LEONORE – Large-Scale Provisioning of Resource-Constrained IoT Deployments

Michael Vögler, Johannes M. Schleicher, Christian Inzinger, Stefan Nastic, Sanjin Sehic and Schahram Dustdar
Distributed Systems Group, Vienna University of Technology, 1040 Vienna, Austria
{lastname}@dsg.tuwien.ac.at

***Abstract*—Internet of Things (IoT) devices are usually considered as external dependencies that only provide data, or process and execute simple instructions. Recently, IoT devices with embedded execution environments emerged that allow practitioners to deploy and execute custom application logic on the device. This approach fundamentally changes the overall process of designing, developing, deploying, and managing IoT systems. However, these devices exhibit significant differences in available execution environments, processing, and storage capabilities. To accommodate this diversity, a structured approach is needed to uniformly and transparently deploy application components onto a large number of heterogeneous devices. This is especially important in the context of current large-scale IoT systems, such as in the smart city domain. In this paper, we present LEONORE, a service oriented infrastructure that provides elastic provisioning of application components on resource-constrained and heterogeneous edge devices in large-scale IoT deployments. LEONORE supports push-based as well as pull-based deployments and we show that our solution is able to elastically provision large numbers of devices using a testbed based on a real-world industry scenario.**

I. Introduction

Current Internet of Things application design and development approaches model IoT solutions as layered architectures [1] with a bottom layer consisting of deployed IoT devices and their communication facilities, a middleware layer to expose the underlying hardware in a unified manner, and a top-level application layer to execute business logic and visualize processed sensor data [2]. This layered architecture implies that business logic is only executed in the application layer, and IoT devices are assumed to be deployed with appropriate software and readily available [3]. However, in practice this is not the case. Apart from the most basic sensors, IoT devices provide constrained execution environments with limited processing, storage, and memory resources to execute device firmware, which can further be used to offload parts of application business logic onto these devices. In the context of our work, we refer to these devices as *IoT gateways*.

In large-scale IoT systems, such as in the smart city domain, leveraging the processing capabilities of gateways is especially important, as their currently untapped processing capabilities can be used to significantly improve IoT applications by moving parts of the application logic towards the edge of the infrastructure. Making edge devices first-class execution environments in the design of IoT applications enables new types of solutions that are able to dynamically adapt to inevitable changes such as new requirements or adjustments in regulations, by modifying their component deployment topology and edge processing logic. System integrators can avoid infrastructure silos and vendor lock-in by implementing custom business logic to be executed on gateways, and even purchase and sell these application components in an IoT application market [4]. However, the heterogeneous nature of current IoT gateways poses challenges for application delivery, due to significant differences in device capabilities in terms of available storage and processing resources, as well as deployed and deployable software components. Furthermore, the large number of devices in typical IoT systems calls for a scalable and elastic provisioning solution that is specifically tailored to their resource-constrained nature.

In this paper, we present LEONORE, a service oriented infrastructure and toolset for provisioning application components on edge devices in large-scale IoT deployments. To accommodate the resource constraints of IoT gateways, installable application packages are fully prepared on the provisioning server and specifically catered to the device platform to be provisioned. Our solution allows for both, push- and pull-based provisioning of devices. Pull-based provisioning, a common approach in contemporary configuration management systems, allows devices to independently schedule provisioning runs to off-peak times, whereas push-based provisioning allows for greater control over the deployed application landscape by immediately initiating critical software updates or security fixes. We illustrate the feasibility of our solution using a testbed based on a real-world IoT deployment from one of our industry partners and show that LEONORE is able to elastically provision large numbers of IoT gateways in reasonable time.

The remainder of this paper is structured as follows: In Section II we present the motivation and outline the specific problem as well as requirements. In Section III we introduce the LEONORE infrastructure and tool set to address the identified problems in deploying large-scale IoT systems. We provide a detailed evaluation in Section IV, discuss relevant related research in Section V, followed by a conclusion and an outlook on future research in Section VI.

II. Motivation

Large-scale IoT systems in the smart city domain face the challenges of connecting and managing millions of heterogeneous IoT devices. Due to the fast paced development of these devices and changing requirements in the smart city domain itself, this is not only a matter of handling large-scale deployments, but also about supporting the necessary evolution to manage this change. One area where this is specifically demanding is large-scale Building Operations and Management (BMO). Providers of such BMOs not only need to be able to stage large amounts of new devices, due to the

978-1-4799-8356-8/15 $31.00 © 2015 IEEE
DOI 10.1109/SOSE.2015.23

rapid urbanization, they also need to reflect the ever changing requirements to their existing infrastructure. Current solutions to address this situation are mostly manual or only deal with parts of a BMO providers infrastructure. These solutions are not only unable to deal with the vast amount of devices and changing requirements in an efficient manner, they are also tedious and due to the high personal effort, very cost intensive.

A. Scenario

In general, Building Management and Operations that work with large-scale IoT systems need to deal with two distinct states. The initial deployment and staging of devices on the one hand, and updates with varying frequency and priorities on the other hand. Consider the case of a BMO provider residing in a large city operating and managing several hundreds of buildings with a broad variety of tenants. These buildings are equipped with a plethora of different IoT enabled devices ranging from sensors to detect smoke and heat, to elevator and door controls, to complex cooling and heating systems. They rely on gateways [5], [6], [7], which provide constrained execution environments with limited processing, storage, and memory resources to execute the device firmware and simple routines. Gateways enable the basic bundling and management of a wide variety of connected entities. Due to the current market situation and the existing lack of standards in this novel field, there exists a huge heterogeneity in terms of software environments when it comes to these gateways. Initially all these devices need to be staged with the necessary capabilities to enable their basic functionality. The connected sensors need to be supported, the latest firmware needs to be installed and they need to be integrated into a specific deployment structure. This is followed by long term evolution in terms of general maintenance, changing deployments, shifting capabilities as well as updating the software environment or firmware. The second sort of updates is caused by security updates and hot fixes that need to be deployed very fast in order to ensure that the whole infrastructure stays operational. These updates are time critical since delays can cause severe security problems in the whole infrastructure. The more devices are connected the more devices are vulnerable in terms of hacks and exploits and in the IoT domain, where these devices are connected to the real world, this poses a specific threat.

