Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 683425, 16 pages
http://dx.doi.org/10.1155/2015/683425

Research Article

Integration of Heterogeneous Devices and Communication Models via the Cloud in the Constrained Internet of Things

Floris Van den Abeele, Jeroen Hoebeke, Ingrid Moerman, and Piet Demeester

Department of Information Technology, Ghent University-iMinds, Gaston Crommenlaan 8/201, 9050 Ghent, Belgium

Correspondence should be addressed to Floris Van den Abeele; floris.vandenabeele@intec.ugent.be

Received 30 March 2015; Revised 2 August 2015; Accepted 5 August 2015

Academic Editor: Massimo Villari

Copyright © 2015 Floris Van den Abeele et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As the Internet of Things continues to expand in the coming years, the need for services that span multiple IoT application domains will continue to increase in order to realize the efficiency gains promised by the IoT. Today, however, service developers looking to add value on top of existing IoT systems are faced with very heterogeneous devices and systems. These systems implement a wide variety of network connectivity options, protocols (proprietary or standards-based), and communication methods all of which are unknown to a service developer that is new to the IoT. Even within one IoT standard, a device typically has multiple options for communicating with others. In order to alleviate service developers from these concerns, this paper presents a cloud-based platform for integrating heterogeneous constrained IoT devices and communication models into services. Our evaluation shows that the impact of our approach on the operation of constrained devices is minimal while providing a tangible benefit in service integration of low-resource IoT devices. A proof of concept demonstrates the latter by means of a control and management dashboard for constrained devices that was implemented on top of the presented platform. The results of our work enable service developers to more easily implement and deploy services that span a wide variety of IoT application domains.

1. Introduction

In the coming years more and more everyday objects are expected to be interconnected to the Internet, which will lead to a vast expansion of the Internet as we know it today. White papers released by Ericsson [1] and Cisco [2] estimate that the Internet will grow tenfold in the near future, with up to 50 billion connected devices by 2020. A considerable amount of these new Internet citizens will be so-called constrained devices. These are small, embedded, and low-cost devices that are purposefully designed for executing specific tasks such as monitoring the physical environment. For performing their tasks, they are often fitted with a microcontroller, sensors, actuators, a wireless transceiver, and an energy source. Due to their low cost, these devices are constrained in terms of processing power, communication capabilities, and energy budget. The widespread deployment and use of such constrained devices across a range of application domains (e.g., smart city, building control, logistics, and transportation) are expected to generate significant efficiency gains as well as driving new business, the combined effect of which is estimated to create 14.4 trillion USD in net value in the next decade [3].

Due to the IoT spanning such a wide range of application domains, there is a diversity of devices, protocols, network connectivity methods, and resulting application models on the market today. Within these IoT solutions some are legacy systems that rely on proprietary technology (sometimes suitably referred to as the Intranet of Things [4]), while others adopt more open IoT standards such as MQTT [5] and the IETF protocol stack for (constrained) IoT devices [6]. This diversity often results in the vertical silos seen today and hinders development of value-added services that use these low-resource IoT devices [7]. For example, in [8] the authors state that the logistics sector should move away from proprietary, stand-alone solutions that are not connected to the rest of the IoT to new platforms that combine various existing hardware and software solutions for end-to-end integrity control of supply chains. But even within one (standardized) protocol suite there are a number of different

communication and application design strategies available which are often tightly tied to the underlying use-case. For example, in logistics, separate communication strategies are necessary for battery-powered tracker devices (which are typically only intermittently connected to the Internet) and trackers with an abundant energy source (which can afford network connectivity for longer periods of time). Forcing IoT users (e.g., service developers and constrained devices) to support these different types of diversity is unfeasible as they typically lack the proper resources (e.g., know-how, time, and processing resources) to handle the specifics of the underlying constrained devices and networks. The goal of this work is to hide this wide range of diverse technologies, protocols, and applications models from IoT users.

As the demand for low-resource IoT devices is expected to rise, this problem is only expected to worsen in the future. Thus, it will become necessary to improve the integration of a wide variety of constrained devices in the IoT. Cloud computing is a suitable method, due to its availability, elasticity (improving scalability), and low-cost of computing resources [9]. Such a cloud-based software system is interesting because it can support different types of low-resource IoT devices by means of an adaptation layer and offer a uniform device abstraction that hides the diversity in devices, protocols, network connectivity methods, and application models from IoT users. By offering well-known interfaces via open standards protocols (RESTful and CoAP in this work) for this device abstraction, the later becomes significantly easier to integrate than the underlying constrained devices. The resulting design greatly improves integration of and service development for constrained IoT devices, while burdening neither the constrained IoT devices nor the service developers.

Our contributions in this paper are as follows. First we design, implement, and evaluate a cloud-based software architecture that enables the integration of heterogeneous low-resource devices in the Internet and more importantly into services. By offering a uniform CoAP interface on top of a virtual device abstraction, our approach is able to support highly heterogeneous devices as well as providing interoperability with legacy IoT devices that were built using proprietary technology. Secondly, we illustrate the feasibility of our developed architecture with a real-life deployment consisting out of constrained devices that embrace open standards and one representative example of a proprietary low-resource IoT device. The proof-of-concept demonstrates that the developed architecture supports a number of different communication models.

The remainder of this paper is structured as follows. First, a case study is presented in the following section to highlight some of the issues faced when developing applications on top of two industrial IoT systems. Section 3 details the research questions addressed by this paper. The next section introduces supporting technologies and other pieces of background information used in the remainder of this paper. Section 5 presents our approach for integrating heterogeneity devices in the constrained Internet of Things. Our proposed solution is evaluated in Section 6 in a wireless sensor network setup and via a real-life proof of concept. Section 7 presents the literature related to our work. Finally, the paper is concluded with possible topics for future work and our conclusions in Section 8.

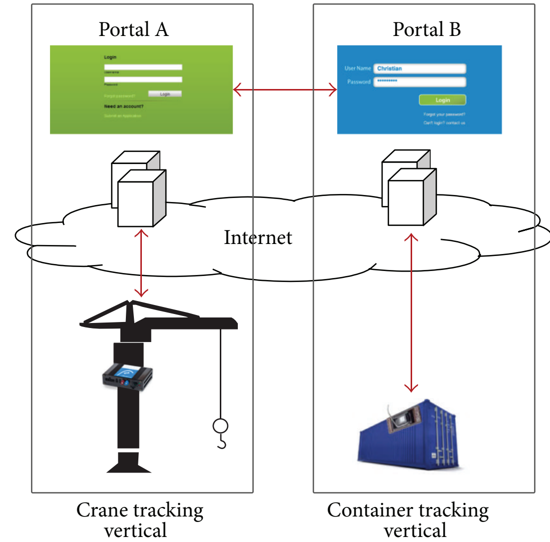


Figure 1: Isolated vertical platforms hinder cross-vendor service delivery.

2. Case Study: Logistics and Transport

The case study presented here considers harbor cranes and freight containers owned by different parties. The harbor cranes are tasked with (un)loading containers from cargo ships. Both the cranes and the containers are equipped with a GPS-enabled tracking device, each of which belongs to a different entity (i.e., vendors A and B). The trackers from vendors A and B each report to their separate back-end systems, where the vendor's customers can follow up on the status of their cranes or container's (contents) via a vendor-specific web portal (see Figure 1). The trackers are connected to the Internet via a GPRS connection.

Due to the container's mobility its tracker is battery-powered, whereas the crane's tracker is powered via the crane's alternator. Subsequently, the container's tracker is in sleep mode most of the time to conserve energy. In order to conserve the limited energy of the container's tracker even further, vendor B wants to update the location of its container by using the location information supplied by the crane when the container is picked up. This way the container's tracker avoids acquiring a GPS and a GPRS signal and transmitting its position, thus reducing its energy expenditure. However, both vendors only offer a web portal to their customers and there are no other interfaces for retrieving data from the trackers. Thus, vendors are forced to either rely on expensive and error-prone human intervention to interface between both systems or try and build on top of interfaces designed for user interaction.

Situations such as these are common in today's Internet of Things, with its abundance of isolated vertical platforms (e.g., in building management systems as per [10]). The costs of deploying and maintaining all the individual systems in

such verticals should not be underestimated as the reusability of components between verticals is typically low. As vendors start to expand their products into other IoT application domains (each with its own heterogeneous set of properties) and as IoT applications start to span across multiple domains (where everything will interact more and more with each other), this approach of building purpose-specific vertical silos will rapidly become inefficient and expensive.

3. Problem Statement and Research Goals

Before the problem statement is formulated, a number of examples of heterogeneity are presented here to illustrate the problems addressed by this work. Due to the wide range of environments where the IoT is considered to be employed [11], a number of different network connectivity technologies will be used depending on the specific use-case. For some applications, devices will be on a tight energy budget (e.g., battery-powered or energy harvesting devices) which might mean conserving energy by sleeping and remaining unreachable for long periods of times. In other applications mobility, remoteness or financial cost might lead to devices with an intermittently connected link to the Internet. And in other cases still, devices might be mains-powered and have a near always-on connection to the Internet. Furthermore, some low-resource devices might not be able to support certain transport protocols (e.g., TCP) and might have to follow a different approach (e.g., UDP or SMS). While in all these cases devices are accessible via a communications link, the properties and behaviour of this link are not always fully comprehended by service developers. Consider as an example a smart freight shipping container. As these containers are transported (often over long journeys), their Internet connectivity will vary: for example, there will be times when they are unreachable and when they are reachable, their Internet end point will change frequently due to their mobility.

Another cause of heterogeneity is the application communication model chosen by low-resource IoT devices. Note that this choice is often influenced by the available network connectivity as discussed in the previous paragraph. Some devices might rely on a "pull model," where the service is expected to initiate all interactions to and from the devices. Other devices, due to network constraints, might employ a "push" approach where devices periodically send data to a data store and sleep for the majority of the time. In this case, push data is often aggregated in order to increase the communication efficiency. In other cases, devices might employ a mix of push and pull; for example, critical events and monitoring data are sent immediately (e.g., for generating alerts), noncritical measurements are aggregated and pushed together, and configuration and management of the devices take place via a pull model (i.e., a management service might send configuration data to the IoT device directly).

