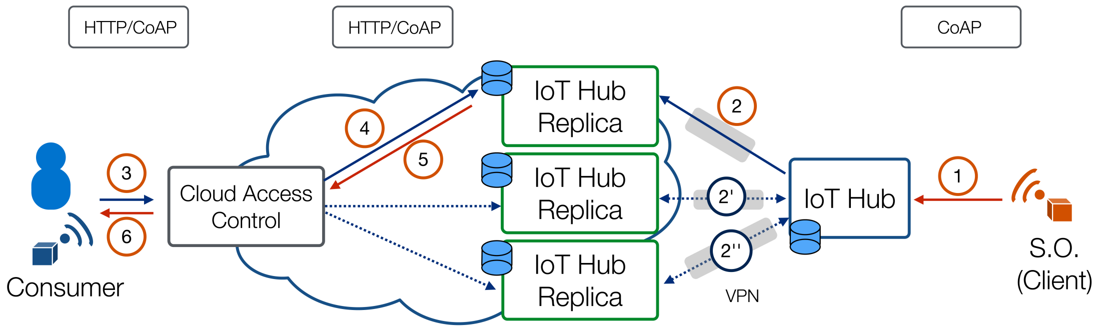


Figure 5. Message flow for the Pushing scenario: a smart object acting as a CoAP client posts and updates resources on the IoT Hub, which acts as origin server. All replicas are kept in sync using the synchronization protocol. External client can request resources, which will served by a replica.

3.3. Synchronization Protocol

The diagram presented in Figure 6 illustrates the operations and involved message exchange during the synchronization phases between involved actors. The operations related to the IoT Hub are detailed as follows:

- The IoT Hub discovers existing Smart Objects that are deployed in the same network using multiple service discovery mechanisms such as ZeroConf [25,26] and UPnP [27] and the list of all the resources that they host.
- The IoT Hub starts observing all the discovered resources, according to [24] to receive updates related to state changes.
- Any time the IoT Hub receives such updates it populates/refreshes its resource cache.
- The IoT Hub publishes a message (using an MQTT topic bound to the resource and the IoT Hub) related to the resource state change to the Cloud Broker to let Virtual Replicas update their mirrored state.

The operations related to the Virtual Replicas are detailed as follows:

- Upon launch the Virtual Replicas subscribe to the Broker to receive updates for topics related to a target IoT Hub.
- Any time the IoT Hub publishes an update for a resource the Virtual Replica will receive a message and will update its state accordingly.

Client applications that wish to interact with resources operate in the following ways:

- Local Clients issue CoAP requests directly to the Smart Objects (if it implements CoAP) or to the IoT Hub otherwise.
- Remote and External Clients may interact with a Smart Objects only through the Virtual Replica which will be publicly accessible using its REST interface.

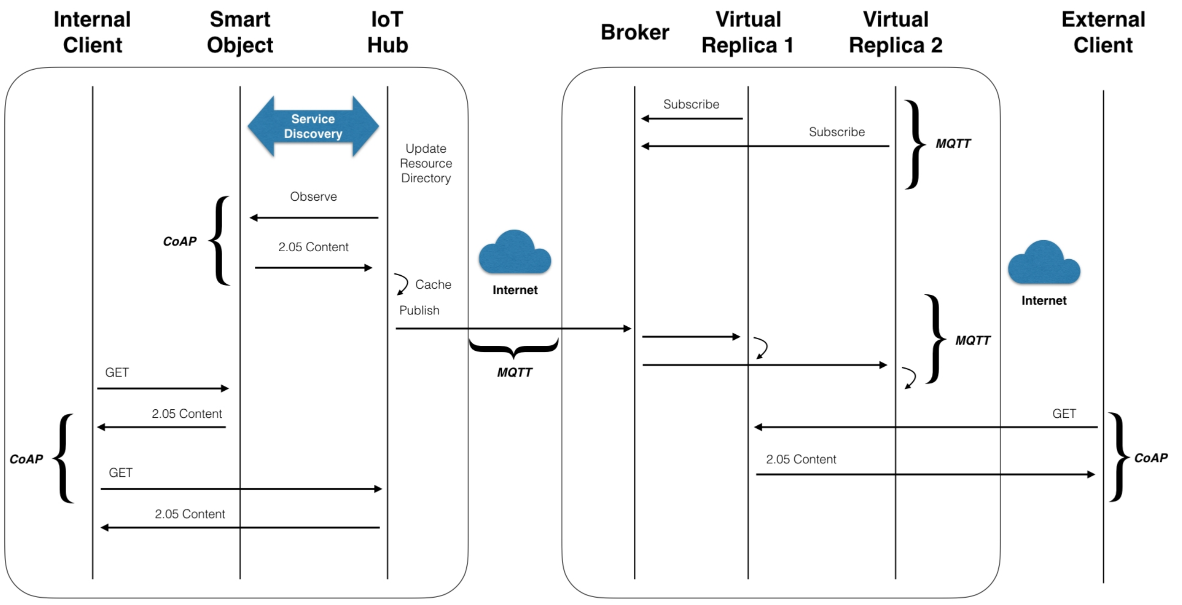


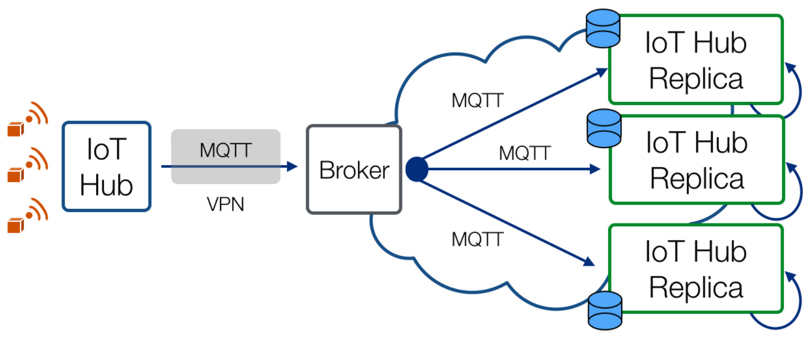
Figure 6. Sequence Diagram illustrating the operation and message exchange between all actors involved and the communication protocols used at each phase