We, therefore, outline the following distinct requirements in the context of this scenario:

- Gateways participating in an IoT infrastructure are resource-constrained in terms of their processing, memory, and storage capabilities.
- Our scenario deals with large-scale deployments comprising thousands of gateways with a wide variety of different supported execution environments.
- Requirements of these gateways change over time, which makes updates necessary. These updates can either be non-time-critical or time-critical, like security updates.
- In order to sustain operations all updates need to be efficient and fast, and, therefore, have to be performed at runtime.

III. Approach

In order to address the previously defined requirements, we present LEONORE, an infrastructure to provision application components on gateways in large-scale IoT deployments. The overall architecture of our approach is depicted in Figure 1 and consists of the following components: (i) Application Packages, (ii) IoT gateways, and (iii) LEONORE, the provisioning framework. In the following, we discuss these components in more detail.

A. Application Packages

Usually an application in the IoT domain consists of different application components and supporting files (e.g., libraries and binaries). To enable automatic provisioning of these so called artifacts, LEONORE builds gateway-specific Application Packages that have the following structure. First, each package has an `id`, which uniquely identifies the package. Second, each package contains a `binary` folder, to store required artifacts. Furthermore, it also contains the resolved application dependencies to avoid expensive dependency resolution on the gateway. Finally, in the `control` folder all instructions for installing, uninstalling, starting and stopping this package are included. Additionally, a `path` file defines the installation paths and the order of installing/uninstalling artifacts. With this packaging approach the heavy lifting is done by the framework, and gateways only have to unpack the package and execute the provided installation instructions, which usually just copy artifacts in place without any additional processing.

B. IoT Gateway

To efficiently provision edge devices in our approach, we first need a general and generic representation of such devices. Therefore, we analyzed the capabilities of several gateways that are commonly applied by our industry partner in the domain of Building Management Systems. Our findings show that in general such gateways have limited hardware components and use some rudimentary, tailored operating system (e.g., a BusyBox[1] user land on a stripped down Linux distribution). Installing or updating software components is a tedious manual task, since there are no supporting packaging or updating tools in place, as known from operating systems used on PCs (e.g., apt[2] for Unix-like computer systems). Furthermore, due to limited resources in terms of disk space, adding new capabilities usually requires the removal of already installed components. Taking all these limitations into account, we derived the final representation of a gateway for our approach as depicted on the right-hand side in Figure 1. The IoT gateway has the following components: (i) a *container*, hosting application packages, (ii) a *profiler*, monitoring the current status of the gateway, (iii) an *agent*, communicating with the provisioning framework, and (iv) a *connectivity layer*, supporting different communication protocols and provisioning strategies.

1) Profiler: As already mentioned, gateways in our approach are usually resource-constrained, which means that they only provide limited disk space, memory and processing power. Therefore, keeping track of these resources is of utmost importance. In order to do that the profiler uses pre-defined interfaces to constantly monitor the underlying system (e.g.,

[1] http://www.busybox.net

[2] https://packages.qa.debian.org/a/apt.html

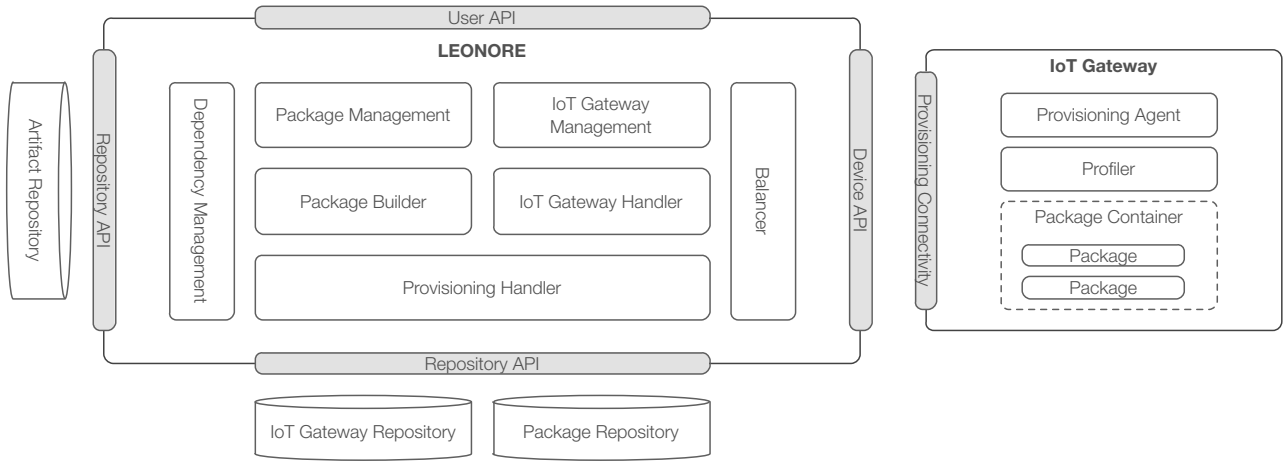


Fig. 1: LEONORE Infrastructure – Overview

static information like id, mac-address, instruction set, or dynamic information like disk- and memory-consumption). The profiler sends the collected information either periodically or on request to the provisioning framework.

2) Application Package and Container: All packages that are not pre-installed on the IoT gateway have to be provisioned by the framework at runtime. Therefore, the IoT gateway uses a runtime container to store and run application packages. By using a separate container it is guaranteed that installing or removing packages does not interfere with the underlying system and, therefore, avoids expensive rebooting or configuration procedures.

3) Provisioning Agent: An essential part for the overall provisioning framework is the provisioning agent. The pre-installed agent is running on each IoT gateway and manages application packages that are locally hosted and stored. The management tasks of the agent comprise installing, uninstalling, starting, and stopping packages. Furthermore, the agent is responsible for handling requests from the framework and triggers the respective actions on the IoT gateways (e.g., gather latest information via the profiler or trigger the provisioning of an application package).

4) Connectivity Layer: Since gateways usually use different software communication protocols in large real world deployments (e.g., oBIX[3] or CoAP[4]), our approach provides a pluggable connectivity layer. This layer can either reuse the deployed software communication protocols or extend services provided by the underlying operating system. Additionally, the layer provides extensible strategies to provision the gateway. In the current implementation, the layer provides two strategies: (i) a *pull-based* approach where the provisioning agent queries the framework for provisioning tasks, and (ii) a *push-based* approach where the framework pushes new updates to the gateway and the agent triggers the local provisioning.