Another fundamental cause of diversity is whether the IoT device employs proprietary or one of the many standards-compliant application protocols and data formats available today. While proprietary purpose-built technology will in some cases outperform standards-compliant technology, it also has a number of downsides, of which the most important for the discussion here is that they are difficult to integrate for third parties. Furthermore, proprietary protocols often lead to higher development and maintenance costs and are less future-proof than open standards. For our smart freight container example, this is also what we see on today's market [8]. There are a number of players that offer tracking services for freight containers, but their systems are using private communication networks and in-house data standards and typically only offer high-level access to data (e.g., via a web-based dashboard for tracking). This greatly impedes other players to integrate smart containers into their products. However, even when relying on standards-compliant technology, there are many different standards available today. Most of these are popular in specific domains, such as BACnet in building automation, ZigBee in home automation, and SEP2.0 in smart metering. Thus, services that want to encompass multiple application domains are forced to interface with a mix of proprietary and domain-specific standards, technologies, and data formats.

A final source of heterogeneity that is addressed by this work, which is related to the diversity of application protocols and data formats, is the wide variety of supporting services for interacting with constrained devices. Every technology comes with its own mechanisms for device and data source discovery, its own security concepts and methods (if available at all), its own notification service, and so forth. As a result there is little reuse between these technologies, which further impedes cross-technology integrations. Moving to open standards will allow using standard-compliant approaches for such supporting services and will provide uniform mechanisms for these services to third party service developers. While the list of heterogeneity presented here is not meant to be exhaustive it does present a clear overview of the types of heterogeneity that this paper considers. A comprehensive overview of the different types of heterogeneity commonly found in wireless sensor networks is presented in [12].

Parties looking to offer new services in this mix of connectivity, application models, standards, and protocols will quickly find themselves forced to integrate a multitude of different technologies. Given the previous paragraphs and their conceptual representation in Figure 2, it becomes clear that supporting all these different types of heterogeneity cannot be expected from service developers. Furthermore, constrained devices are unable to adapt to the different service providers due to their low resources. These devices will typically implement one of many instances of heterogeneous technologies presented here, depending on availability of resources, communication, energy, and application requirements. While cloud-based platforms are a good match for complementing constrained low-resource devices [9], a gap analysis of existing IoT platforms [13] shows that support for heterogeneous and constrained devices in such platforms is still lacking. One of the problems identified by Mineraud et al. [13] is that most IoT platforms assume constrained devices to support HTTP, which, given the heterogeneous nature of these devices, is definitely not always the case.

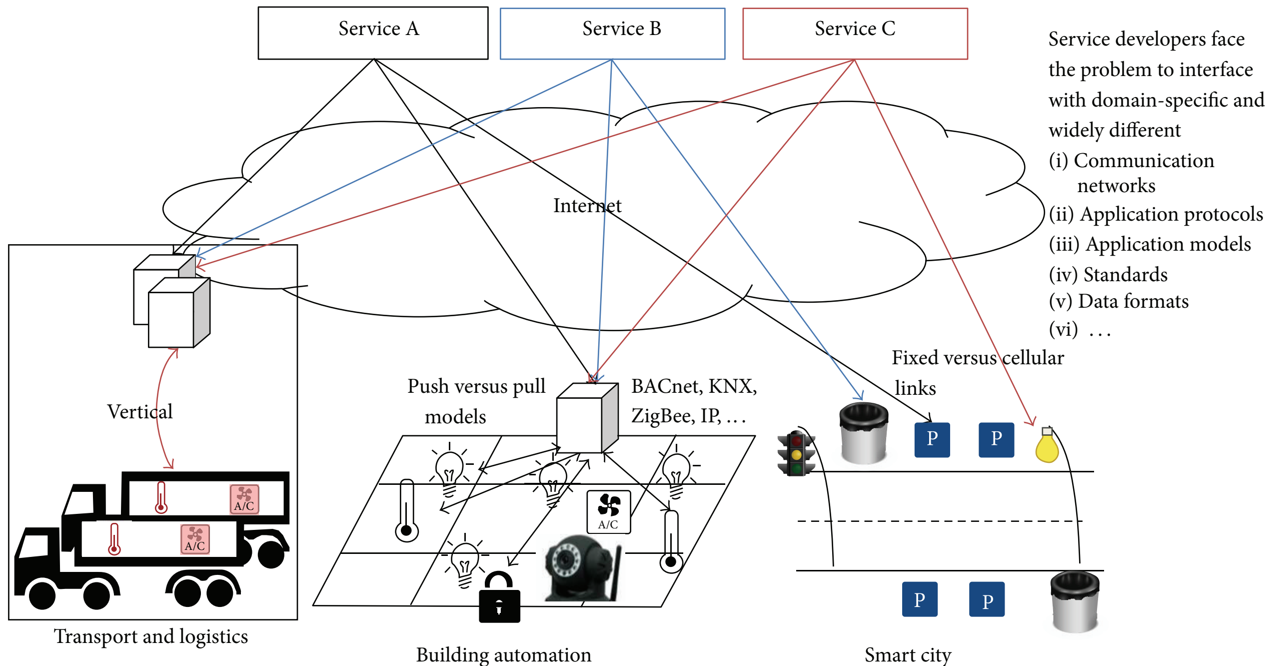


Figure 2: Problem statement: as each domain of the IoT comes with its own set of IoT devices, protocols, standards, data formats, and connectivity options, service providers are forced to integrate a multitude of different technologies when developing cross-domain services.

In this paper our goal is to tackle the heterogeneity presented here by focusing on a standards-compatible cloud-based software system that can integrate with a multitude of different technologies and protocols with a focus on low-resource IoT devices. Our research aims to answer the following questions:

(i) How can standards-based IoT technology be combined with cloud computing to handle the diversity of underlying low-resource IoT devices?

(ii) How can such a cloud computing approach present a uniform interface to third party developers given the different sources of heterogeneity?

(iii) What types of diversity in protocols and communication models can such a cloud-based approach handle?

(iv) What is the impact of this fusion of cloud and IoT on the communication with low-resource IoT devices? Can this communication be made more efficient?

(v) How can we further exploit the power of cloud computing to support interactions from third parties with IoT devices?

4. Background: Embedded Web Services via CoAP

The diverse environments in which IoT devices have to operate have led to a mix of proprietary and standard-based protocols and different application models that are deployed in today's Internet of Things. While there are a number of standards relevant for the Internet of Things [14], this paper will build on the standardization as per the IETF protocol stack for constrained devices [6, 15] and more specifically on the embedded RESTful approach followed by the Constrained Application Protocol (CoAP) [16, 17]. CoAP was chosen because it is a lightweight but powerful protocol that is an ideal candidate for integrating constrained devices into the cloud.

CoAP is a specialized web transfer protocol for use with constrained devices and networks. In CoAP, every physical object (i.e., thing) hosts multiple resources that represent data gathered from sensors or actions available to actuators. Every resource is accessible via a unique uniform resource identifier (URI) and can be interacted with via the GET, PUT, POST, and DELETE REST methods. CoAP can be considered as a highly optimized version of HTTP/1.1 for use in the low-resource embedded domain. Main differences with HTTP include the use of connectionless UDP, support for multicast-based group communication, built-in discovery support, simplified header parsing, and a publish/subscribe extension [18]. Daniel et al. present a detailed comparison between CoAP, HTTP, and SPDY in [19].

A typical CoAP exchange is shown in Figure 3. The CoAP RFC specifies the well-known/core resource as the entry point for resource discovery. In this example the CoAP server responds that it hosts a temperature and light intensity resource. The server then responds to the client's temperature resource request. CoAP requests and responses can be sent in Confirmable (CON) and Nonconfirmable (NON) CoAP messages. As the name suggests CON messages expect

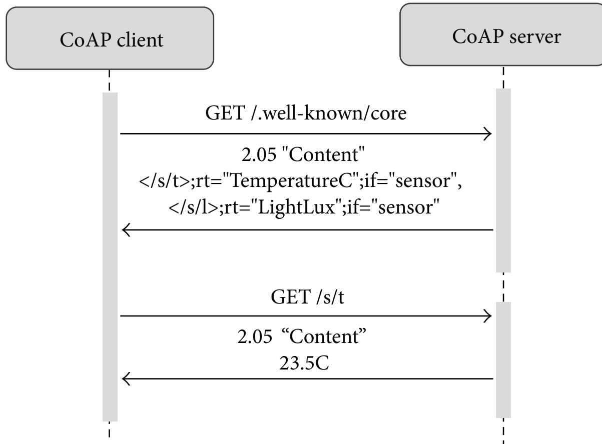


Figure 3: A typical request/response exchange between a CoAP client and server.

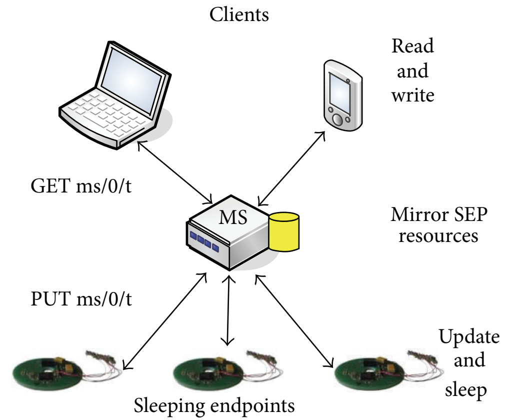


Figure 4: CoAP mirror server: clients and sleeping endpoints can communicate in an asynchronous fashion.

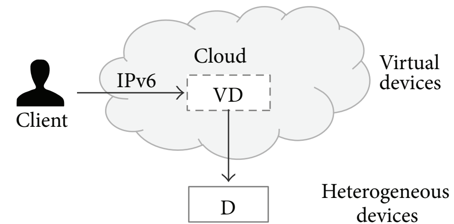


Figure 5: Virtual devices in the cloud represent their real heterogeneous counterparts.

the receiver to acknowledge the reception of the message via an acknowledgement (ACK). Most of the time, the response to the client's request is piggy-backed on top of this ACK.

As mentioned, CoAP provides a publish/subscribe extension in the form of the CoAP Observe mechanism [18]. When a client is observing a resource, the server promises to send new representations of the resource to the client following a best-effort strategy. This frees the client from having to explicitly poll the resource for changes. As observe notifications are regular CoAP responses with the observe option set, they can be sent as CON and NON messages.