The synchronization protocol used in the architecture implements a pub/sub communication model. In fact, communication follows a one-to-many pattern from the IoT Hub to all of its replicas. All messages are sent by the IoT Hub to an MQTT message broker (hosted on the Cloud platform and illustrated in Figure 7a) using specific MQTT topics (which can be used to selectively target one, many, or all replicas in order to implement unicast, multicast, or broadcast communications respectively) that is managed by the Cloud platform, using a VPN connection, for security and addressing reasons, as shown in Figure 7a. The IoT Hub and its replicas are all identified by a system-wide identifier, assigned by the designed Cloud platform.

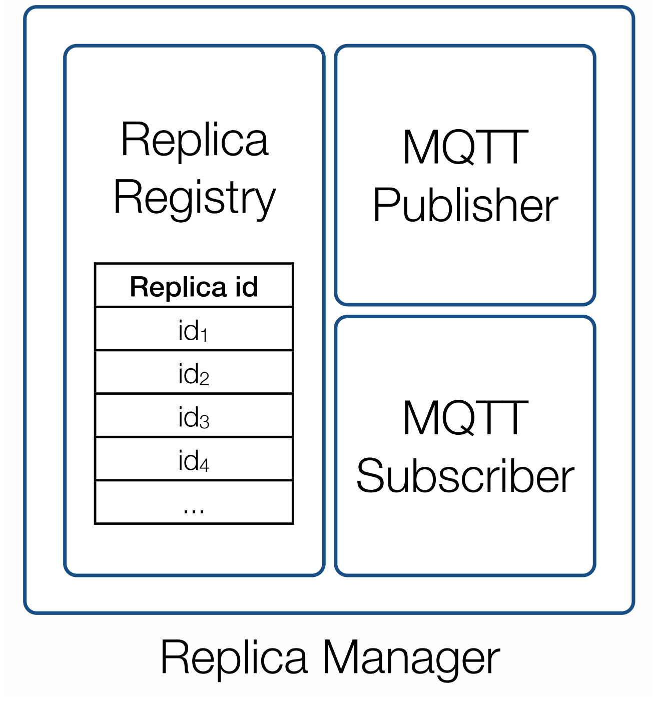
Each IoT Hub includes a Replica Manager (RM), which is a dedicated software module responsible for the synchronization among the IoT Hub and its replicas. The RM is composed by the following items, as shown in Figure 7b.

- A *Replica Registry* (RR), which contains the list of the identifiers of all the replicas of the IoT Hub.
- An *MQTT subscriber*, which registers to the broker to receive messages related to two topics: (i) its own identifier (id_i) and (ii) the identifier of the actual IoT Hub (id_{hub}); these may coincide in the case of the actual IoT Hub.
- An *MQTT publisher*, which publishes messages to the broker, using the method $pub(t, m)$, where t is the topic and m is the message to be published.

The IoT Hub is in charge of keeping full synchronization of its resources with its replicas, in order to ensure that all requests are served in the same way, regardless of the specific replica that was targeted by the client. Synchronization comes into play every time a resource on the IoT Hub changes. This can be caused by different events. At startup, a replica of the IoT Hub needs to synchronize with the actual IoT Hub. The procedure is shown in Figure 8. The RM of the replica publishes its id_i to the topic id_{hub}, in order to inform of its creation ($pub(id_{hub}, id_i)$). At this point, the IoT Hub updates its RR by adding id_i and then starts publishing to the broker all the resources (R) using the topic id_i ($pub(id_i, R)$), which guarantees that the new replica will receive the resources. When the synchronization procedure has ended, the replica will be automatically kept synchronized with the IoT Hub during the normal system lifecycle.



(**a**)



(**b**)

Figure 7. The broker-based message flow between the IoT Hub and its replicas is shown in (**a**), while the internal structure of the Replica Registry module of the IoT Hub is shown (**b**).

When resources are polled for (as will be described in Section 3.2.1), the IoT Hub might find out that a resource targeted by some requests has changed. A request targeting a resource that either has not been cached or is not considered fresh must be forwarded by the IoT Hub to the smart object. Upon receiving the response from the smart object, after updating its cache and forwarding the response to the requesting replica, the IoT Hub uses the synchronization protocol to publish the updated information to all of its replicas. When observing resources (as described in Section 3.2.2), the synchronization procedure resembles the same of the polling case. When resources are pushed by smart objects to the IoT Hub (as described in Section 3.2.3), the IoT Hub uses the synchronization protocol to publish the updated information at all replicas. Note that this synchronization strategy is needed only for those replicas that are part of the request/response loop: in fact, all resources are automatically synchronized with the request-issuing replica by design, since the replica also perfectly reproduces the behavior of the actual IoT Hub.