[3]http://www.obix.org

[4]http://coap.technology

C. LEONORE – Provisioning Framework

The enabling framework to provision edge devices in large-scale deployments is depicted on the left-hand side in Figure 1. LEONORE is a cloud-based framework and the overall design follows the micro service architecture[5] approach. This approach enables building scalable, flexible, and evolvable applications. Especially the flexible management and scaling of components is important for LEONORE when dealing with large-scale deployments. In the following, we will introduce the main components of LEONORE and discuss the balancer-based scaling approach.

1) Repositories:

- **Artifact repository** Usually, an application consists of multiple artifacts that get linked together for specific requirements. To handle these artifacts and also make them reusable, a repository is used. The repository manages artifacts by storing source code, pre-built binaries, dependencies, possible configurations, and further necessary information that is required for the application building process. Furthermore, the repository provides a mechanism to store different versions of an artifact.
- **IoT gateway repository** This repository stores the gateway specific information that is needed for creating the deployable application package. This Information includes: hardware configuration (e.g., disk space, memory, processor, etc.), software (kernel version, installed components/tools, etc.), as well as supported provisioning strategies and communication protocols. Additionally, for each IoT gateway the repository stores the provisioned application packages, which is important in case a different version of an installed package needs to be provisioned, since this might require the removal of an already installed version.
- **Package repository** Application packages specifically built for a set of IoT gateways are stored in the package repository. This approach guarantees that packages are

[5]http://martinfowler.com/articles/microservices.html

only built once, and all gateways get provisioned with the same package. Furthermore, by storing the packages in a repository it is easier to scale the framework, since no data is stored in memory and therefore components can be easily replicated. After IoT gateways are successfully provisioned, the package is removed after some time to avoid storing unnecessary data.

2) Package Management: To provision artifacts with LEONORE, users have to add these artifacts via the package management component. The package management is responsible for retrieving all necessary information (e.g., name and version), required binaries, available source files, configurations, policies, and dependencies on other artifacts, from the user. After the user has provided this information along with the artifacts, the package management stores them in the respective repository. The structure of the repository follows the layout of conventional software package management systems (e.g., Maven[6]).

3) Dependency Management: Since many applications depend on libraries or other applications, LEONORE needs a mechanism to resolve these application dependencies. Therefore, according to the desired artifact, the dependency management finds a list of suitable artifacts and provides a plan that can be used to build the actual application package. The plan includes a dependency tree and all needed artifacts. The dependencies are represented as a directional graph, with nodes representing artifacts like applications, libraries, operating system tools, and hardware components, whereas edges represent dependencies between nodes.

4) Package Builder: To create the actual application package that can be provisioned, the package builder is used. In order to build an application package, the builder performs the following steps: (i) retrieve gateway-specific information from the IoT gateway management, (ii) use the dependency management to gather a list of suitable plans, (iii) based on the plan, build an application package, (iv) if the build was successful, notify the provisioning handler to trigger the actual provisioning, (v) if the build failed try next plan in list.

5) IoT Gateway Management and IoT Gateway Handler: In order to deal with the bootstrapping problem, i.e., to know which IoT gateways are available for provisioning, LEONORE follows the following approach. When an IoT gateway starts for the first time, the local provisioning agent registers the gateway with the framework by providing its unique identifier (e.g., name, id, and mac-address) and the gathered profile. Based on this information the manager creates an entry in the IoT gateway repository and stores the provided information. The registration process is finalized by negotiating the supported provisioning strategy and communication protocol. This can be done, since each IoT gateway is pre-configured and provides some already installed communication protocols and provisioning strategies. Next, a suitable IoT gateway handler is assigned for this gateway. IoT gateways that use the same communication protocol and provisioning strategy are grouped together and managed by a designated IoT gateway manager. This assures more flexibility and avoids mediating between protocols. Once the registration process is successful, the IoT gateway can be provisioned by the framework.

[6]http://maven.apache.org

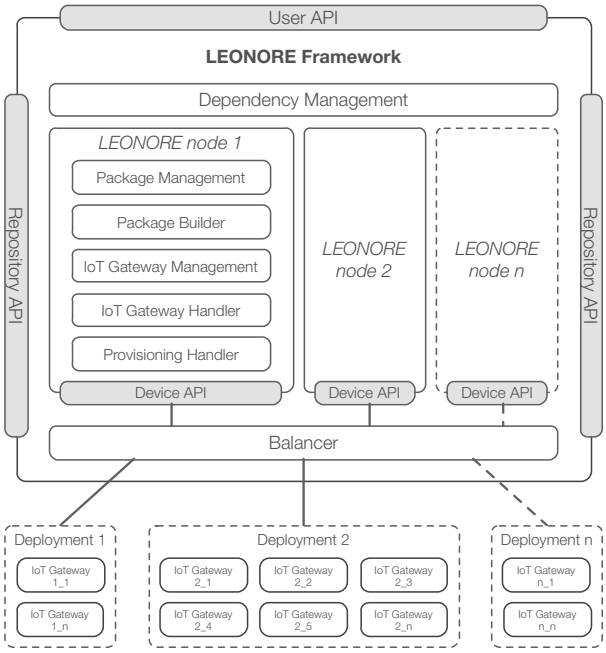


Fig. 2: LEONORE Infrastructure – Balancer

6) Provisioning Handler: To provision application packages, the provisioning handler first chooses the suitable provisioning strategy according to the information provided by the IoT gateway management. Then the handler triggers the building of gateway-specific application packages by invoking the package builder. Once the builder creates the packages, the provisioning handler executes the provisioning strategy. This means that the IoT gateway can either query the framework for application packages or the handler delegates the provisioning request to the respective IoT gateway handler, which pushes the update to the gateway and triggers the provisioning.