In the CoAP ecosystem there are a number of other works relevant to our discussion here. A CoRE Resource Directory [20] facilitates the discovery of CoAP devices and resources in cases where direct discovery is not practical due to sleeping nodes, disperse networks, or networks where multicast is inefficient. To this end, a resource directory hosts descriptions of resources that are available on other CoAP servers. Clients can perform lookups within a resource directory via the web interface specified in the IETF Internet draft.

A second relevant CoAP mechanism is that of a CoRE Mirror Server [21]. Such a server mirrors the resources of a constrained devices, thereby enabling these devices to go into sleep mode and to disconnect their network link in order to save resources. Reference [21] defines the web interfaces for registering resources, sending resource updates, querying for mirrored resources, and retrieving updates of mirrored resources. A mirror server can also be extended to mirror resources on behalf of mobile devices, which frequently change their Internet endpoint due to their mobility. An example of a mirror server is shown in Figure 4.

Sleeping endpoints (bottom of the figure) start by registering their to-be mirrored resources with the mirror server via a POST request (not shown in the figure). From then on, sleeping endpoints (bottom of the figure) operate as CoAP clients. They update their resources on the mirror server via CoAP PUT requests and can query the mirror server for updates to their resources via a CoAP POST request. After a sleeping endpoint has retrieved a list of changed resources (via the POST request), it can choose to process the changes by retrieving every updated resource via a CoAP GET request. The "Client Operation" interface of the mirror server enables clients (top of the figure) to retrieve and change the mirrored resources. Note that clients must know and implement this Client Operation interface.

5. Cloud Platform for Supporting Heterogeneous Devices and Communication Models

A high-level overview of our approach is presented in Figure 5. Low-resource IoT devices with various forms of heterogeneity are shown at the bottom. These employ diverse hardware, protocols, and communication models. For each of these devices, our cloud-based platform hosts a virtual counterpart that is made available as a dedicated IPv6 endpoint (i.e., the virtual device). Clients interact only with the virtual device and the cloud takes care of mapping these interactions to the particular constrained device. In our approach the dedicated IPv6 endpoint hosts both a CoAP and a DTLS server and therefor all interactions between a client and a virtual device run over CoAP.

In effect, our approach follows the Sensor as a Service (SenaaS) paradigm where clients interact with a virtual cloud-based counterpart of a real-life sensor or device. The main benefit of SenaaS is that a virtual device has significantly more

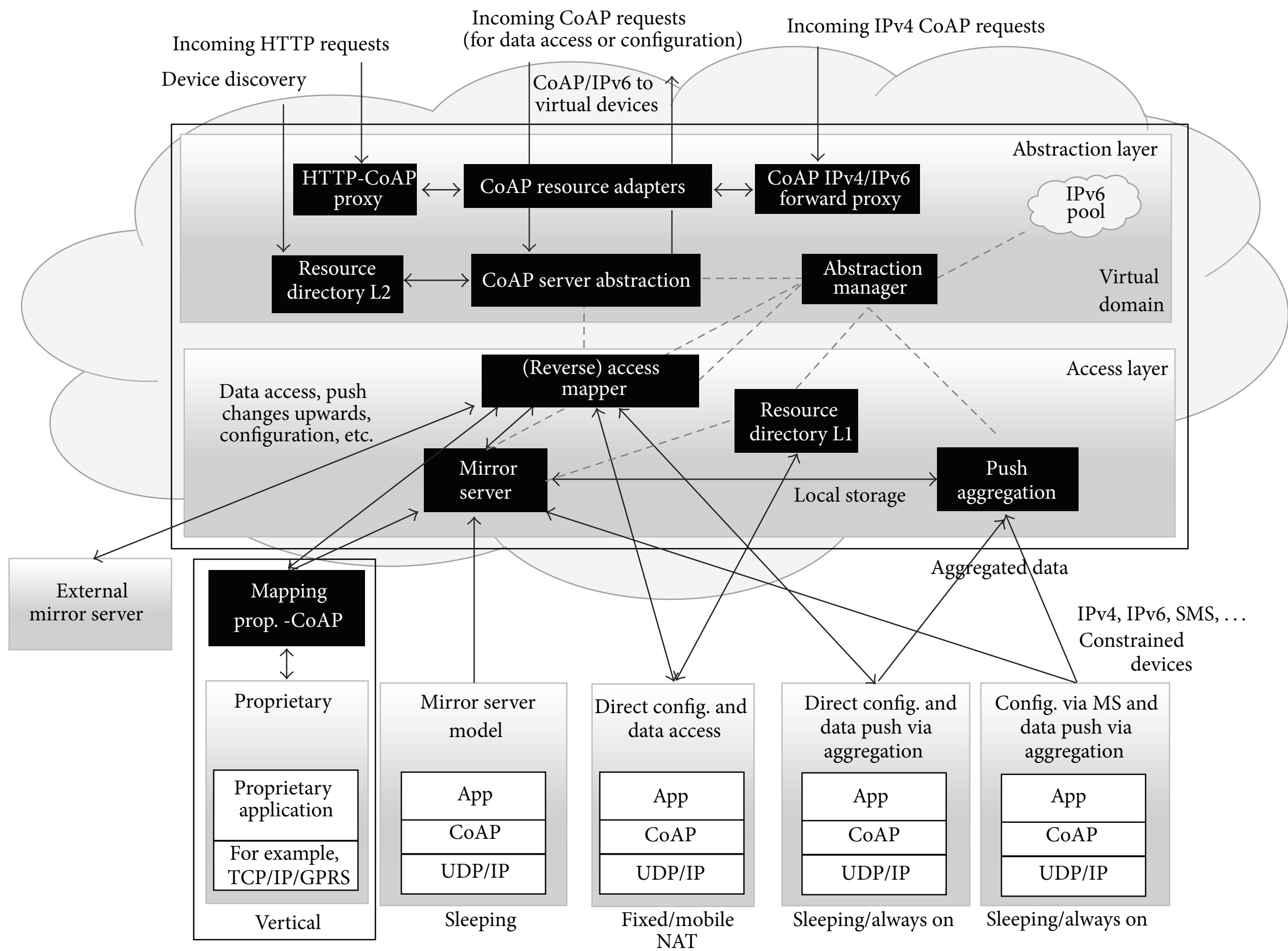


FIGURE 6: The access and abstraction layers of the design enable uniform access to heterogeneous constrained IoT devices.

resources at its disposal than its constrained counterpart and is therefore not hindered by the constraints common to low-resource devices. For example, a virtual device is always available whereas a constrained device might be asleep or temporarily unreachable (e.g., due to mobility). Furthermore, by deploying our platform in the cloud it can support on-demand dimensioning of computing resources when the load on the platform fluctuates. This allows scaling efficiently as the size of the deployment grows and avoids under- and overdimensioning computing capacity. Finally, maintaining the platform's computing infrastructure is outsourced to a specialized external party in this case.

The use of CoAP leads to a lightweight solution where a virtual device can be used by both conventional services as well as by the low-resource devices themselves. The straightforward mapping between CoAP and HTTP has the benefit that virtual devices can easily be integrated into existing RESTful web services. CoAP also provides built-in support for response caching. The result is a resource-oriented architecture that facilitates the integration of constrained devices into third-party services and applications.

To realize this architecture, we have split our cloud-based platform into two layers. The first layer offers a uniform interface to the underlying constrained devices by means of virtual devices. The second layer handles the heterogeneity in constrained devices and applications models. Both of these layers are presented in the following two subsections.

5.1. The Access Layer: Providing Access to Heterogeneous Devices and Communication Models. The access layer is closest to the constrained devices and is responsible for communication to and from these heterogeneous devices. In some cases the access layer also stores the data provided by the constrained devices. As can be seen from Figure 6, access is provided to constrained devices that employ a number of heterogeneous communication models. At the heart of this layer is the access mapper component; this module maps the uniform representation from the abstraction layer to the particular implementation of the constrained device and vice versa. To this end, the access mapper translates requests for CoAP resources hosted by the device abstraction to device-specific actions. In the next few paragraphs we will discuss specific instances of this mapping for the different communication models that are shown at the bottom of Figure 6.

The first communication model is that of the mirror server model that was already introduced in Section 4.

For constrained devices hosted on mirror servers, the access layer translates requests for virtual devices to requests destined to the corresponding mirror server. It does this by taking the URI path of the incoming request and adding it to the mirror server URI handle of the device. For example, a request for coap://[2001:6a8:1d80:600::21]/s/t on the virtual device is translated to coap://ms.example.com/ms/4/st on the mirror server. Requesting the .well-known/core discovery resource will trigger a request to the device's handler resource on the mirror server (i.e., ms/X). For .well-known/core responses, the access mapper will remove all occurrences of the device's mirror server URI-path handle in the response from the mirror server (otherwise clients would see the s/t resource as ms/4/s/t). Note that the mirror server itself can be running alongside the cloud platform but that it can also be hosted externally (e.g., on-site for reduced latency).

In the direct configuration and data access model, the constrained device hosts a plain CoAP server. In this case, the access mapper operates as a standard CoAP reverse proxy [16] where requests for the virtual device are mapped to the CoAP server on the constrained device. The constrained device can be a mobile device, where its IP endpoint changes as the device's location changes. Also, a constrained device might be behind NAT and/or a stateful firewall. In both cases, access to the device is typically restricted to those parties with whom the device maintains active communication. In case of NAT, the transport layer mapping at the NAT box is expected to be volatile as well. To this end the access layer allows updating the mapping of a device via the "/registerendpoint" CoAP resource. This way the IP endpoint of a device is kept up to date and any state in intermediaries is kept alive. One optimization supported by our reverse proxy is combining multiple CoAP Observe relationships into one relationship. If N clients are observing the same resource on the virtual device, then the mapper will maintain only one observe relationship with the constrained device. Consequently the constrained device has to send only one notification instead of N notifications per resource change, thereby reducing its load and energy consumption.