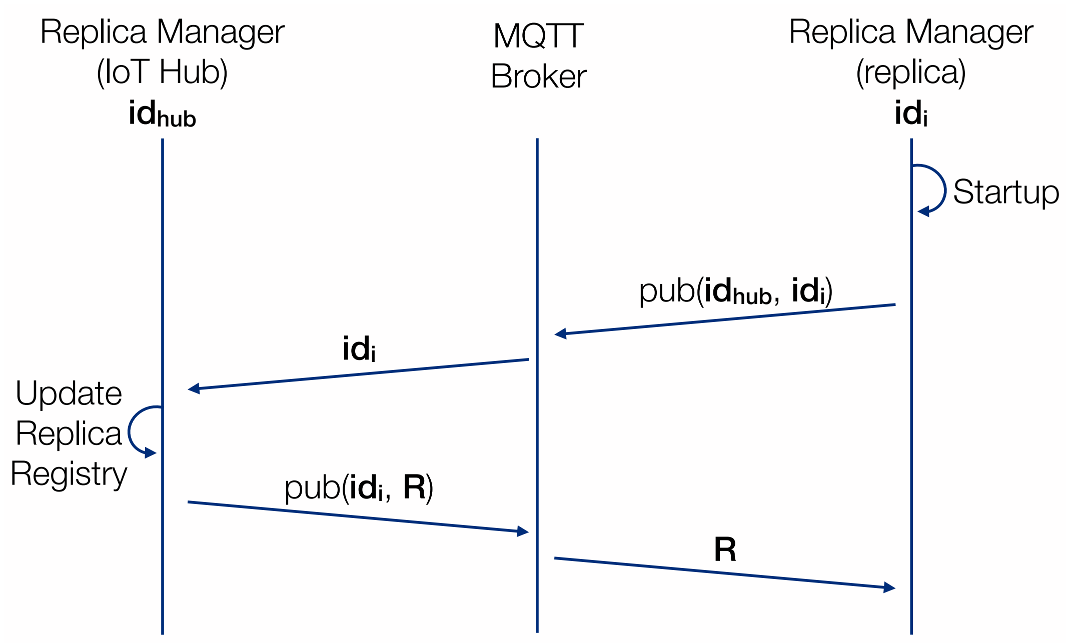


Figure 8. Synchronization procedure performed at startup of the replica of an IoT Hub.

Pub/Sub approaches for data synchronization may cause inconsistencies between the state of the IoT Hub and Virtual Replicas at some moments. This may be caused by the following:

- A lack of Internet connectivity with Cloud Virtual replicas: in this case there is no way to keep the states synchronized until the connection resumes;
- In a Pub/Sub architecture messages may be delivered to consumers at different times so that a Client requesting the state of a resource may receive different responses depending on the Virtual Replica serving the request. However, it is import to point out that all replicas are typically being executed in Cloud Environments/Carrier-grade Data Centers which guarantee high availability and reliable and low latency connectivity, thus minimizing the inconsistency time windows.

Our work relies on an Optimistic Locking approach which solves the inconsistency issue by versioning the state of resources and leaving clients responsible for managing race conditions. Consensus Algorithms (e.g., Raft or Paxos [28,29]) may introduce additional overhead whose trade offs should be carefully evaluated according to application requirements.

4. Experimental Analysis

In order to validate the feasibility of the proposed IoT architectural solution and to evaluate its performance, an extensive experimentation has been conducted. The evaluation focuses on the resource management on both local and remote IoT Hubs and the synchronization mechanisms described in Section 3.

4.1. Experimental Setup

The experimental setup has been designed and deployed with the aim of creating a realistic scenario with heterogeneous components and nodes in a local IoT network and the Cloud. The main components can be summarized as follows.

- *Smart objects*: Real and virtual nodes with CoAP modules based on the Californium framework [30].
- *IoT Hubs*: Raspberry Pi Model B [31] or independent VM instance running all the functional modules presented in Section 3 (Resource Discovery, Proxy CoAP/CoAP and HTTP/CoAP, Border Router, Cache and Replica Manager).

- *Virtualization Platforms*: Four different virtualization configurations on both local and Cloud platforms are considered: (i) Microsoft Azure [32]; (ii) Amazon EC2 [33]; (iii) Open Stack [34] on Microsoft Azure; and (iv) Open Stack on a local physical machine.
- *Resource External Consumer*: Real and virtual external consumers implementing HTTP and CoAP modules to dynamically interact and consume available resources managed by the platform and active IoT Hubs and smart objects.

We have configured and tested multiple virtualization configurations in order to evaluate the performance of the designed IoT architecture both on local and remote VMs. In particular, the Open Stack layer has been initially tested on a local installation at the Department of Information Engineering of the University of Parma and, at a later stage, on Microsoft Azure in order to obtain and measure more realistic results on a professional Cloud infrastructure. The local Open Stack installation runs on a physical machine with two 1.6 GHz processors and 3 GB RAM, while the Azure configuration is characterized by a Virtual Machine with four 2.0 GHz cores and 8 GB RAM. Both platforms have been used to dynamically manage replicas of active IoT Hubs and properly handle resource synchronization and remote data access. A virtual instance of an IoT Hub replica is characterized by an hardware profile with: a single core 2 GHz processor; 1 GB RAM; and a 8 GB disk space. The internal IoT Hub runs a Linux Ubuntu 14.04 LTS Operating System with SSH remote access, Oracle Java VM [35], and all the required functional software modules already installed and properly configured. In the experimental evaluation we consider a number of resources for each IoT Hub that is ranging from 5 to 100, which represent a reasonable number of resources associated to a single constrained Hub for example handling a floor of a Smart Building or a Parking Area in a Smart City.

The following key metrics are defined to measure the performance key values at different architectural layers.