7) Balancer: Since LEONORE needs to provision large-scale deployments of IoT gateways, scalability is essential. Therefore, LEONORE provides several strategies to deal with the immense workload. First, the framework's design follows the micro service architecture principle. Thus, optimizing single components is relatively easy by moving them from one host to a more powerful host. Additionally it is possible to scale components by replicating them and therefore distributing the workload across multiple computing resources. Following this approach, components of LEONORE are classified in scalable and not scalable. Components that should be scaleable are grouped together in so-called LEONORE nodes. These nodes comprise all components that are required to handle and provision IoT gateways. The classification in scalable and not scalable is flexible and can be adapted depending on the requirements. Now that LEONORE provides the ability to replicate components via the notion of nodes, we further need a component that is responsible for creating and destroying these nodes, as well as distributing incoming requests to them. To this end, we introduce a balancer. In general, a balancer aims to optimize resource usage, to minimize the response

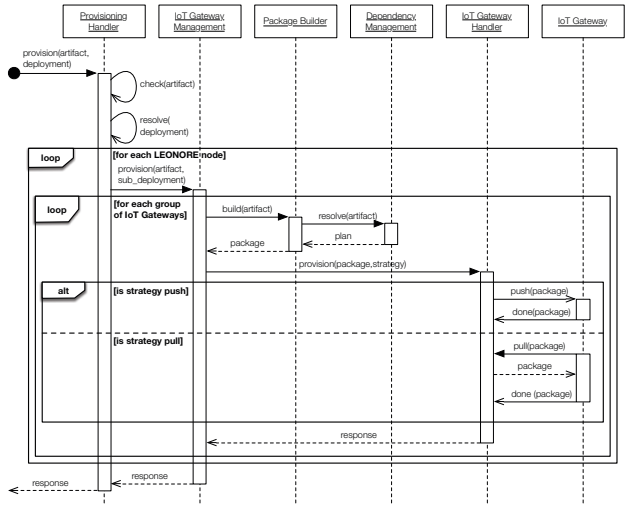


Fig. 3: Provisioning process

time and to maximize the throughput. Figure 2 depicts how LEONORE scales up with growing number of deployments by using the balancer. In Figure 2 we see that the balancer receives incoming requests from IoT gateways deployed in different areas. Based on a pluggable strategy the balancer gathers a suitable node from the pool of available LEONORE nodes and assigns the gateway to this node. The node is then responsible for handling any further interaction with the respective IoT gateway. In case all available nodes are fully loaded, the balancer spins up a new node and queues incoming requests. Once the node is up and running, it registers at the balancer and the balancer assigns the queued up requests to this new node. Currently, LEONORE provides a load-based scaling strategy that scales nodes based on the number of provisioned IoT gateways. In the future, we will provide additional strategies, such as a location-aware strategy that aims at deploying nodes close to affected IoT gateways to reduce network overhead.

D. Provisioning of Application Packages

In order to better illustrate the overall provisioning process, Figure 3 depicts the involved LEONORE components and their interactions for a specific request. Whenever the provisioning of an artifact is requested for a certain deployment, LEONORE performs the following steps: (i) check if the requested artifact is available; (ii) resolve the given deployment to retrieve the set of IoT gateways that have to provisioned; (iii) find the responsible LEONORE nodes, group the gateways according to their node assignment and delegate the provisioning task; (iv) on each node: analyze if the requested artifact is compatible with every IoT gateway and group gateways that require the same application package (e.g. equal hardware or installed packages); (v) on each node: for each group of IoT gateways resolve dependencies and create application package; (vi) on each node: execute required provisioning strategy for each IoT gateway; (vii) on each node: wait until IoT gateways successfully provisioned the package to complete the provisioning task; (viii) check if all nodes have completed their provisioning task to finalize the overall provisioning.

IV. Evaluation

To evaluate our provisioning framework we created a test setup in the cloud by using CoreOS to virtualize devices as Docker containers. IoT gateways in our experiments use two types of provisioning strategies – a pull and a push based approach.

When an IoT gateway uses the pulling approach, the gateway's agent pulls the provisioning framework for new tasks in a configurable interval (e.g. every second). The framework only provides new provisioning tasks for the IoT gateway, which collects and executes these tasks. With short polling intervals, this approach generates increased load on the framework, consumes more bandwidth, and uses more resources on the IoT gateways, but is more fault-tolerant in case of connectivity problems due to inherently frequent retries. For the push-based approach, the IoT gateway's agent only registers the gateway once at the framework and then remains idle until the framework pushes an update. When the agent gets pushed by the framework, it collects the provisioning task, executes it and returns to the idle state. In general, the push-based approach generates less load on both the IoT gateway and framework, but is more vulnerable to connectivity problems and operators need to take care to not inadvertently disrupt gateway operations by placing additional load on it.

To simulate real-world provisioning directives, we use the following two application packages. The first package is the Sedona Virtual Machine[7] (SVM). SVM is written in ANSI C and is highly portable due by design. It allows to execute applications written in the Sedona programming language and is optimized to run on platforms with less than 100KB of memory. For our experiments we developed a small sample application and used SVM Version 1.2.28. The final application package created by LEONORE has approximately 120 KB – including the application code (.sab, .sax, .scode and Kits-file) and the required SVM binary.

As second package we use Java 8 for ARM[8](JVM). In general, using Java on an embedded device is a challenging task, since the JVM binary is quite big and often does not fit due to limited disk space. However, for our experiments we created a compact[9] Java version specifically for our gateway. Additionally, we developed a small sample application that pushes temperature readings to a web server. In total, the JVM application package created by LEONORE has approximately 12MB – including the application code (compiled .class files) and the JVM binary.

In the remainder of this section we give an overview of the used cloud setup, present two scenarios and analyze the gathered results.

A. Setup

To see how LEONORE deals with large-scale deployments, we created an IoT testbed in our private OpenStack[10] cloud.

[7] http://www.sedonadev.org

[8] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-arm-downloads-2187472.html

[9] http://docs.oracle.com/javase/8/embedded/develop-apps-platforms/jrecreate.htm

[10] http://www.openstack.org

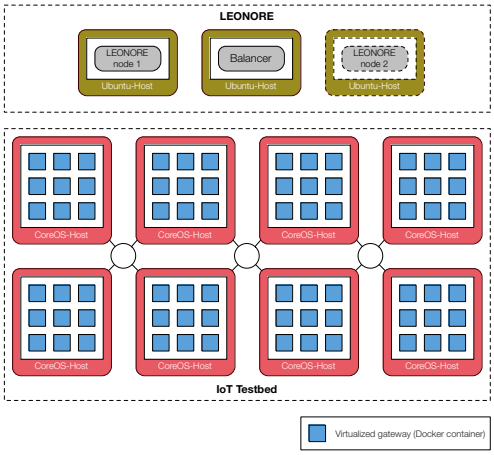


Fig. 4: Evaluation - Setup

In order to simulate large-scale deployments, we first created a snapshot of a real world gateway that is used by our industry partner. Based on this snapshot, we created an image that can be run in Docker[11]. The running image (Docker container), is then used to virtualize and mimic the physical gateway in our cloud.