Communication models three and four are hybrid models where data (i.e., originating from the device) and configuration changes (i.e., destined to the device) are handled via different methods. Both models differ from the mirror server model in that they push data for multiple resources in a single request as opposed to pushing data for a single resource per request (as is the case in the mirror server model). The push aggregation module in the access layer is responsible for splitting the incoming aggregated data into multiple requests (i.e., one per resource) to the mirror server. This can lead to considerable energy savings for the constrained device as the total number of requests is reduced. The difference between these two models is the method they use for configuration changes. One model allows hosting a CoAP server that enables direct configuration changes, whereas the other model relies on a mirror server for configuration.

For the proprietary communication model there are a number of different mapping strategies available, the exact choice of which will be highly dependent on the proprietary technology that is being mapped. There are however two straightforward ways to provide a mapping to constrained devices that are operating inside a vertical. In the first method the constrained devices in a vertical are represented as CoAP clients, while in the second they are represented as CoAP servers. In both cases the vertical defines its own set of resources that provide a suitable RESTful interface for interacting with the proprietary constrained device. In case of the CoAP client option, the vertical can register its devices on a mirror server and all data and configuration changes are handled via the mirror server. The evaluation section presents an example of this approach for a container tracking vertical. In the CoAP server case, the vertical can sometimes act as a cross-protocol proxy.

The final component is the access layer resource directory where mirror servers and direct access CoAP servers register themselves so that they can be discovered by the abstraction manager. The abstraction layer uses this information to instantiate the CoAP server abstractions (i.e., one per constrained device) and the corresponding access mapper instances. The next subsection looks at how this is realized in the abstraction layer.

5.2. The Abstraction Layer: A Homogeneous Restful Interface for Constrained Devices. The abstraction layer is responsible for providing a homogeneous interface to devices that implement the different communication models mentioned in the previous paragraphs. Furthermore, it also allows extending this device abstraction with new functionality. Finally, the layer provides proxy services to IPv4 and HTTP for broadening the interfacing possibilities with the device abstraction. The layer is part of the virtual domain, where also the device abstractions reside.

In terms of the device abstraction, we chose to represent every constrained device as a virtual device that implements a CoAP server with one or more resources. This virtual device is hosted as a dedicated IPv6 endpoint by allocating an IPv6 address from an IPv6 subnet that is routed to the cloud. While this abstraction is hosted at the network layer, our abstraction layer will only process CoAP (and DTLS) traffic for virtual devices; other types of traffic are not processed. One benefit of using a network-layer abstraction is that multiple IPv6 subnets and IPv6 routing can be used for distributing device abstractions over a number of different cloud systems in order to improve scalability. The result is that every constrained device is made available as an open standards-compliant (virtual) CoAP server regardless of the underlying communication model and protocols of the device. To this end, the L2 resource directory contains web links to all virtual devices and their resources.

On top of the server abstraction is a block that enables extending the functionality provided by the server abstraction on both the transport layer and the application layer (i.e., on the level of CoAP resources). The "CoAP resource adapters" module allows a managing party to instantiate chains of plugin-like functional blocks that extend the server abstraction in a desired way. This concept has been presented in previous work [22] and has been shown to significantly reduce the communication overhead of DTLS in IP-based wireless sensor networks [23]. In this work we reuse the

concept of adapter chains for realizing the security model of our platform as explained in the next paragraph. It is also used to implement a cache for CoAP responses, which is running on top of memcached.

In terms of the security model, the cloud platform is considered as a trusted entity by constrained devices. Policies are defined in the cloud platform to delegate (parts of) this trust to clients. Clients of a virtual device are authenticated by the credentials they provide during the DTLS handshake. Currently the platform supports preshared keys (useful for constrained clients), raw public keys, and standard X.509 certificates issued by a party trusted by the policy (see later). The server abstraction is authenticated to the client via the same types of credentials. The resource adapters block provides a DTLS adapter type for authenticating clients and handling the DTLS protocol, as per [23]. It also provides an adapter type that handles authorization. This is accomplished by allowing administrators to define policies based on the credentials of the user, the destination of the request, and the desired operation (i.e., CoAP method and targeted CoAP resource). The authorization adapter processes the output of the DTLS adapter (i.e., plain text CoAP requests) and drops all requests that do not adhere to the defined policy.

The final two components enable access to the IPv6 CoAP server abstraction from HTTP clients and from IPv4-only CoAP clients. As mentioned CoAP is designed to interface easily with HTTP, so mapping HTTP and CoAP messages is a straightforward process for the HTTP/CoAP proxy. Furthermore, our device abstraction enables us to host a HTTP server using the IPv6 address of the virtual device. Therefor, the URI-mapping mechanism is very simple (i.e., change the scheme from http to coap) and the URI-mapping issues raised by [24] are not applicable in this case. Finally, our proof of concept deployment learned that not all clients have IPv6 Internet access. More specifically, GPRS-based Internet access is often restricted to the IPv4 Internet. To address this issue, the platform provides an IPv4/IPv6 forward CoAP proxy where the target (IPv6) CoAP URI is encoded in a "Proxy-URI" CoAP option [16].

5.3. Machine-to-Machine Communications. An important application for IoT platforms is machine-to-machine communications. One common issue in M2M is device management, that is, tracking and configuring devices that are deployed in the field. In 2014, the Open Mobile Alliance (OMA) has defined a set of standards for device management in M2M: Lightweight M2M (http://openmobilealliance.org/about-oma/work-program/m2m-enablers/). LWM2M has adopted CoAP and its RESTful mechanisms as the protocol of choice for interfacing with devices. They also mandate the use of DTLS for security and a resource directory for discovery of devices. While the platform presented here does not implement the standardized OMA interfaces (i.e., CoAP resources) for bootstrapping, management, and services, adding them would be straightforward as both our platform and LWM2M share the same building blocks.

One typical characteristic of M2M systems is the use of SMS services for communications, as opposed to IP-based communications. The CoAP protocol was designed with a range of low-power and lossy network types in mind [25] and an adaptation of CoAP to SMS transport is presented in [26]. Thus, the same application protocol can be used for interfacing with both SMS and IP-based constrained devices. Furthermore, the design of our platform can easily be extended with SMS functionality in both the abstraction and the access layers. Virtual devices could be allocated a unique mobile number, which would make them accessible to machines that are restricted to SMS communications. Finally, adding an SMS gateway in the access layer would allow virtual devices to map to SMS-only constrained devices.

6. Evaluation

We have evaluated our platform via three methods, two of which provide a quantitative evaluation about specific aspects of the platform while the third is a qualitative evaluation in the form of a proof of concept.

The setup for the quantitative evaluations is shown in Figure 7. The client is a Raspberry Pi model B connected natively to the IPv6 Internet via a commercial Belgian ISP. The cloud platform is running on a virtual machine hosted at our University's Data Center, where Internet peering is provided by Belnet. Belnet is a Belgian Internet provider for educational institutions, research centers, scientific institutes, and government services. The wireless sensor network consists of three Zolertia Z1 nodes, which are equipped with TI's 16 bit msp430 microcontrollers (8 KB RAM and 128 KB ROM) and a CC2420 802.15.4 radio. According to IETF terminology, these devices fall under the "Class 1" category [27]. These are devices that are constrained in code space and processing capabilities, in that they cannot easily communicate with other Internet nodes employing a full protocol stack such as using HTTP, TLS, and related security protocols and XML-based data representations. However, they have enough power to use a protocol stack specifically designed for constrained nodes (such as CoAP over UDP) and participate in meaningful interactions without the help of a gateway node.

All Zolertia nodes are running the IETF stack for constrained devices available in contiki. The two sensor nodes on the right of the figure are battery-powered and employ the ContikiMAC MAC and Radio Duty Cycling protocol. The channel check frequency of ContikiMAC was lowered to 4 Hz to conserve energy. Routing inside the 6LoWPAN network is enabled by the RPL routing protocol. One Zolertia node is configured with the direct access model (pull) and one with the mirror server model (push). The third sensor node acts as a 6LoWPAN and RPL border router and is connected to a Raspberry Pi model 2. This Raspberry Pi is connected to the IPv6 Internet via a SixXS.net tunnel (on the Belgian Easynet PoP). The 2001:6f8:202:85cc::/64 subnet is routed over this tunnel and distributed inside the 6LoWPAN network.

6.1. Virtual Device Abstraction: Scalability and Latency. While routing all traffic to the cloud-based virtual device has

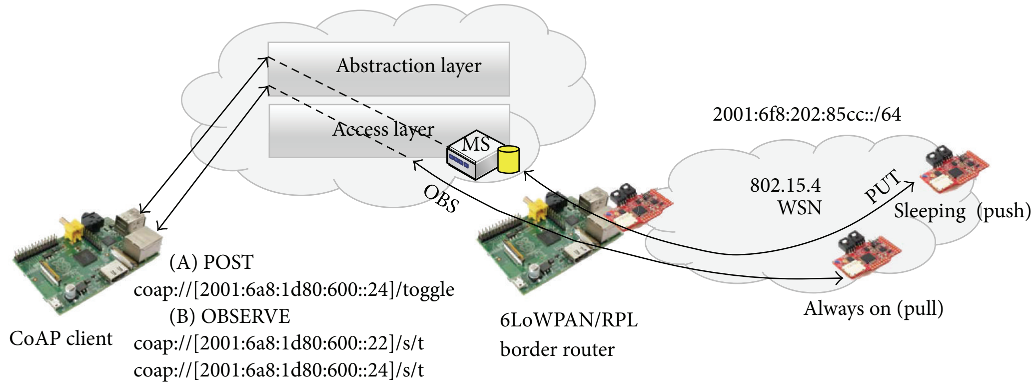


Figure 7: Two Raspberry Pi's and an 802.15.4 wireless sensor network operating 6LoWPAN from the evaluation setup for our cloud platform.

many benefits (the most important being availability of computing resources), it may also introduce some undesirable downsides. One obvious issue is that of scalability. As the number of IoT devices continues to increase in the coming years, the volume and velocity of IoT traffic are expected to grow exponentially [28]. However, this issue can be addressed by relying on the flexibility provided by our network layer abstraction in terms of allocating IPv6 addresses to virtual devices and in terms of configuring routing tables to distribute traffic to the machines running our cloud platform. Via dynamic routing tables we can move virtual devices from machines that experience a high load to newly allocated computing resources. The reverse mechanism allows scaling down when the load decreases, thus realizing the resource elasticity common to cloud computing. However, this is still future work.