- *IoT Hub Replica Creation Time* (dimension: [s]): the time required to create and run, on the target Cloud infrastructure, a new instance of an IoT Hub Replica.
- *Resource Synchronization Time* (dimension: [ms]): the elapsed time needed to synchronize a new resource between two IoT Hubs.
- *Resource Access Time* (dimension: [ms]): the time required to access and retrieve a response for a resource of interest. It can be associated with different configurations: (i) direct and local access to the CoAP Server smart object (e.g., if the consumer and the node are in the same network); (ii) remote access through the Cloud and communication with the physical Hub; (iii) remote access to the cached value stored on an IoT Hub Replica.
- *CPU Usage %* (adimensional: [percentage]): the percentage of CPU used by the IoT Hub core process.
- *Memory Usage %* (adimensional: [percentage]): the memory percentage used by the core process of the IoT Hub.

4.2. Performance Evaluation

The first phase of experimental performance analysis focuses on the evaluation of the amount of time required to create (from scratch) and run a new IoT Hub Replica instance on different virtualized Cloud infrastructures. The obtained results are shown Figure 9. Each value has been obtained averaging over 10 different VM creation runs with a confidence interval of 99%. It can be observed that: (i) the average costs on remote and professional Cloud infrastructures is comparable; and (ii) the cost is higher on local and not optimized solution (such as the Open Stack instance running in our department). We remark that the metric corresponds to the total amount of time required to create a new VM from scratch (starting from a pre-configured image and adding the time to start all the required services and architectural software processes). This cost should be considered only once for each IoT Hub Replica and is significantly written off with the increasing of the hub lifetime. Native VMs on Microsoft Azure and Amazon EC2 present approximately the same creation time, while the use of the Open Stack platform introduces a small delay associated with the additional overhead (of the platform itself) required to manage multiple s.

The second phase of the experimental analysis has been entirely focused on (i) the evaluation of the required time for synchronization of resources between two IoT Hubs (physical and virtual); and (ii) the time needed by an external consumer to access a target resource of interest in different scenarios and configurations. In Figure 10a, (i) the total amount of time required for the synchronization of a set of resources between an IoT Hub and its new Replica and (ii) the average time to synchronize a single resource in the same set are shown as functions of the number resources. The obtained results show that the average synchronization cost for a single resource is stable and, consequently, the total required time is proportional to the number of resources to be synchronized. Moreover, results show that for a reduced number of resources, the impact of the cost of MQTT connection creation is slightly relevant compared with the payload. Obviously, by increasing the number of resources synchronized using the same MQTT connection it is possible to reduce this effect and reach a stable value below 25 ms/resource.We expect that, by increasing the number of nodes the number of resources will increase "faster". However, the proposed framework automatically encompasses this aspect, as we consider the number of resources (i.e., the aggregate of the numbers of resources per nodes). The total synchronization time is a linearly increasing function of the number of resources.

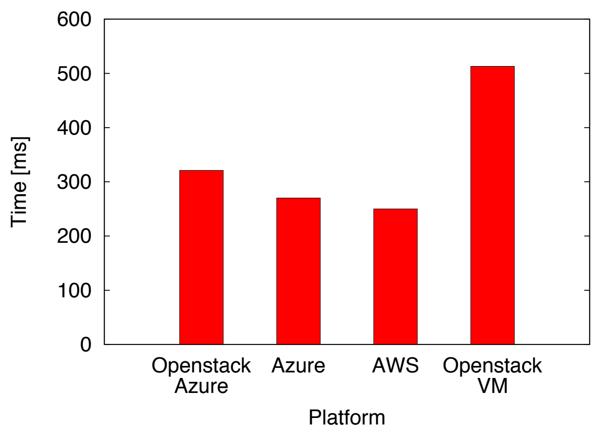
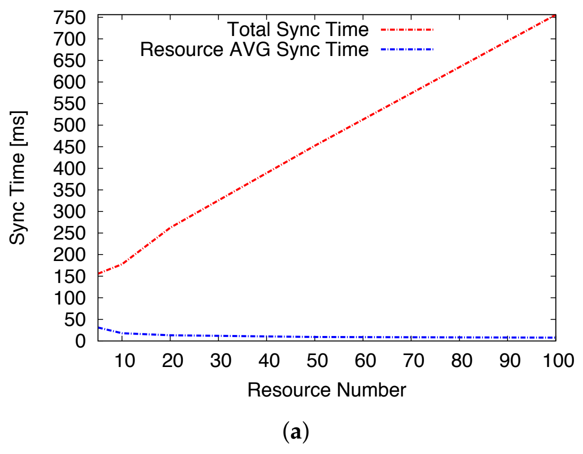


Figure 9. Average IoT Hub creation time (dimension: [s]) on different Cloud platforms.



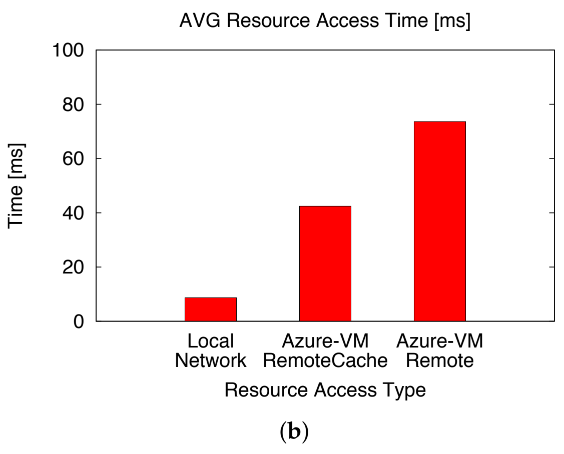


Figure 10. (**a**) Average synchronization time (dimension: [ms]) respect to the number of synchronized resources; (**b**) Average remote resource access time (dimension: [ms]) in different application scenarios.