Since for our evaluation we want to use several thousand virtualized gateways, we employed CoreOS[12] clusters. In general, CoreOS is a light-weight Linux distribution and designed for security, consistency, and reliability. Instead of installing packages via a package management system like apt, CoreOS uses Docker to manage services at a higher level of abstraction. The service code and all dependencies are packaged within a container that can be run on one or many CoreOS machines. Containers provide benefits similar to full-blown virtual machines, but focus on applications instead of entire virtualized hosts. Since containers use the Linux kernel of the host, they have very little performance overhead, which allows for fewer machines to operate and a lower number of compute resources. CoreOS also provides fleet[13], a distributed init system that allows to treat a CoreOS cluster as if it is a single shared init system. We used fleet's notion of service units to dynamically generate according fleet unit files and use fleet for the automated deployment of virtualized gateways.

For our experiments we used the setup depicted in Figure 4: a CoreOS cluster of 8-16 virtual machines (depending on the scenario), where each VM is based on CoreOS 444.4.0 and uses the m1.medium flavor (3750MB RAM, 2 VCPUs and 40GB Disk space). Our gateway-specific framework components are pre-installed in the containers.

The LEONORE framework is initially distributed over 2 VMs using Ubuntu 14.04. The first VM hosts the balancer and uses the m1.medium flavor (3750MB RAM, 2 VCPUs and 40GB Disk space). In order to represent a LEONORE node we created a reusable snapshot of a VM hosting all necessary LEONORE framework components and repositories.

[11]https://www.docker.com
[12]https://coreos.com
[13]https://github.com/coreos/fleet

For the initial deployment of LEONORE one instance of this snapshot is started at the beginning of the experiment. During the experiments LEONORE, more precisely the balancer, spins up another instance of a LEONORE node to distribute the load created by the gateways. The VMs hosting the LEONORE nodes use the m2.medium flavor (5760MB Ram, 3 VCPUs and 40GB Disk space).

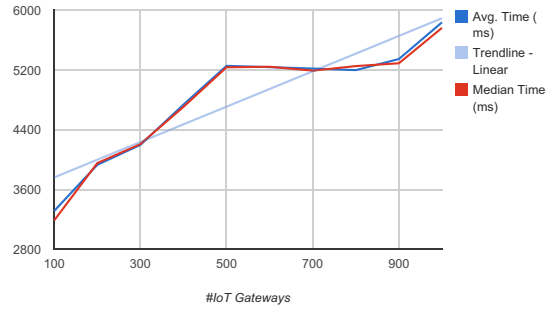
B. Scenario 1: 100 - 1000 IoT Gateways

For the first experiments we picked a scenario with 1000 virtual gateways. The scale of this scenario corresponds to a medium building management system, containing several big buildings (each with more than 10 floors). The 1000 virtual gateways are distributed among a CoreOS cluster consisting of 8 machines, where each machine hosts 125 containers. To demonstrate the scalability of our framework we show how our approach behaves with increasing load (number of gateways).

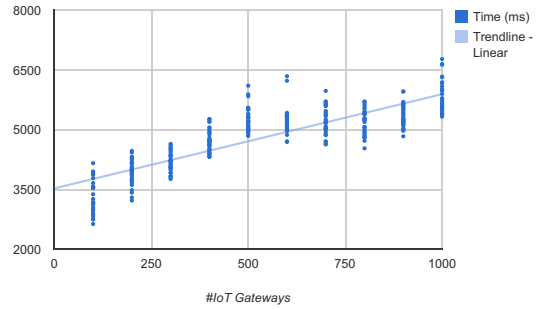
Figure 5 shows the overall execution time of the provisioning process for different deployments by using the pull-based (gateways poll the framework every second) approach. In Figure 5a we show the execution time for provisioning the SVM application package. We see that the execution time increases almost linear until reaching 300 IoT gateways and then has a sharp increase up to 500. When reaching 500, the balancer spins up another LEONORE node and evenly schedules requests to both nodes. Therefore, provisioning time slightly decreases and at approximately 600 becomes constant. When reaching 900 IoT gateways, the provisioning time starts to rise again, which means that at this point both LEONORE node are fully loaded. In order to investigate possible outliers during the evaluation, we created a scatter plot, which is depicted in Figure 5b. Since the SVM application is quite small and the pulling interval of one second has a strong impact on the overall execution time, we executed each experiment 30 times. In Figure 5b we notice that at 600 IoT gateways we have some executions that finished more slowly, which is caused by the high network load and small polling interval. In general, the deviation of provisioning times is small. This shows that provisioning using the pulling strategy is stable and provides reliable results.

Figure 5c shows the execution time when provisioning the JVM application package. We clearly see that due to the increased size of the package the provisioning takes noticeably longer than for the SVM package. When the deployment reaches 500 IoT gateways, the balancer kicks in, which leads to a slight increase. In general, we notice that the provisioning of the JVM package scales linear and produces almost no outliers, as one can see in Figure 5d. Since this application package is quite big and therefore the provisioning time also increases significantly the overhead of the pulling approach is not noticeable.

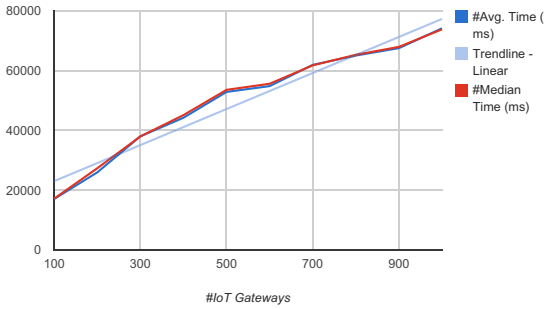
Figure 6 shows the overall execution time of the provisioning process for different deployments using the push-based (framework pushes provisioning tasks to IoT gateways) approach. In Figure 6a we see the overall execution time for provisioning the SVM application package. We notice a sharp increase up to 500 IoT gateways, which is due to the framework pushing requests to all gateways at once and therefore leads to a high load on both the IoT gateways and