Another potential problem is latency; that is, how does routing traffic via the virtual device abstraction impact latency between a client and a constrained device? In a lot of cases our approach can actually provide latency improvements by, for example, serving content from a cache. However, not all requests can be satisfied from the cache; activating an actuator is a prime example of this. What is the impact on latency in this case? In order to quantify this impact, we have conducted response time measurements using the setup from Figure 7. Confirmable CoAP POST requests that toggle a LED on the Zolertia sensor node were sent for measuring the round trip time (RTT) from client to constrained device. Only the always-on node (with RDC) was used in this experiment. Over 8000 CoAP requests were sent sequentially for the following two configurations:

(1) Cloud: the client sends the POST request to the virtual device in the cloud, where it is translated to a POST request to the corresponding constrained device. The response follows the reverse path.

(2) End-to-end: the client sends the POST request to the constrained device directly (not shown in Figure 7).

A CDF of the response times is shown in Figure 8. We can see that the differences between the two configurations are small and negligible for most use-cases. This is because

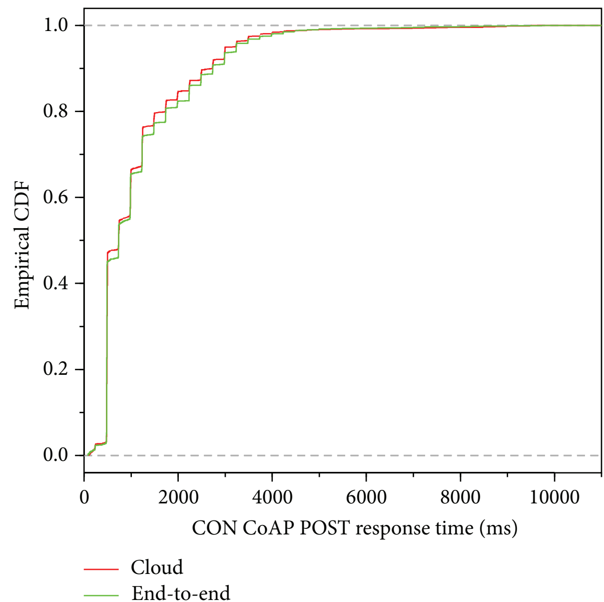


Figure 8: Cumulative distribution function of response times for confirmable CoAP POST requests.

most of the response time is spent on sending the request over the last wireless hop from the border router to the sensor node (due to the RDC). For the end-to-end configuration, the average RTT between the client and the 6LoWPAN router was only 29.8 ms (with a standard deviation σ of 5.6 ms). For the cloud configuration, the average RTT between client and cloud was 22.5 ms ($\sigma = 5.1$ ms) and between cloud and 6LoWPAN router was 17.3 ms ($\sigma = 1.5$ ms). For the end-to-end configuration the minimum CoAP POST RTT measured was 74.7 ms; for the cloud configuration it was 89.7 ms, that is, a difference of 15.0 ms. However, considering the fact that for the cloud configuration more than 90% of the measured response times were longer than 482.0 ms (and 99.9% were longer than 100 ms), we can see that the impact of the lossy 802.15.4 network in combination with radio duty cycling on latency is much higher than hosting our virtual device in the cloud.

6.2. Communication Models: Push versus Pull. In this subsection the mirror server and direct access communication models, introduced in the previous section, are compared in terms of energy consumption. For these experiments the setup from Figure 7 is also used. There are two configurations for each of the models: one where the data communication period is 10 seconds and one where this period is 30 seconds. In all experiments the Raspberry Pi CoAP client observes a temperature resource on the virtual CoAP server. The abstraction of the sleeping device is hosted at 2001:6a8:1d80:600::22, whereas 2001:6a8:1d80:600::24 corresponds to the direct access device. When the sleeping device is not engaged in an active CoAP exchange it switches its radio into sleep mode until its next transmission; otherwise it employs RDC. The always-on device continuously employs RDC.

When the CoAP client observes the temperature resource on the virtual device, the cloud platform will observe the corresponding resource on either the mirror server or the constrained device itself. The sleeping device is configured to update the mirror server every 10 or 30 seconds via CoAP PUT requests (push model). Every twentieth PUT request is a confirmable CoAP message, whereas the other requests are nonconfirmable messages with the CoAP No-Response option. Responses are suppressed because the client would not be awake to receive them; see https://tools.ietf.org/html/draft-tcs-coap-no-response-option-10 for further information. The always-on device is configured to update its resource every 10 or 30 seconds and to send one confirmable notification for every twenty notifications (pull model). The other nineteen notifications are sent as nonconfirmable messages. Note that this is the default behaviour for CoAP Observe in Contiki's Erbium. The resulting four configurations are named PUSH10, PUSH30, PULL10, and PULL30, respectively. Every time the client receives a notification it estimates the energy consumption of the constrained device for transmitting the data that triggered the notification from Energest data that is retrieved from the constrained device [29]. Energest values are read from a serial connection with the constrained node that is running over a TCP connection to a serial forwarder that is attached to the UART0 line of the sensor node. More than 400 measurements were collected per configuration.

The box plots of total energy usage in Figure 9 show that in general the push model consumes less energy than the pull model for this experimental setup. This is due to the fact that for the push model the radio can be kept in sleep mode for longer periods of time per transmission. The stacked bar plot confirms this for the "median energy usage" case, where it is clear that the energy spent in the reception category for the pull models is significantly higher than in the push models. Note that the PUSH30 data points are significantly higher than the PUSH10 data points. If we compare the median cases, then we can see that while the median radio energy expenditures are similar (the same amount of data is being sent in both cases) the CPU and IRQ categories are not. This is because the microcontroller has to process more timer interrupts per data transmission as the time between transmissions is three times longer. Finally, also note that the two distributions of measurements for the 30-second configurations are clearly skewed towards higher energy expenditures (i.e., larger errors bars with larger variation at higher values). This difference is caused by the fact that the total number of sent RPL messages for refreshing upwards and downwards routes increases as the data transmission interval increases. Thus for larger intervals the fraction of energy expenditure data points that include RPL message transmissions will be higher and therefor the box plots will be skewed towards higher values.

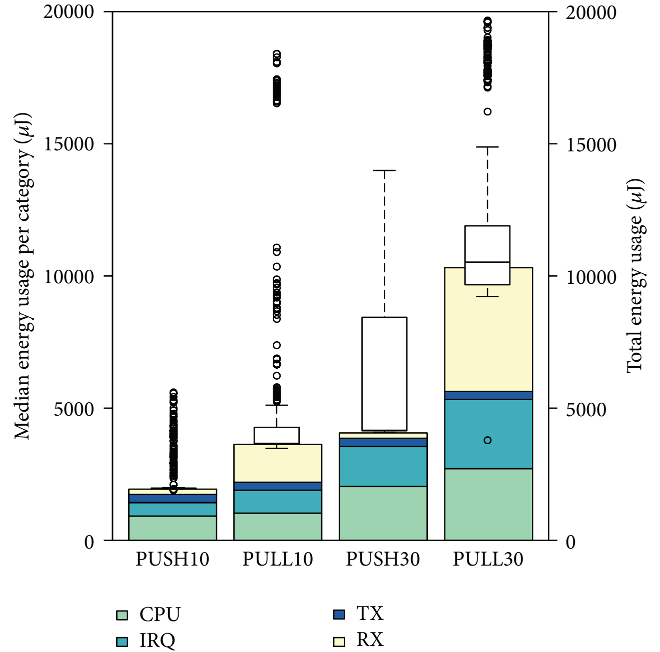


FIGURE 9: Left: stacked bar plot of median energy usage per category. Right: box plot of total energy usage.

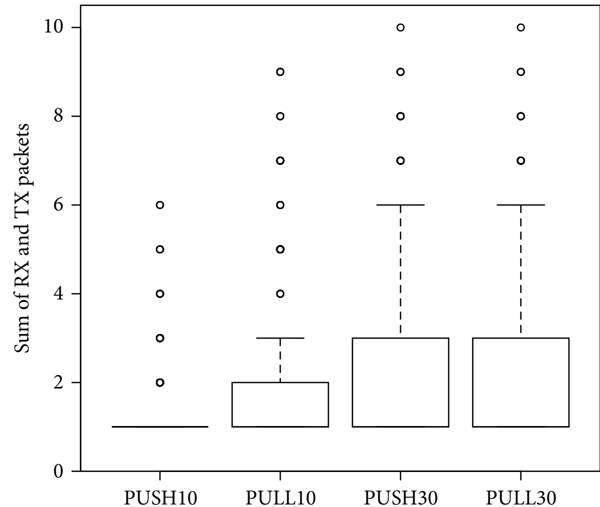


FIGURE 10: Sum of packets received and transmitted for different communication models.

Figure 10 shows the sum of packets received and transmitted by the constrained node per data transmission period.