In Figure 10b, the average time required by a consumer to access a resource of interest provided by a smart object is evaluated in different scenarios. In particular, we have considered three different configurations where: (i) the consumer is in the same local network of the target smart object; (ii) the external actor accesses a cached value on the active IoT Hub Replica on the Azure platform; and (iii) the consumer accesses a resource that is not cached on the IoT Hub Replica and, consequently, requires a direct communication between the virtual Replica and the real IoT Hub. The presented results show, as intuitively expected, that the quickest access is obtained if the consumer is in the same network and does not require additional communication with a remote infrastructure. However, if the consumer is an external actor the average cost, still considering a realistic deployment through Microsoft Azure, is below 80 ms/resource. This value decreases if we consider a resource that is cached on the Replica Hub, which does not require any additional communication with the local Hub and, eventually, with the smart object.

Our experimentation has also investigated the cost, in terms of CPU usage of an IoT Hub process, both on local (Raspberry Pi node) and remote (Microsoft Azure VM) instances. In Figure 11, the percentage of CPU usage during a run of 60 s of core activities is shown highlighting: (i) initialization of the main process and the creation of a set of 5 new resources on the local Hub; (ii) activation of the remote Replica and its initialization; (iii) synchronization of the group of 5 initial resources between local and remote replica Hub; and (iv) sporadic addiction of single resources until the end of the experiment. The obtained results show how the initialization of the Hub represents an intensive activity on both local and remote instances. The CPU usage incurs a significant one-time cost due to: setup the Hub configurations (such as node identification and software module selection); establishing the VPN connection activating the CoAP Server and the MQTT module (listener and publisher on specific topics); and starting up the Resource Directory. After the initialization phase, the CPU usage significantly reduces for both resource group synchronization and sporadic management of new single resources. These activities represent common and frequent tasks for an active IoT Hub that typically handles small variations of the set of managed resources during its lifetime. The main software process consumes a reduced amount of CPU (under 10%) for a small amount of time on both local and remote instances.

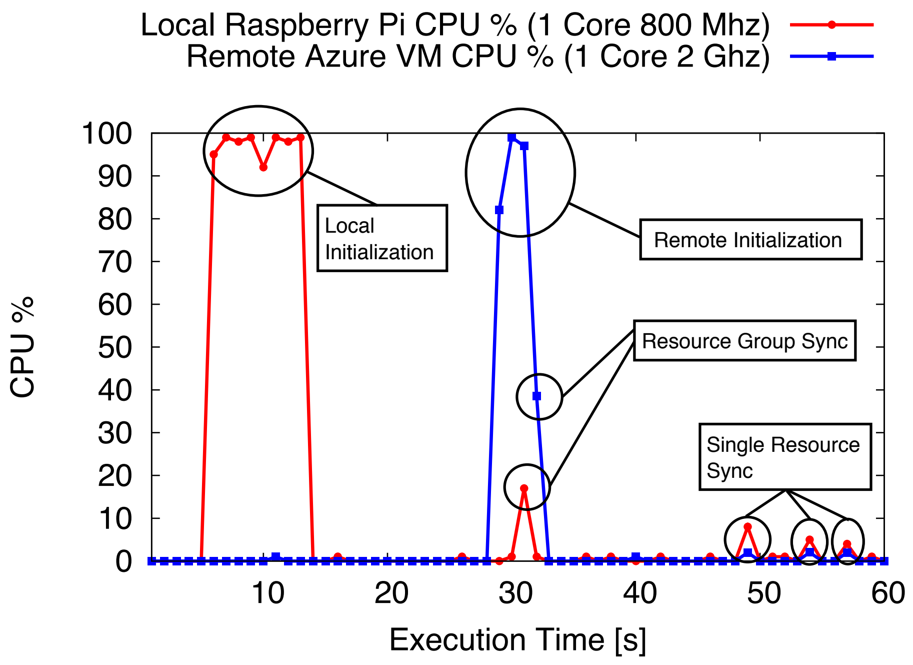


Figure 11. IoT Hub process CPU percentage usage (dimension: [adimensional]) on a local (Raspberry Pi node) and remote instance (Microsoft Azure VM).

In order to complete the presented analysis, Table 1 shows the average values (obtained through multiple independent runs and a confidence interval of 99%) of CPU and Memory usage related to each specific Hub procedure. The presented data confirm the cost distribution, with a percentage peak due to initialization phase and lower values for group and single resource synchronization. Memory utilization has been measured as the offset respect to the previous value and depends on the Java Virtual Machine memory management [36]. In particular, when an object is no longer used, the Java Garbage Collector reclaims the underlying memory and reuses it for future object allocation without an explicit deletion (no memory is given back to the operating system).

Table 1. Average CPU and Memory utilization percentages related to specific IoT Hub procedures on both local and remote instances.

Hub	Initialization	Res. Add	Sync Res. Group	Sync Single Res.
[Local] Raspberry Pi CPU	96%	17.45%	15.15%	6.93%
[Local] Raspberry Pi Memory	+2.7%	+0.02%	+0.01%	+0.002%
[Remote] Azure VM Remote CPU	97%	NA	34.22%	1.93%
[Remote] Azure VM Memory	+1.9%	NA	+0.01%	+0.001%