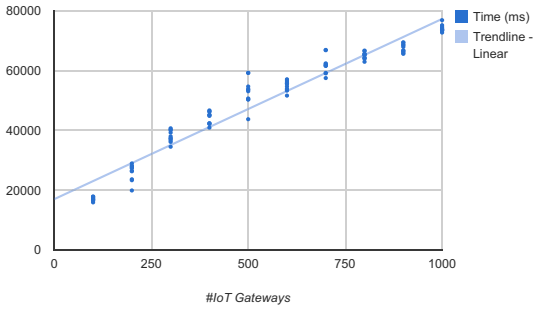
(a) Evaluation Results for SVM



(b) Evaluation Results for SVM - Scatter



(c) Evaluation Results for JVM



(d) Evaluation Results with for JVM - Scatter

Fig. 5: Evaluation Results - Pulling

the framework. Once the balancer spins up another LEONORE node, the execution time is almost constant, because the load is evenly distributed. When the deployment size reaches 900 IoT gateways, the execution time starts to rise again, which indicates that at this scale both nodes are fully loaded. The corresponding scatter plot is depicted in Figure 6b, which reveals that there is only a very small deviation among the data points.

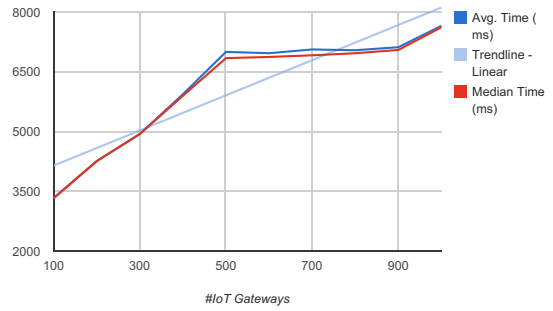
Figure 6c depicts the provisioning time when using the JVM application package. Taking the results of the pulling approach into account, we notice that the initial execution times are identical. However, at 300 IoT gateways we see that the initial overhead of the pushing approach is compensated and therefore the execution time decreases a little bit. From 400 to 500 IoT gateways, the node reaches maximal load. After the deployment size reaches 500, the balancer schedules the load evenly on two LEONORE nodes. The corresponding scatter plot, depicted in Figure 6d, unveils that the deviation of data points is very small and the execution time increases linearly.

After comparing both approaches, we see that our framework scales almost linearly and that for smaller application packages the pull-based approach is faster. For bigger packages both approaches put the framework under heavy load, but produced similar results.

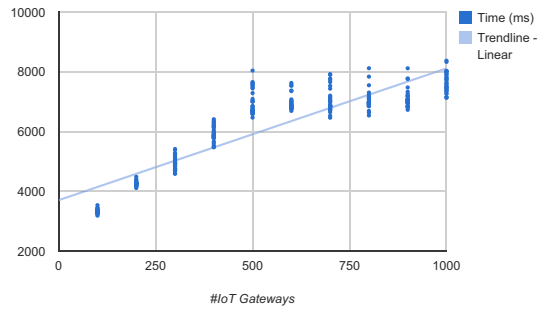
C. Scenario 2: 500 - 4000 IoT Gateways

For the second experiment we used a scenario with 4000 virtual gateways, which corresponds to a large building management system containing dozens of big buildings (each with more than 10 floors). The 4000 virtual gateways are distributed among two CoreOS clusters, each consisting of 8 machines, where each machine hosts 250 containers. With this scenario we want to see how our framework scales when dealing with a large-scale deployment.

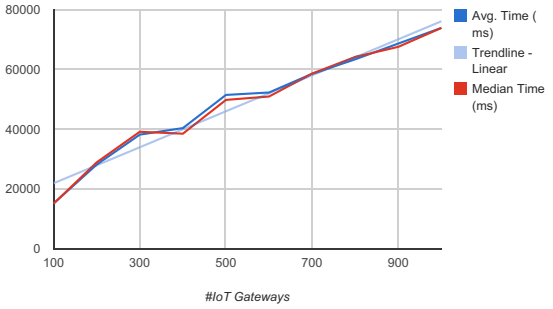
Figure 7 shows the overall execution time of the provisioning process for different deployments by using the push-based approach and the SVM application package. In Figure 7a we notice that due to the deployment scale the overall execution time got slower compared to the first scenario. This is obvious since for this scenario we doubled the amount of CoreOS hosts and deployed twice as many containers on each CoreOS machine. This increase in both the hosts and containers, generates a lot of traffic for the underlying network infrastructure of our cloud, which causes slower response times and therefore the overall provisioning takes longer. Furthermore, for this scenario we changed the balancer to allow 2500 IoT gateways per LEONORE node. We clearly see that up to 2500 IoT gateways, the execution time increases almost linearly. At 2500 the balancer schedules the requests evenly to both nodes,



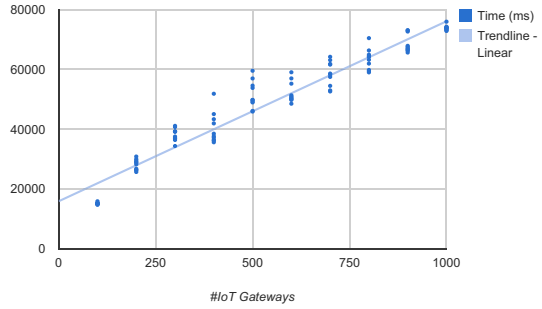
(a) Evaluation Results for SVM



(b) Evaluation Results for SVM - Scatter



(c) Evaluation Results for JVM



(d) Evaluation Results with for JVM - Scatter

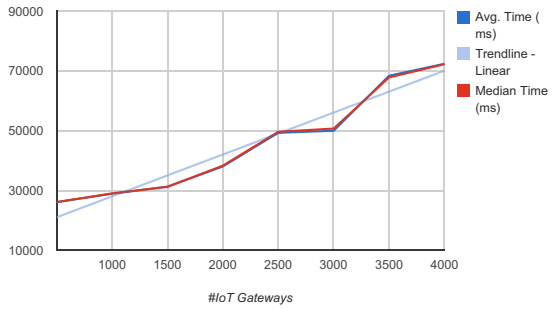
Fig. 6: Evaluation Results - Pushing

which causes a constant execution time. When reaching 3000 deployments, the execution time rises again, but once more starts to flatten at 4000. When looking at the scatter plot depicted in Figure 7b we see that at the beginning of the experiments the deviation among data points is very small and gets bigger with increasing number of IoT gateways. Nevertheless, we clearly see that our framework deals well with this rather large scenario and provides almost linear scale.