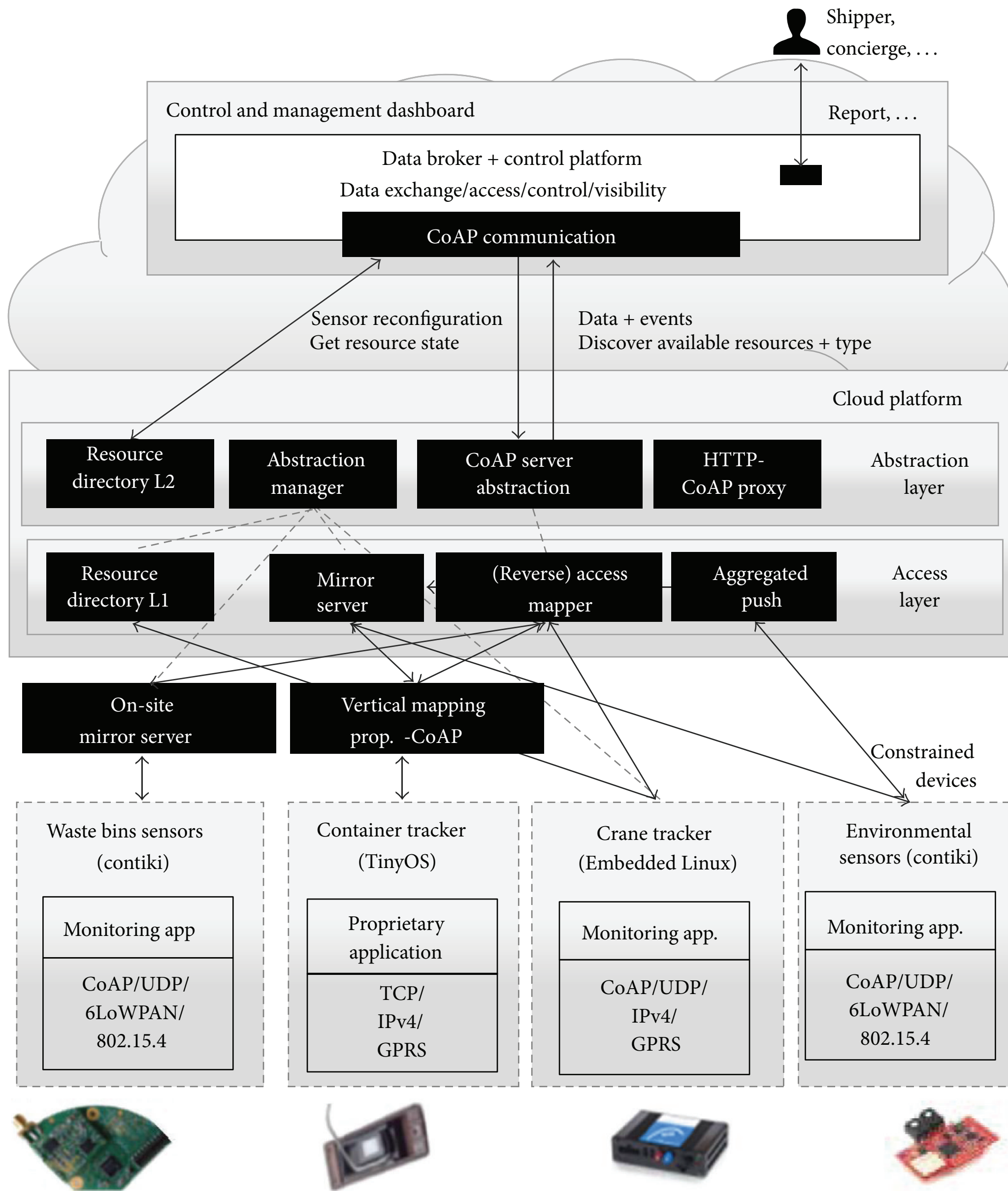


Figure 11: Proof of concept: components and setup.

The median for all configurations is one, which is explained by the fact that nineteen out of twenty data messages are NON messages with no response. Here we can also see that the RPL messages skew the box plots for the PUSH30 and PULL30 scenarios towards the top. Note that the effect of retransmissions of CON messages is hard to see in this plot as they only make up five percent of the data points.

6.3. Proof of Concept: Real World Deployment. In the context of a national project (https://www.iminds.be/en/projects/2014/03/03/comacod), a proof of concept was developed that demonstrates the feasibility of the platform presented in this paper. The project involved three industrial parties that develop monitoring solutions for waste bins, construction cranes, and freight containers. Each of these partners has developed their own vertical system, where customers access monitoring data via a web portal that is hosted inside the vertical. The goal of the proof of concept was to show that our platform enables third parties to collect data and configure constrained devices spanning multiple application domains while maintaining the same levels of performance (i.e., code size, energy expenditure, and life time) and service of the constrained devices as offered by the existing (proprietary) solutions. Figure 11 shows the different components in the proof of concept. At the bottom there are four different types of constrained devices, each fulfilling a monitoring task for a specific use-case.

The waste bin trackers consist of a 32-bit ARM Cortex-M3 microchip (Silicon Labs EFM32) and an Analog Devices ADF7021-N narrow-band transceiver that operates at

169 MHz. These trackers run on contiki, implement the IETF stack for constrained devices, and form a wireless sensor network via RPL in nonstoring mode. The trackers are sleeping devices and employ the mirror server model to send periodical status information (waste bin tray events and battery status) to a mirror server that is running on the 6LoWPAN router (i.e., the on-site mirror server in Figure 11). The 6LoWPAN router is connected to the private network of the company via a VPN tunnel over the public Internet. By also connecting to this private network from the access layer and offering a virtual device for every waste bin tracker registered on the on-site mirror server, the cloud platform has enabled applications to integrate the waste bins as normal CoAP servers. This is a great improvement over the database with web API integration option which was the only choice offered by the company in the past.

The container tracker is composed of a msp430f5437 microcontroller, a CC2520 802.15.4 transceiver, and a Telit HE910 HSPA+ modem with built-in GPS. Containers form an 802.15.4 network that is used for intercontainer communication towards a data sink that transmits the data to a back-end system using its HSPA modem. The containers run TinyOS and use a proprietary data format over TCP for communicating with the back-end. In this case, it was not considered viable to change the proprietary stack running on the devices as the development effort was deemed too high. Instead, the container tracker back-end system implements a mapping from the proprietary technology to embedded web services by means of a CoAP mirror server. For every tracker, the back-end system registered an endpoint on the mirror server with the corresponding resources for location, container mode, temperature, and humidity. Whenever the back-end receives data from the container trackers, the resources of the corresponding tracker on the mirror server are updated. Whenever the container trackers wake up, the back-end checks the mirror server for changes (e.g., the container's mode resource has changed), retrieves all updates for the mirrored resources, and translates these updates to corresponding actions in the proprietary technology. Because of the virtual device abstraction, all trackers are hosted as CoAP servers.

The crane tracker is a powerful embedded Linux device that is running an ARM9 CPU with 128 MB of ROM and 64 MiB RAM. The trackers incorporate a GPS and GPRS chip and used to establish a VPN tunnel with the back-end. In this private network, a tracker remained reachable at a fixed IP address even when its public IP address changed on the cellular network and when it was behind NAT. Data was exchanged using a proprietary format over TCP. There was no real integration possible as the company only offered a web portal for its customers for following up on their cranes. For this use-case, the proprietary technology was dropped and the tracker implements a mobile CoAP server instead. Using our cloud platform, the VPN connection was no longer necessary as the virtual device in the cloud can offer a fixed IPv6 endpoint for the tracker. Whenever the tracker's GPRS IPv4 address changes, it updates the mapping in the access layer via a CoAP POST request to the cloud system. In order to traverse any firewall and NAT systems between the tracker and the cloud, the cloud's access layer uses a UDP source address that is equal to the destination UDP address that was used by the tracker for updating its mapping. As in the previous case, there was no viable integration strategy for third parties (only a web portal was available). By using our cloud platform, trackers are offered as virtual CoAP servers and made discoverable via the resource directory. Figure 12 shows the tracker updating its access layer mapping as well as the message exchange triggered by a CoAP request to the /gps/full resource of the corresponding virtual device. The two packets marked in black show that our system uses the destination UDP address of the registration to contact the tracker afterwards.

The final class of devices consists of Zolertia Z1 nodes that were used as the main testing platform and were deployed as per Figure 7. These devices periodically collect temperature and humidity data using the aggregated push and mirror server communication models.

To illustrate that our cloud platform facilitates integration of constrained devices in the Internet, the project also implemented a control and management dashboard for the different devices from the previous paragraphs. This dashboard is hosted on server infrastructure owned by one of the project partners. HTTP is used for communication from the dashboard to the different types of constrained devices via the corresponding virtual devices and the HTTP/CoAP proxy. The dashboard discovers the available devices and their resources via the resource directory hosted in the abstraction layer of the platform. Figure 14 shows the interface offered to a user to inspect the list of available devices and resources. Resource attributes (such as type and being readable and writable) are determined from the CoAP resource type that is available at discovery. For every resource, the user can choose to view collected data, retrieve a fresh representation, or update the resource. Figure 13 shows the collected data for the /s/w resource of the waste bin tracker hosted at the coap://[2001:6a8:1d80:600::4] virtual device. Data includes switch events (i.e., waste bin lid events) and battery status.

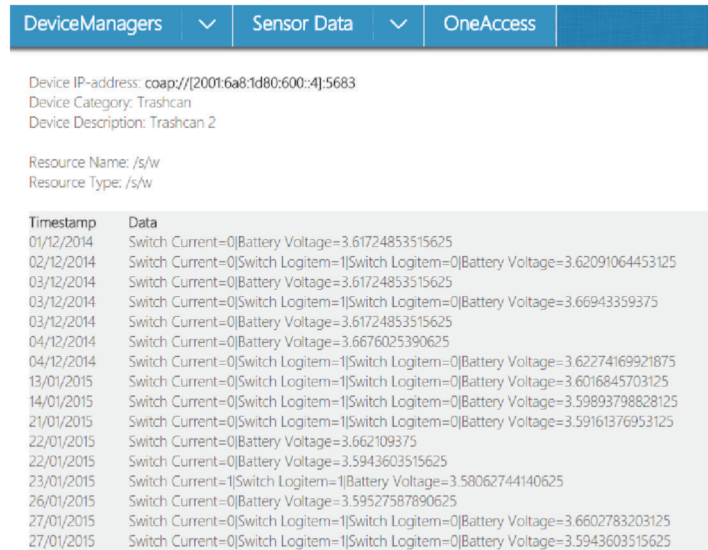
Finally, the case study from Section 2 where the crane tracker updates the location of the container tracker when it picks up a container was realized. One problem here was the IPv4-only Internet access provided by the crane's GPRS modem. Here the CoAP IPv4/IPv6 forward proxy was used for updating the location resource on the virtual device of the container tracker. Another problem was that of local discovery; that is, how does the crane tracker discover the virtual device corresponding to the container it picked up? Here a number of solutions were formulated: using RFID communication for communicating the virtual device's address or searching the platform's L2 resource directory using the container's endpoint identifier (communicated via RFID or optically via a barcode). However, at the end a solution for this local discovery problem was not implemented in this proof of concept.