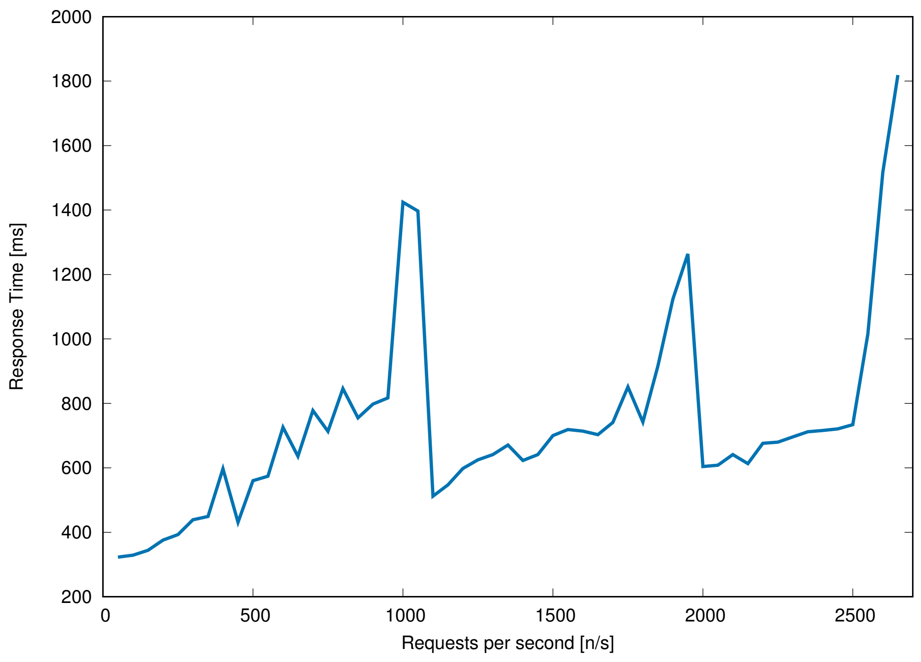


Figure 12. Effect of replica management with respect to the increasing number of requests per second on Microsoft Azure Infrastructure.

An important aspect for a truly scalable architecture is the ability to quickly react to a dynamic load, characterized by the rate of incoming requests that need to be served. The replication strategy proposed in this paper aims at providing a flexible and efficient management of IoT Hub replicas in order to guarantee that the response time remains below a given threshold. The virtualization approach provides a flexible solution to guarantee significant availability and efficient load balancing in highly dynamic IoT environments. In order to validate the designed replica management scheme, an additional experimental phase has been carried out. Leveraging the same Microsoft Azure Infrastructure used for all the other experiments, we measure the response time for incoming requests for Smart Objects resources managed by the IoT Hub. In Figure 12, the effect of replica management in investigated in terms of response time (dimension [ms]) as a function of the incoming requests rate. In particular, the results refer to the shortest value according to which a new replica of the IoT Hub is activated whenever the response time exceeds a threshold set to 800 ms. The graph clearly shows that the response time tends to increase linearly as a function of the rate of requests until the IoT Hub reaches a breaking point associated with the maximum number of requests it can handle. This is clearly visible in the areas of the graph characterized by steep slopes. When a slope is detected, we activate a new replica, which brings the response time back to a value that meets our operational requirements. As the request rate increases, new replicas are created. The average creation time of a IoT Hub replica is related to the maximum amount of resources generated by the nodes: once the number of resources reaches the processing limit of the IoT Hub, then a new replica is created. Therefore, if the number of nodes increases, this has anticipates the creation of a replica, but it does not influence the creation time. As shown in Figure 12, we stopped our experimentation after the activation of 3 replicas. It is worth noting that: (i) a new replica is activated almost periodically (every 800 ms, according to the set threshold) and (ii) the slope of the response time between two consecutive activations is inversely proportional to the number of active replicas.

The definition and the widespread adoption of container-based technologies have significantly changed and re-designed the way Cloud and Fog applications can be deployed and delivered to final users. In order to provide a thorough performance evaluation of the proposed solution, we have also created a container-based version of the IoT Hub using the Docker platform [15]. The container has the same networking configuration, features, and services running on the VM-based version but shares the Operating System Kernel and features with other running container instances. The local experimentation has been conducted using a Virtual Machine running Ubuntu 14.04, with one 2.0 GHz processor and 1 GB RAM, on top of which Docker 1.11 has been executed with the aim of evaluating the startup time of the dockerized IoT Hub. This low-end hardware profile, compared to realistic data center facilities, has been purposely chosen to show the small footprint of the IoT Hub.

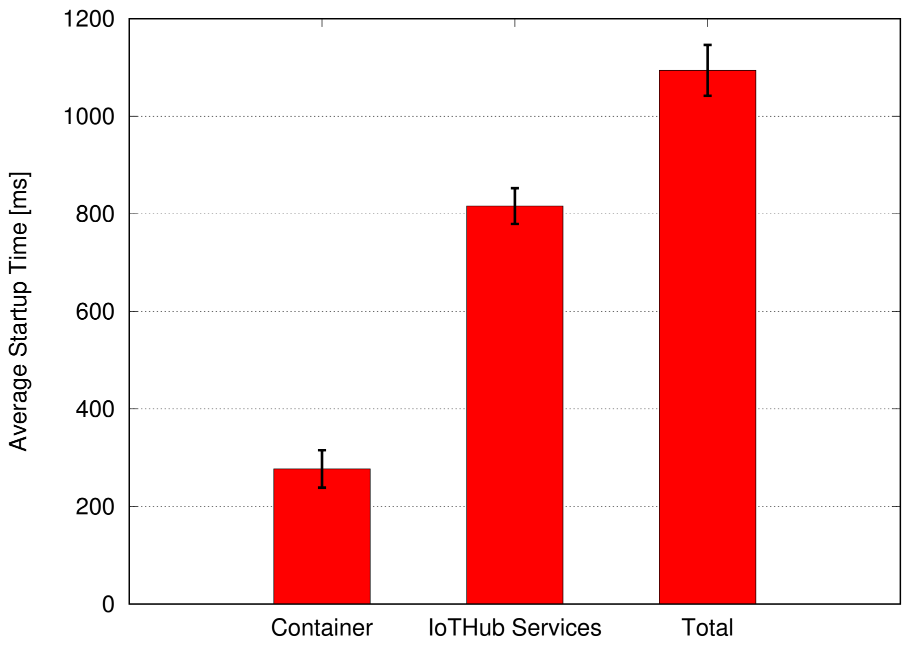


Figure 13. Average IoT Hub startup time (dimension: [ms]) on a Docker container.