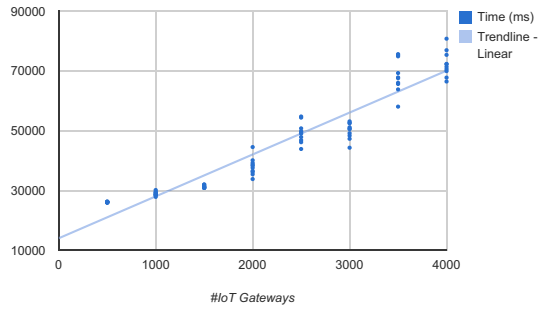
V. Related Work

Recently, applications in the Internet of Things are receiving a lot of attention. More specifically, we notice that the scale of IoT applications can vary from embedded services to enterprise applications. Therefore it is necessary to think of different ways of how to design, develop, deploy and manage such applications not only in the cloud, but also in the underlying IoT infrastructure. Oriwoh et al. [8] presents initial guiding principles (commandments) that stakeholders should follow when developing and deploying IoT enabled devices. Significant work has been conducted to present challenges and key findings when developing a city-scale framework for IoT [9], [10], [11]. Among others an important challenge discussed by the authors is a solution that provides fine-grained provisioning and management of resources. To address some of the aforementioned challenges, [12], [13] define abstract IoT reference architectures and evolutionary approaches to standardize the Internet of Things. Since current IoT solutions often build upon or rely on resource constrained devices, [14] addresses the general problems of managing these resource constrained devices. The authors investigate whether the management of these devices can be accomplished by adopting existing network management protocols.

In addition to general challenges and reference architectures, platforms dealing with the deployment and provisioning of IoT applications emerged. INOX [15] is a robust and adaptable Platform for IoT that provides enhanced application and service deployment capabilities by integrating ideas from Autonomic Network Management. The approach creates a resource overlay to virtualize the underlying IoT infrastructure. The Sensing and Actuating as a Service (SAaaS) paradigm is presented in [16] for pervasively available and geographically distributed IoT infrastructure, to deploy custom functionality. The authors propose a hypervisor as the lowest block of SAaaS to virtualize IoT devices. In contrast to the aforementioned papers, our approach avoids adding another layer of abstraction and allows the direct provisioning of IoT enabled devices, which provides more control and better utilization. The IoT.est project [17] proposes a framework for service creation and



(a) Evaluation Results for SVM with Pushing



(b) Evaluation Results for SVM with Pushing - Scatter

Fig. 7: Evaluation Results - Pushing - Large Deployment

testing in IoT environments. The authors concentrate on the provisioning of services within the boundaries of the framework since underlying resources are abstracted via resource emulators. Adding an abstraction layer on top of the IoT infrastructure gets frequently adopted in the literature (e.g. [18], [19], [20], [21], [22], [23]). All these approaches have in common that the underlying infrastructure stays untouched when provisioning or deploying an IoT solution, in contrast to our approach that also considers IoT devices as first-class execution environments. The Smart-M3 platform is proposed in [24] and aims to create a M3 space by deploying agents on IoT devices. The agents interact based on a space-based communication and synchronization model. Although the authors mention the provisioning of IoT devices, they solely focus on the actual application design. Gemini, a deployment scheme for IoT, is presented in [25] and introduces a system framework for general IoT deployment. Unfortunately, deployment in this context means placement of network elements in an IoT infrastructure, which in our scenario is not feasible as we are dealing with fixed physical deployments that can not be changed easily. Chen et al. [26] introduces over the air provisioning of IoT devices using Self Certified ECDH authentication technology. Although this approach shares the same general idea, the authors explicitly focus on one specific device and do not provide a general and scalable approach. A solution for automatic configuration of IoT infrastructures based on interpretable configuration suggestions is presented in [27]. In contrast to our approach, here the authors use application components that are already pre-installed on the IoT devices and only focus on provisioning application-specific configurations. Li et al. [20] presents an automatic approach to deploy applications on IoT devices by facilitating TOSCA. Since changing applications at runtime is not addressed, our approach can be considered an extension to this work.

Another important area of interest are configuration management (CM) solutions, with the most prominent representatives being Chef[14] and Puppet[15]. While CM addresses a similar overall problem, current tools come with the following limitations that make them unsuitable for the IoT domain. First, they are inherently pull based approaches with clients running on the respective machines, making push based hot fixes (e.g. important security updates) impossible. Second, dependency resolution is usually handed off to a distribution package manager, which is not suitable for the strongly resource-constrained environments we are dealing with.

[14]http://chef.io

[15]http://puppetlabs.com

VI. Conclusion

IoT application development models usually consider IoT devices as external dependencies that only act as data sources or endpoints to send commands to. This leads to a layered architecture with a bottom layer comprising deployed IoT devices and their communication facilities, a middleware layer to expose the underlying hardware in a unified manner, and a top-level application layer to execute business logic. Such a layered architecture implies that business logic is only executed in the application layer, and IoT devices are assumed to be already deployed with appropriate software and readily available. In practice, however, this is not the case. The proliferation of IoT devices with embedded execution environments that are able to execute custom application logic allows practitioners to fundamentally change the way IoT systems are designed, developed, deployed, and managed. However, these devices exhibit significant differences in available execution environments, processing, and storage capabilities. This calls for a structured way to uniformly and transparently deploy application components onto a large number of heterogeneous devices, especially in the context of current large-scale IoT systems, such as in the smart city domain. In this paper, we introduced LEONORE, a service oriented infrastructure for elastically provisioning of application packages on resource-constrained and heterogeneous edge devices in large-scale IoT deployments. Our tool supports push-based as well as pull-based deployments and we showed that our solution is able to elastically provision large numbers of devices using a testbed based on a real-world industry scenario.