7. Related Work

While cloud computing and the Internet of Things come from different backgrounds, recent developments show that

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000	178.144.143.106	193.191.148.156	CoAP	145	CON, MID:45400, POST, coap://dtm.comacod.test.iminds.be/dtm/registerendpoint
2	0.0028	193.191.148.156	178.144.143.106	CoAP	106	ACK, MID:45400, 2.05 Content
3	27.275	2001:6a8:1d80:1128:d187:f813:134a:c6a	2001:6a8:1d80:600::15	CoAP	77	CON, MID:8657, GET, End of Block #0, /gps/full
4	27.627	193.191.148.156	178.144.143.106	CoAP	58	CON, MID:27686, GET, TKN:ad, End of Block #0, /gps/full
5	28.120	178.144.143.106	193.191.148.156	CoAP	81	ACK, MID:27686, 2.05 Content, TKN:ad (text/plain)
6	28.122	2001:6a8:1d80:600::15	2001:6a8:1d80:1128:d187:f813:134a:c6a	CoAP	102	ACK, MID:8657, 2.05 Content, End of Block #0 (text/plain)

Figure 12: (1) + (2): crane tracker updates access layer mapping. (3)–(6): CoAP request to device abstraction (3) triggers corresponding access layer request (4).



Timestamp	Data
01/12/2014	Switch Current=0|Battery Voltage=3.61724853515625
02/12/2014	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.62091064453125
03/12/2014	Switch Current=0|Battery Voltage=3.61724853515625
03/12/2014	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.66943359375
03/12/2014	Switch Current=0|Battery Voltage=3.61724853515625
04/12/2014	Switch Current=0|Battery Voltage=3.6676025390625
04/12/2014	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.62274169921875
13/01/2015	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.6016845703125
14/01/2015	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.59893798828125
21/01/2015	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.59161376953125
22/01/2015	Switch Current=0|Battery Voltage=3.662109375
22/01/2015	Switch Current=0|Battery Voltage=3.5943603515625
23/01/2015	Switch Current=1|Switch Logitem=1|Battery Voltage=3.58062744140625
26/01/2015	Switch Current=0|Battery Voltage=3.59527587890625
27/01/2015	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.6602783203125
27/01/2015	Switch Current=0|Switch Logitem=1|Switch Logitem=0|Battery Voltage=3.5943603515625

Figure 13: The dashboard presents a table with collected data of a waste bin tracker to the user.

they are increasingly being adopted together. The cloud and IoT can complement one another in a number of interesting ways, as will become clear from the works presented in the following paragraphs.

The work of Guinard et al. was one of the first to propose a resource-oriented architecture for the Internet of Things [30, 31]. By implementing RESTful resources on things, the interfaces of things have become similar to those found on the World Wide Web and therefor this is referred to as the "Web of Things." When these resources are integrated into larger services, the result becomes a service-oriented architecture which is a well-known concept that is often realized on top of cloud computing infrastructure.

Historically, one of the first research domains where cloud computing and IoT have been combined was wireless sensors networks. In [32] a Sensor-Cloud infrastructure is introduced, where physical sensors are virtualized as virtual sensors on cloud computing. Motivation for doing so include the limited resources and capabilities of physical sensors and the ease of use and management of virtual sensors. A comprehensive work on WSN and cloud is the survey [32] by Yuriyama and Kushida. The authors argue that the combination of Cloud and WSN enables remote access and allows annotating sensors with XML in the cloud. The use of XML encourages the interchangeability of different types of sensors and allows describing services offered by sensors (e.g., via the Web Service Description Language). Here also the use of web services is presented to enable different applications to talk to one another. The resulting WSN is coined as a service-oriented sensor network.

In recent years, a sizeable amount of research has been performed on the combination of cloud computing and the IoT. This has resulted in numerous papers as well as cloud-based IoT platforms. In [9] Botta et al. introduce the CloudIoT paradigm which is the integration of cloud and IoT. The paper is a good introduction to the topic as it lists the motivation, applications, related work, research challenges, open issues, and platforms for the CloudIoT paradigm. The listed motivations for CloudIoT include the abundance of resources (e.g., storage, computational, and network resources) of the cloud as well as significant improvements in scalability, interoperability, and security. One identified research challenge is the integration of huge amounts of highly heterogeneous things into the cloud, which overlaps with our research objectives from Section 3.

Another work focuses on moving application logic from the firmware to the cloud. In [33] Kovatsch et al. introduce the Thin Server Model, where devices in the role of a server do not host any application logic. Instead, devices expose their elementary functionality via RESTful services and any application logic of devices runs on application servers (possibly in the cloud). The resulting IoT infrastructure is said to be agnostic of any applications. Our work adopts some of these ideas (e.g., thin devices and extradevice applications); however we also address the issue that not all devices can be modeled using the Thin Server Model. Some (legacy)

Figure 14: Control and management dashboard: the user is presented with a list of devices and resources. The user can access data collected from resources, retrieve a new resource representation, or update a resource.

devices implement proprietary technology or implement a client-based communication model (e.g., sleeping devices). The work presented here also looks at how these devices can be integrated into the Internet and be made available to such "application servers."

Works by Alam et al. [34] and Zaslavsky et al. [28] propose the Sensing and Sensor as a Service ideas. SenaaS exposes functional aspects of sensors as services by hiding technical details of the sensor from the user. By specifying and delivering sensor functionalities and capabilities as services, one can exploit all the existing service standards for interacting with sensors. Similar to our work, [34] defines an "IoT Virtualization Framework" where IoT devices are virtualized and offer virtual services. However, in our work the virtualization takes place at the device level (i.e., with every device having a virtual counterpart that is hosted at a specific IPv6 address) whereas the virtualization in [34] takes place on the service layer. Furthermore, the technology used in both approaches is different: our work focuses on the IETF stack for constrained devices whereas the work of Alam et al. focuses on the standards developed by the Open Geospatial Consortium (i.e., Sensor Web Enablement or SWE). One important difference is that our approach enables virtual devices to be used by constrained devices. This is typically not the case for the more verbose SWE standards.

In [10], Li et al. present a cloud-based Platform as a service solution (PaaS) for delivering IoT services. Like our work, the authors make the observation that today IoT services are typically delivered as physically isolated vertical solutions where all system components are customized and tightly coupled for each use-case. The authors propose moving to "virtual verticals" that are built on top of a common cloud infrastructure according to the presented IoT PaaS architecture. By building on top of the functionality already provided by the platform, IoT solution providers can deliver their services more efficiently and can continuously extend their product. Our work is similar to the IoT PaaS concept, but it is focused on solving the specific problem of integrating heterogeneous constrained devices into the IoT. Furthermore we argue that the IoT should move away from all verticals (both virtual and physical) in order to realize applications that span multiple domains and IoT product manufacturers.

In Cloud4Sens [35] two strategies for managing sensing resources in the cloud and providing them as a services are discussed and a cloud framework that handles these two strategies is presented. In the data-centric model, the cloud offers data to its clients as a service without knowing how the data is measured and processed. In the second model, the device is at center and clients can access data via devices and customize one or more virtual devices. To this end the architecture includes an SWE abstraction layer that enables a number of different thing deployments to interface with the cloud. The framework also includes a Software as a Service component (SaaS) for offering data to interested clients and an Infrastructure as a Service (IaaS) component that enables clients to interact with virtual devices. Our solution can be seen as an alternative to the IaaS component that is geared towards supporting heterogeneous devices and that is designed to allow constrained devices to interact with other virtual devices. Through the resource adapters, our approach also allows enhancing virtual devices with additional functionality.

Next to the works presented above, there are also many IoT platforms—both commercial and academic—available today. The gap-analysis presented in [13] considers over 30 of such platforms. One identified shortcoming is the integration of sensor technologies without the use of gateways. Problems include the lack of standards and heterogeneous interaction models. Both are addressed by our work by relying on CoAP and showing how heterogeneous devices (with different communication models) can be integrated. Other shortcomings include data ownership and data fusion and sharing; while these topics are not addressed by our platform itself they were part of the control and management dashboard in the proof of concept in Section 6.3. In the future we will look at how these can be integrated as part of the platform.

In [36] a system architecture for IoT cloud services based on CoAP named Californium is presented. The architecture consists of three stages (network, protocol, and logic) that form a processing pipeline where each stage has its own

separate thread pool. The result is a highly scalable architecture as proven by a comparison with state of art alternatives from both the CoAP and HTTP server domain. Even though the focus of the work is different than ours (scalability versus integration and heterogeneity), it is mentioned here to demonstrate that CoAP is a suitable protocol for (scalable) IoT cloud services.

CloudThings [37] is a service platform that allows users to run IoT application on cloud hardware. It includes tools for application development and for operational management and deployment. The platform uses 6LoWPAN and CoAP for communication with things and RESTful web services for integration with the cloud. The platform offers web services for data subscription and discovery. However, these are only accessible over HTTP which limits their use for constrained devices that wish to discover things via the platform. Also, the evaluation is very limited and considers only HTTP for communication between the cloud and things.

8. Conclusion

We have shown that our cloud-based platform facilitates the integration of constrained IoT devices into other services without significantly impacting service and device operation. Experiments show that our platform supports a number of different communication models that are abstracted away from services by interfacing with a virtual CoAP server abstraction. Furthermore, the proof of concept demonstrates that the platform supports heterogeneous hardware platforms, communication models, and proprietary protocols. The developed control and management dashboard show that our platform can easily integrate constrained devices into third party services.

Future work will focus on what is currently missing in the platform. By offering Data as a Service (DaaS) services, others can access (historical) data without being forced to capture this data when it is made available via the virtual device abstraction. This will also require a data ownership mechanism, as data will be offered separate from the device abstraction. Another weak point of the platform is device-to-device interaction between local devices. In case of real-time applications, mechanisms to allow direct access between devices (e.g., by redirecting devices to the device itself) might be necessary. However, due to the heterogeneity of the devices involved, we expect this to be a tall order. Finally, improving and quantifying the scalability of our platform (e.g., via dynamic routing tables as mentioned in Section 6.1) will also be considered in the future.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to acknowledge that part of this research was supported by the COMACOD project. The iMinds COMACOD project is cofunded by iMinds (Interdisciplinary Institute for Technology) a research institute founded by the Flemish Government. Partners involved in the project are Multicap, OneAccess, Track4C, Invenso, and Trimble, with project support of IWT.