In Figure 13, the average total startup time (dimension: [ms]) required to activate a fully operative IoT Hub instance, as well as the breakdown of container and IoT Hub services startup time, are shown. The presented results have been averaged on 1000 runs on the configured setup. The container startup time simply considers the activation of a Docker container instance. On top of this running instance, we measure the time required to activate all IoT Hub services and to respond to incoming HTTP and

CoAP requests. The results show how a container-based approach can efficiently be adopted both on Cloud or Fog infrastructures (according to the target application scenarios) to support efficient and highly dynamic creation and management of IoT Hub replicas on top of existing host machines. Unlike a VM-based approach, container-based IoT Hub instances can be instantiated and removed dynamically, depending on the instantaneous load and without affecting the host machine or requiring infrastructure-related efforts (such as file/template management, configuration and activation of a new machine).

5. Conclusions

In this paper, we have presented a novel Cloud-based architecture to efficiently manage resources in IoT scenarios. The proposed architecture relies on a network element, denoted as IoT Hub, which implements several functions, such as management of heterogeneous networks and connectivity protocols (e.g., low power Wi-Fi, IEEE 802.15.4, BLE), caching, and proxying. The IoT Hub is used to provide a transparent and seamless access to resources hosted by smart objects, by allowing users to ignore the low-level communication details and focus on resources. As the IoT Hub plays a central role in the management of smart objects, an architecture based on Cloud (or Fog) IoT Hub replicas has been proposed in order to provide a scalable and efficient management of resources and to protect the IoT Hub from extremely high processing loads due to concurrent requests and/or attacks. The use of replicas is also expedient to implement a load-balancing mechanism, as their number can be scaled according to the actual number of incoming requests, thus enabling a typical "pay-as-you-go" model of Cloud computing. The overall architecture has been entirely implemented using Open Source software libraries and relying on different Cloud platforms. An extensive experimentation has been conducted in order to prove the feasibility of the solution and evaluate its performance according to key metrics and under several conditions, in terms of number of managed resources, hardware used, and Cloud platform. The experimental results show that IoT Hub replicas can be created and synchronized with good performance, thus enabling an efficient and scalable management of resources, while providing several benefits, such as seamless and secure access to resources, load balancing on replicas, and protection of the real IoT Hub. In future work we will investigate how consensus algorithm may be integrated in the synchronization protocol to guarantee more reliable and consistent access to resources, which could be critical in case of real-time scenarios and high frequency updates. Furthermore, as introduced and presented in [37,38] we will evaluate the combination of advanced self-configuration and semantic Web technologies to the IoT Hub infrastructure in order to simplify the building and definition of new applications.

Author Contributions: M.P. and S.C. conceived and designed the idea, the architecture and the experiments; M.P. and S.C. developed the main Software modules involved in the platform. M.M. during his work in the IoT Laboratory of the University of Parma implemented additional Software components and performed the experiments; M.P., S.C., G.F. and M.M. analyzed the data; M.P. and S.C. with the supervision and coordination of G.F. wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bormann, C.; Ersue, M.; Keranen, A. *Terminology for Constrained-Node Networks*; RFC 7228 (Informational); Internet Engineering Task Force (IETF): Fremont, CA, USA, 2014.
2. Vasseur, J.P.; Dunkels, A. *Interconnecting Smart Objects with IP—The Next Internet*; Morgan Kaufmann: Burlington, MA, USA, 2010.
3. Falk, J.; Kucherawy, M. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*; RFC 6650; Internet Engineering Task Force (IETF): Fremont, CA, USA, 2012.
4. Shelby, Z.; Hartke, K.; Bormann, C. *The Constrained Application Protocol (CoAP)*; RFC 7252 (Proposed Standard); Internet Engineering Task Force (IETF): Fremont, CA, USA, 2014.
5. MQTT. Message Queue Telemetry Transport. Available online: http://mqtt.org/ (accessed on 1 March 2018).

6. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 13–17 August 2012; ACM: New York, NY, USA, 2012; pp. 13–16.
7. Aazam, M.; Khan, I.; Alsaffar, A.; Huh, E.N. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In Proceedings of the 2014 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 14–18 January 2014; pp. 414–419.
8. Aazam, M.; Hung, P.P.; Huh, E.N. Smart gateway based communication for cloud of things. In Proceedings of the 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, 21–24 April 2014; pp. 1–6.
9. Kleine, O. The smart service proxy—A middlebox for a semantic Web of things. In Proceedings of the 2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), Seattle, WA, USA, 22–25 June 2015; pp. 160–162.
10. Yannuzzi, M.; Milito, R.; Serral-Gracià, R.; Montero, D.; Nemirovsky, M. Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing. In Proceedings of the 2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Athens, Greece, 1–3 December 2014; pp. 325–329.
11. Coulson, G.; Porter, B.; Chatzigiannakis, I.; Koninis, C.; Fischer, S.; Pfisterer, D.; Bimschas, D.; Braun, T.; Hurni, P.; Anwander, M.; et al. Flexible Experimentation in Wireless Sensor Networks. *Commun. ACM* **2012**, *55*, 82–90.
12. Karkazis, P.; Trakadas, P.; Zahariadis, T.; Chatzigiannakis, I.; Dohler, M.; Vitaletti, A.; Antoniou, A.; Leligou, H.C.; Sarakis, L. Resource and Service Virtualisation in M2M and IoT Platforms. *Int. J. Intell. Eng. Inform.* **2015**, *3*, 205–224.
13. Kim-Hung, L.; Datta, S.K.; Bonnet, C.; Hamon, F.; Boudonne, A. A scalable IoT framework to design logical data flow using virtual sensor. In Proceedings of the 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Rome, Italy, 9–11 October 2017; pp. 1–7.
14. Datta, S.K.; Bonnet, C. An edge computing architecture integrating virtual IoT devices. In Proceedings of the 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), Nagoya, Japan, 24–27 October 2017; pp. 1–3.
15. Docker. Container Platform. Available online: https://www.docker.com (accessed on 1 March 2018).
16. Rufino, J.; Alam, M.; Ferreira, J.; Rehman, A.; Tsang, K.F. Orchestration of containerized microservices for IIoT using Docker. In Proceedings of the 2017 IEEE International Conference on Industrial Technology (ICIT), Toronto, ON, Canada, 22–25 March 2017; pp. 1532–1536.
17. Großmann, M.; Klug, C. Monitoring Container Services at the Network Edge. In Proceedings of the 2017 29th International Teletraffic Congress (ITC 29), Genoa, Italy, 4–8 September 2017; Volume 1, pp. 130–133.
18. Ismail, B.I.; Goortani, E.M.; Karim, M.B.A.; Tat, W.M.; Setapa, S.; Luke, J.Y.; Hoe, O.H. Evaluation of Docker as Edge computing platform. In Proceedings of the 2015 IEEE Confernece on Open Systems (ICOS), Bandar Melaka, Malaysia, 24–26 August 2015; pp. 130–135.
19. Xu, Y.; Mahendran, V.; Radhakrishnan, S. SDN docker: Enabling application auto-docking/undocking in edge switch. In Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, USA, 10–14 April 2016; pp. 864–869.
20. Ramalho, F.; Neto, A. Virtualization at the network edge: A performance comparison. In Proceedings of the 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Coimbra, Portugal, 21–24 June 2016; pp. 1–6.
21. Morabito, R. A performance evaluation of container technologies on Internet of Things devices. In Proceedings of the 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, USA, 10–14 April 2016; pp. 999–1000.
22. Morabito, R.; Petrolo, R.; Loscrí, V.; Mitton, N. Enabling a lightweight Edge Gateway-as-a-Service for the Internet of Things. In Proceedings of the 2016 7th International Conference on the Network of the Future (NOF), Buzios, Brazil, 16–18 November 2016; pp. 1–5.
23. Cirani, S.; Davoli, L.; Ferrari, G.; Leone, R.; Medagliani, P.; Picone, M.; Veltri, L. A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things. *IEEE Internet Things J.* **2014**, *1*, 508–521.

24. Hartke, K. *Observing Resources in the Constrained Application Protocol (CoAP)*; RFC 7641 (Proposed Standard); Internet Engineering Task Force (IETF): Fremont, CA, USA, 2015.
25. Cheshire, S.; Krochmal, M. *Multicast DNS*; RFC 6762 (Proposed Standard); Internet Engineering Task Force (IETF): Fremont, CA, USA, 2013.
26. Cheshire, S.; Krochmal, M. *DNS-Based Service Discovery*; RFC 6763 (Proposed Standard); Internet Engineering Task Force (IETF): Fremont, CA, USA, 2013.
27. Boucadair, M.; Penno, R.; Wing, D. *Universal Plug and Play (UPnP) Internet Gateway Device-Port Control Protocol Interworking Function (IGD-PCP IWF)*; RFC 6970 (Proposed Standard); Internet Engineering Task Force (IETF): Fremont, CA, USA, 2013.
28. Lamport, L. Paxos made simple. *ACM Sigact News* **2001**, *32*, 18–25.
29. Ongaro, D.; Ousterhout, J.K. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference, Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.
30. Kovatsch, M.; Lanter, M.; Shelby, Z. Californium: Scalable Cloud Services for the Internet of Things with CoAP. In Proceedings of the 4th International Conference on the Internet of Things (IoT 2014), Cambridge, MA, USA, 6–8 October 2014.
31. Raspberry Pi Foundation. Message Queue Telemetry Transport. Available online: http://www.raspberrypi.org/ (accessed on 1 March 2018).
32. Microsoft. Microsoft Azure-Cloud platform. Available online: http://azure.microsoft.com/it-it/ (accessed on 1 March 2018).
33. Amazon. Amazon EC2. Available online: http://aws.amazon.com/ec2/ (accessed on 1 March 2018).
34. Rackspace, NASA. OpenStack Cloud Software-Open Source Software for Building Private and Public Clouds. Available online: https://www.openstack.org/ (accessed on 1 March 2018).
35. Oracle Java VM. Java. Available online: https://www.oracle.com/java/index.html (accessed on 1 March 2018).
36. Dhurjati, D.; Kowshik, S.; Adve, V.; Lattner, C. Memory Safety Without Runtime Checks or Garbage Collection. In Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems, San Diego, CA, USA, 11–13 June 2003.
37. Pfisterer, D.; Romer, K.; Bimschas, D.; Kleine, O.; Mietz, R.; Truong, C.; Hasemann, H.; Kröller, A.; Pagel, M.; Hauswirth, M.; et al. SPITFIRE: Toward a semantic web of things. *IEEE Commun. Mag.* **2011**, *49*, 40–48.
38. Chatzigiannakis, I.; Hasemann, H.; Karnstedt, M.; Kleine, O.; Kröller, A.; Leggieri, M.; Pfisterer, D.; Römer, K.; Truong, C. True self-configuration for the IoT. In Proceedings of the 2012 3rd IEEE International Conference on the Internet of Things, Wuxi, China, 24–26 October 2012; pp. 9–15.

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).