In our ongoing work, we plan to further extend LEONORE to address further challenges. For example, we want to develop new balancing strategies to investigate how different

approaches can improve the overall scalability of the framework and reduce the provisioning time. A possible balancing extension would not only consider the size of deployments, but also group devices according to their location. Additionally, we also want to develop advanced provisioning approaches, that can be migrated or moved closer to the deployment infrastructure to reduce for example the overall network traffic or address scenarios where deployments only have limited connectivity. We also see the necessity to improve our IoT gateway representation to better utilize the underlying device-specific capabilities and develop techniques to allow the creation and deployment of more flexible IoT applications. Furthermore, we plan to integrate and align LEONORE with our work on software-defined IoT systems, where it will serve as an integral part of IoT infrastructure management. Moreover, we will explore how existing application development methodologies (e.g., [28]) can and need to be extended in order to efficiently and effectively support application design, deployment, and composition considering large numbers of IoT gateways to perform certain parts of application business logic.

ACKNOWLEDGMENTS

This work is sponsored by Pacific Controls Cloud Computing Lab (PC^3L)[16], a joint lab between Pacific Controls L.L.C., Scheikh Zayed Road, Dubai, United Arab Emirates and the Distributed Systems Group of the Vienna University of Technology.

REFERENCES

[1] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: new wine or just new bottles?" *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1647–1648, Sep. 2010.

[2] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, pp. 1–17, Apr. 2014.

[3] L. Da Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[4] M. Vögler, F. Li, M. Claeßens, J. M. Schleicher, S. Nastic, and S. Sehic, "COLT Collaborative Delivery of lightweight IoT Applications," in *International Conference on IoT as a Service*, 2014, p. to appear.

[5] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "IOT Gateway: BridgingWireless Sensor Networks into Internet of Things," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, 2010, pp. 347–352.

[6] ThereCorporation, "ThereGate." [Online]. Available: http://smarthomepartnering.com/en/platform

[7] Tridium, "JACE Controller." [Online]. Available: http://www.tridium.com/cs/products_/_services/jace

[8] E. Oriwoh, P. Sant, and G. Epiphaniou, "Guidelines for Internet of Things Deployment Approaches – The Thing Commandments," *Procedia Computer Science*, vol. 21, pp. 122–131, 2013.

[9] E. Theodoridis, G. Mylonas, and I. Chatzigiannakis, "Developing an IoT Smart City framework," in *Information, Intelligence, Systems and Applications (IISA), 2013 Fourth International Conference on*, 2013, pp. 1–6.

[10] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, Mar. 2014.

[11] A. R. Biswas and R. Giaffreda, "IoT and cloud convergence: Opportunities and challenges," *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pp. 375–376, 2014.

[12] M. Bauer, M. Boussard, N. Bui, J. De Loof, C. Magerkurth, S. Meissner, A. Nettsträter, J. Stefa, M. Thoma, and J. Walewski, "IoT Reference Architecture," in *Enabling Things to Talk*. Springer Berlin Heidelberg, 2013, pp. 163–211–211.

[13] S. Shen and M. Carugi, "Standardizing the Internet of Things in an evolutionary way," in *ITU Kaleidoscope Academic Conference: Living in a converged world - Impossible without standards?, Proceedings of the 2014*, 2014, pp. 249–254.

[14] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the internet of things," *Communications Magazine, IEEE*, vol. 50, no. 12, pp. 144–149, 2012.

[15] S. Clayman and A. Galis, "INOX: a managed service platform for interconnected smart objects," in *IoTSP '11: Proceedings of the workshop on Internet of Things and Service Platforms*. ACM Request Permissions, Dec. 2011.

[16] S. Distefano, G. Merlino, and A. Puliafito, "Application deployment for IoT: An infrastructure approach," in *Global Communications Conference (GLOBECOM), 2013 IEEE*, 2013, pp. 2798–2803.

[17] S. De, F. Carrez, E. Reetz, R. Tönjes, and W. Wang, "Test-Enabled Architecture for IoT Service Creation and Provisioning," in *Lecture Notes in Computer Science*, A. Galis and A. Gavras, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 233–245–245.

[18] F. Li, M. Vögler, M. Claessens, and S. Dustdar, "Efficient and Scalable IoT Service Delivery on Cloud," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, 2013, pp. 740–747.

[19] S. Murphy, A. Nafaa, and J. Serafinski, "Advanced service delivery to the Connected Car," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, 2013, pp. 147–153.

[20] F. Li, M. Vögler, M. Claessens, and S. Dustdar, "Towards Automated IoT Application Deployment by a Cloud-Based Approach," in *Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on*, 2013, pp. 61–68.

[21] G. Katsaros, E. Wittern, B. Gray, and S. Tai, "A Service Delivery Framework to Support Opportunistic Collaborations," in *Service-Oriented and Cloud Computing*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 4–18.

[22] S. Nastic, S. Sehic, M. Vögler, H. L. Truong, and S. Dustdar, "PatRICIA – A Novel Programming Model for IoT Applications on Cloud Platforms," in *Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on*, 2013, pp. 53–60.

[23] R. Fantacci, T. Pecorella, R. Viti, and C. Carlini, "Short paper: Overcoming IoT fragmentation through standard gateway architecture," *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pp. 181–182, 2014.

[24] D. Korzun, S. Balandin, and A. Gurtov, "Deployment of Smart Spaces in Internet of Things: Overview of the Design Challenges," in *Lecture Notes in Computer Science*, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 48–59–59.

[25] Y. Liu, Y. Meng, and J. Huang, "Gemini: A green deployment scheme for Internet of things," in *Wireless and Optical Communication Conference (WOCC), 2013 22nd*, 2013, pp. 338–343.

[26] D. Chen, M. Nixon, T. Lin, S. Han, X. Zhu, A. Mok, R. Xu, J. Deng, and A. Liu, "Over the air provisioning of industrial wireless devices using elliptic curve cryptography," in *Computer Science and Automation Engineering (CSAE), 2011 IEEE International Conference on*, 2011, pp. 594–600.

[27] A. Papageorgiou, M. Zahn, and E. Kovacs, "Auto-configuration System and Algorithms for Big Data-Enabled Internet-of-Things Platforms," *Big Data (BigData Congress), 2014 IEEE International Congress on*, pp. 490–497, 2014.

[28] C. Inzinger, S. Nastic, S. Sehic, M. Vögler, F. Li, and S. Dustdar, "MADCAT - A Methodology for Architecture and Deployment of Cloud Application Topologies," in *Proceedings of the 8th International Symposium on Service-Oriented System Engineering*. IEEE, 2014, pp. 13–22.

[16] http://pc3l.infosys.tuwien.ac.at/