References

[1] L. Ericsson, "More than 50 billion connected devices—taking connected devices to mass market and profitability," Tech. Rep., 2011, http://vdna.be/publications/Wp-50-Billions.Pdf.

[2] D. Evans, "The internet of things: how the next evolution of the internet is changing everything," Tech. Rep., 2011, http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[3] J. Bradley, J. Barbier, and D. Handler, *Embracing the Internet of Everything to Capture Your Share of 14.4 Trillion USD*, Cisco Ibsg Group, 2013, http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf.

[4] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi, "From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view," *IEEE Wireless Communications*, vol. 17, no. 6, pp. 44–51, 2010.

[5] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "MQTT-S—a publish/subscribe protocol for wireless sensor networks," in *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pp. 791–798, January 2008.

[6] M. R. Palattella, N. Accettura, X. Vilajosana et al., "Standardized protocol stack for the internet of (important) things," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 3, pp. 1389–1406, 2013.

[7] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[8] J. Macaulay, L. Buckalew, and G. Chung, "Internet of things in logistics," Tech. Rep., 2015, http://www.dhl.com/en/about_us/logistics_insights/dhl_trend_research/internet_of_things.html.

[9] A. Botta, W. D. Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and internet of things," in *Proceedings of the 2nd International Conference on Future Internet of Things and Cloud (FiCloud '14)*, 2014, http://wpage.unina.it/walter.dedonato/pubs/iot_ficloud14.pdf.

[10] F. Li, M. Voegler, M. Claessens, and S. Dustdar, "Efficient and scalable IoT service delivery on cloud," in *Proceedings of the IEEE 6th International Conference on Cloud Computing (CLOUD '13)*, pp. 740–747, IEEE, Santa Clara, Calif, USA, July 2013.

[11] L. Atzori, A. Iera, and G. Morabito, "The internet of things: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[12] K. Romer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54–61, 2004.

[13] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of internet-of-things platforms," http://arxiv.org/abs/1502.01181.

[14] S. Husain, A. Prasad, A. Kunz, A. Papageorgiou, and J. Song, "Recent trends in standards related to the internet of things and machine-to-machine communications," *Journal of Information and Communication Convergence Engineering*, vol. 12, no. 4, pp. 228–236, 2014.

[15] I. Ishaq, D. Carels, G. K. Teklemariam et al., "IETF standardization in the field of the Internet of Things (IoT): a survey," *Journal of Sensor and Actuator Networks*, vol. 2, no. 2, pp. 235–287, 2013.

[16] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, *RFC 7252: Constrained Application Protocol (CoAP)*, 2014, https://tools.ietf.org/html/rfc7252.

[17] C. Bormann, A. P. Castellani, and Z. Shelby, "CoAP: an application protocol for billions of tiny internet nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[18] K. Hartke, *Observing Resources in CoAP*, IETF CoRE Working Group, 2014, https://tools.ietf.org/html/draft-hartke-coap-observe.

[19] L. Daniel, M. Kojo, and M. Latvala, "Experimental evaluation of the CoAP, HTTP and SPDY transport services for internet of things," in *Proceedings of the 7th International Conference on Internet and Distributed Computing Systems*, pp. 111–123, 2014.

[20] Z. Shelby and C. Bormann, *Core Resource Directory*, IETF CoRE Working Group, 2014, https://tools.ietf.org/html/draft-ietf-core-resource-directory-02.

[21] M. Vial, *CoRE Mirror Server*, 2013, https://tools.ietf.org/html/draft-vial-core-mirror-server-01.

[22] F. van den Abeele, J. Hoebeke, I. Moerman, and P. Demeester, "Fine-grained management of CoAP interactions with constrained IoT devices," in *Proceedings of the IEEE Network Operations and Management Symposium (NOMS '14)*, pp. 1–5, IEEE, Kraków, Poland, May 2014.

[23] F. Van den Abeele, T. Vandewinckele, J. Hoebeke, I. Moerman, and P. Demeester, "Secure communication in IP-based wireless sensor networks via a trusted gateway," in *Proceedings of the IEEE 10th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP '15)*, Porto, Portugal, 2015.

[24] E. Dijk, A. Rahman, T. Fossati, S. Loreto, and A. Castellani, *Guidelines for HTTP-CoAP Mapping Implementations*, IETF CoRE Working Group, 2014, https://tools.ietf.org/html/draft-ietf-core-http-mapping-06.

[25] B. Savolainen and T. Silverajan, *CoAP Communication with Alternative Transports*, 2015, https://tools.ietf.org/html/draft-silverajan-core-coap-alternative-transports-08.

[26] M. Becker, K. Li, K. Kuladinithi, and T. Poetsch, *Transport of CoAP over SMS*, 2014, https://tools.ietf.org/html/draft-becker-core-coap-sms-gprs-05.

[27] C. Bormann, M. Ersue, and A. Keranen, "RFC 7228: Terminology for Constrained-Node Networks," 2014, http://tools.ietf.org/html/rfc7228.

[28] A. Zaslavsky, C. Perera, and D. Georgakopoulos, "Sensing as a service and big data," in *Proceedings of the International Conference on Advances in Cloud Computing*, 2013.

[29] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based sensor node energy estimation," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, pp. 409–410, ACM, 2007.

[30] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of web services," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223–235, 2010.

[31] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the internet of things to the web of things: resource-oriented architecture and best practices," in *Architecting the Internet of Things*, pp. 97–129, Springer, Berlin, Germany, 2011.

[32] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure—physical sensor management with virtualized sensors on cloud computing," in *Proceedings of the 13th International Conference on Network-Based Information Systems*, pp. 1–8, Takayama, Japan, September 2010.

[33] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: towards the thin server architecture for the internet of things," in *Proceedings of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS '12)*, pp. 751–756, IEEE, Palermo, Italy, July 2012.

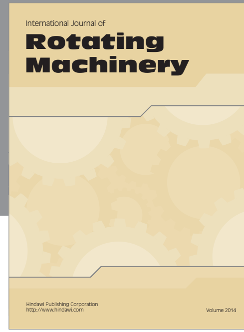
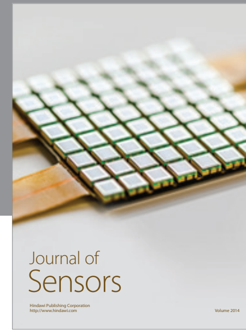
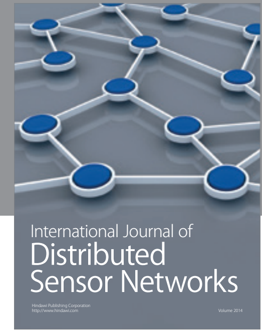
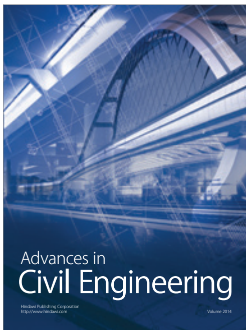
[34] S. Alam, M. M. R. Chowdhury, and J. Noll, "SenaaS: an event-driven sensor virtualization approach for Internet of Things cloud," in *Proceedings of the IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, pp. 1–6, Suzhou, China, Novemeber 2010.

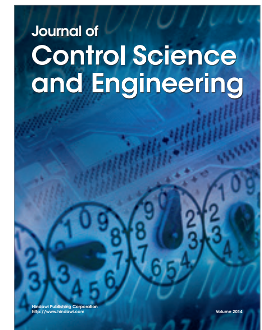
[35] M. Fazio and A. Puliafito, "Cloud4sens: a cloud-based architecture for sensor controlling and monitoring," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 41–47, 2015.

[36] M. Kovatsch, M. Lanter, and Z. Shelby, "Californium: scalable cloud services for the internet of things with coap," in *Proceedings of the International Conference on the Internet of Things (IOT '14)*, pp. 1–6, IEEE, Cambridge, Mass, USA, October 2014.

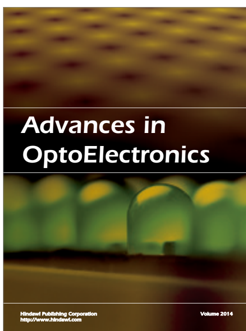
[37] J. Zhou, T. Leppanen, E. Harjula et al., "CloudThings: a common architecture for integrating the Internet of Things with Cloud Computing," in *Proceedings of the IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD '13)*, pp. 651–657, June 2013.

Journal of
Engineering

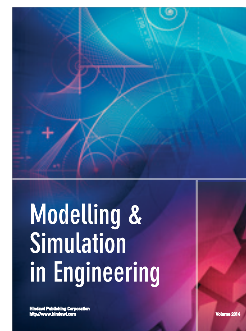
The Scientific
World Journal

International Journal of
Rotating
Machinery

Journal of
Sensors

International Journal of
Distributed
Sensor Networks

Advances in
Civil Engineering

Hindawi
Submit your manuscripts at
http://www.hindawi.com

Journal of
Control Science
and Engineering

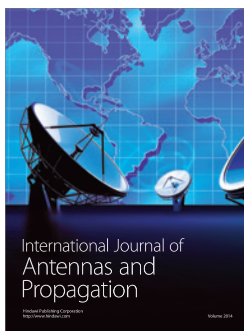
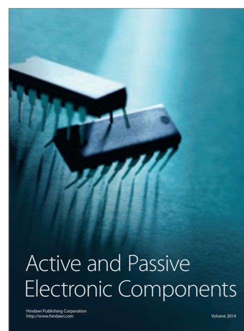
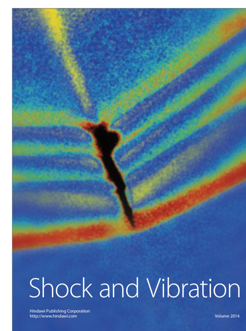
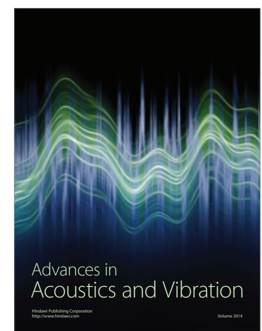
Journal of
Robotics

Journal of
Electrical and Computer
Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and
Observation

Modelling &
Simulation
in Engineering

International Journal of
Aerospace
Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and
Propagation

Active and Passive
Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration