Mobile Simulation Environment for Investigating IoT-Cloud Systems

Tamás Pflanzner

Supervisor:

Dr. Attila Kertész

A THESIS SUBMITTED FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
OF THE UNIVERSITY OF SZEGED

Doctoral School of Computer Science
University of Szeged
Szeged, 2020

Contents

1	Introduction	1
I	MobIoTSim	5
2	A Mobile IoT Device Simulator for IoT-Cloud Systems	7
2.1	The need for a mobile simulator	7
2.2	IoT simulators	8
2.3	A Survey of IoT Cloud Use Cases	11
2.3.1	Smart home applications	11
2.3.2	Smart city applications	12
2.3.3	Smart region applications	12
2.4	A Taxonomy of IoT Application Properties	13
2.4.1	Context	14
2.4.2	Participants	15
2.4.3	Sensors	16
2.4.4	Network requirements	17
2.4.5	Other properties	18
2.4.6	Categorizing the overviewed IoT applications	21
2.5	Cloud-based solutions for supporting IoT systems and applications . . .	21
2.6	The architecture of the proposed semi-simulator environment	24
2.7	The structure of MobIoTSim	24
2.8	Using MobIoTSim	27
2.9	Evaluation of the multiple device simulation scalability of MobIoTSim .	28
2.10	Summary	32
II	IoT Gateway	33
3	A Private Gateway for Investigating IoT Data Management	35
3.1	The need for private gateways in IoT systems	35
3.2	Related work in IoT Gateway solutions	36
3.3	Developing a private gateway	37
3.3.1	A comprehensive IoT-Cloud simulation environment	37
3.3.2	Evaluation of the proposed private gateway	38
3.4	Platform-independent gateway	42
3.5	Summary of contributions related to the generic IoT Gateway	43

4	Designing an IoT-Cloud Gateway for the Internet of Living Things	45
4.1	The IoT project and the need for a specialized gateway	45
4.2	IoT solutions for monitoring plants	46
4.3	The design of a Smart Pot for the IoT project	48
4.3.1	Implementation of the IoT Smart Pot Gateway	49
4.3.2	A solution for monitoring and analyzing plant growth over time	50
4.4	Evaluation of the Smart Pot Gateway	53
4.4.1	Simulations with a python tool	53
4.5	Real world measurements	59
4.6	Summary of contributions related to the IoT gateway	59
III	SUMMON	61
5	Analyzing IoT-Cloud Environments Using Real Sensor Data	63
5.1	The need for open data reuse of smart systems	63
5.2	Simulations supported by real-world data	64
5.3	Analyzing Sensor Data Formats and Contents of Smart Cities	64
5.3.1	Sensor Data in the SmartME project	64
5.3.2	Sensor Data in the CityPulse project	68
5.3.3	Sensor Data in the Smart City of Surrey	69
5.4	Filtered Datasets for Experimenting with IoT-Cloud Environments . .	71
5.5	Conclusion	75
6	A Crawling Approach to Facilitate Open IoT Data Archiving	77
6.1	The need for gathering dynamic IoT and sensor data	77
6.2	Extending SUMMON with a web crawling service	78
6.2.1	Case study 1: Idokep.hu	82
6.2.2	Case study 2: AA Traffic news	84
6.2.3	Case study 3: ParkMe	86
6.2.4	Evaluation	86
6.3	Conclusion	87
7	Conclusions	89
	Appendices	91
A	Summary in English	93
A.1	Introduction	93
A.2	New scientific results	94
	Summary	97
B	Magyar nyelvű összefoglaló	99
B.1.	Bevezetés	99
B.2.	Új tudományos eredmények	100
	Összefoglalás	103
	Bibliography	107

List of Tables

2.1	Comparison of related simulators	10
2.2	Comparison table of IoT Use Cases - Part I	19
2.3	Comparison table of IoT Use Cases - Part II	20
2.4	Data-related IoT-Cloud provider features.	22
2.5	Generic IoT Cloud provider features.	22
2.6	Device group simulation performance using different number of threads	31
3.1	Results of measurements with the Weather template.	41
3.2	Comparison of MQTT brokers	43
4.1	Comparison of commercial smart pot solutions	47
4.2	Comparison of the four evaluation rounds	57
4.3	Comparison of the 30 min. evaluation rounds	58
5.1	Sensor types in the Smart City of Messina	66
5.2	Temperature measurements in the Smart City of Messina	66
5.3	Brightness measurements in the Smart City of Messina	67
5.4	Humidity measurements in the Smart City of Messina	67
5.5	Pressure measurements in the Smart City of Messina	67
5.6	Noise measurements in the Smart City of Messina	68
5.7	Traffic light measurements in the Smart City of Messina	68
5.8	Measurements in the Smart City of Aarhus	69
5.9	Parking Data in the Smart City of Aarhus	70
5.10	Historical Climate Data of Surrey	70
5.11	Statistics on the river Fraser water levels dataset	70
5.12	Statistics on the river Fraser water level monitoring stations	71
6.1	Internals of the crawler service	82
6.2	The structure of the analyzed website	83
6.3	Comparison of the three use cases	86
7.1	Publications, Theses and Independent citations	90

List of Figures

2.1	The Context category of the taxonomy	14
2.2	The Participants category of the taxonomy	15
2.3	The Sensors category of the taxonomy	16
2.4	The Network requirements category of the taxonomy	17
2.5	The Other category of the taxonomy	18
2.6	A framework for simulating IoT-Cloud systems	23
2.7	MobIoTSim GUI: a.) On the left - Cloud settings, b.) In the middle - Devices, c.) On the right - Device settings	25
2.8	Screenshots of MobIoTSim connected to the private gateway: a.) On the left: Cloud settings, b.) On the right: Devices screen showing a warning	26
2.9	Data visualization in a private gateway service in Bluemix for a single device	27
2.10	Data visualization in the private gateway service in Bluemix for two groups of Thermostat devices showing median sensor values	27
2.11	MQTT scalability testing application	28
2.12	Number of sent messages and time between send cycles with 10 devices	29
2.13	Number of sent messages and time between send cycles with 15 devices	29
2.14	Number of sent messages and time between send cycles with 20 devices	30
2.15	Results with 0.5 and 0.25 second frequency	30
3.1	Evolution of cloud systems	36
3.2	Detailed, parameter-wise data visualization in the private gateway in the IBM Cloud sent by a group of devices in MobIoTSim	38
3.3	Initial evaluation with Thermostat devices	39
3.4	Evaluation results with CPU stressing	39
3.5	Comparison of different device groups	40
3.6	Comparison of different data generation intervals	40
3.7	Comparison of message data sizes	41
3.8	Comparison of CPU utilization of the two device templates	42
3.9	Platform-independent gateway architecture	43
4.1	IoLT Smart Pot prototype (left), and one pot of the cluster (right) . . .	48
4.2	The architecture of the IoLT Smart Pot Gateway	49
4.3	Historical sensor data visualization in the IoLT Smart Pot Gateway . .	50
4.4	Real and segmented pictures of the Smart Pot cluster taken at 2019.01.01	51
4.5	Projected leaf area values to be stored in the database of the gateway .	51
4.6	The gateway screen for querying detailed leaf area values	51
4.7	Selection of a pot from the cluster	52

4.8	Detailed leaf area values over time for a pot	52
4.9	Sample JSON message of seven sensor values of a pot	53
4.10	CPU and memory usage measurement results for 250 pots	53
4.11	Network and I/O usage measurement results for 250 pots	54
4.12	CPU and memory usage measurement results for 50 pot clusters	54
4.13	Network and I/O usage measurement results for 50 pot clusters	55
4.14	CPU and memory usage measurement results for 100 pot clusters	55
4.15	Network and I/O usage measurement results for 100 pot clusters	56
4.16	CPU and memory usage measurement results for 250 pot clusters	56
4.17	Network and I/O usage measurement results for 250 pot clusters	57
4.18	Data visualization of a real world measurement in the IoLT Smart Pot Gateway	58
4.19	Sensor data of a day in the IoLT Smart Pot Gateway	59
5.1	Sample temperature sensor data of SmartME	66
5.2	Sample traffic light sensor data of SmartME	69
5.3	Opening page of SUMMON	71
5.4	The basic operations of SUMMON	72
5.5	IoT environment simulation with SUMMON and MobIoTSim	73
5.6	A form generated by SUMMON to filter a dataset	74
5.7	Importing a file to MobIoTSim	74
6.1	An environment for simulating IoT-Cloud systems	78
6.2	Importing trace data to SUMMON through a REST API	79
6.3	Visualizing trace data gathered from OpenWeatherMap through the REST API	80
6.4	Architecture of the proposed crawler service	81
6.5	Web interface of the proposed crawler service	82
6.6	Data fields of a sensing message originated from Idokep.hu	83
6.7	Data fields of a sensing message originated from theaa.com	84
6.8	Executed measurements in the web interface	85
6.9	Data fields of a sensing message originated from parkme.com	85

To my beloved family...

1

Introduction

Cloud computing [95] has become a widespread and reliable solution over the past decade by providing scalable, virtualized data storage and computation. In the past decade, we experienced an evolution in Cloud Computing: the first clouds appeared in the form of a single virtualized datacenter, then broadened into a larger system of multiple, interconnected datacenters. As the next step, cloud bursting techniques were developed to share the resources of different clouds, then cloud federations were realized by interoperating formerly separate cloud systems. By overcoming the interoperability issues of public cloud providers and various middleware implementations, the process of creating and managing cloud federations was clarified and applied [94]. There are various reasons to optimize resource management in such federations: to serve more users simultaneously, to increase quality of service, to gain higher profit from resource renting, or to reduce energy consumption or CO_2 emissions. After solving optimization issues in federations addressing datacenter consolidation, operating costs, and energy efficiency, further research directions started to use clouds to support newly emerging domains, such as the Internet of Things (IoT) [105].

An IoT system is a form of a global, dynamic network infrastructure composed of smart devices and sensors having self configuring capabilities [130]. These things can interact and communicate through the Internet by exchanging sensor data, and they can react autonomously to certain events. We can also influence them by triggering actions without direct human intervention. It is obvious that such systems are suitable for numerous application areas having different properties and requirements. According to reports and predictions in the IoT field, there will be 30 billion devices always online and more than 200 billion devices discontinuously online by 2020 [115]. Huawei also foresees 100 billion new IoT connections till 2025 [90]. These estimations project the growing appearance of smart solutions in the near future, which should be capable of interconnecting and efficiently controlling the ecosystem of IoT devices.

Many cloud providers offer IoT-specific services nowadays, because cloud computing can potentially serve most IoT needs, including transparent data generation, processing, and visualization. As a result, the most popular cloud platform providers support basic IoT capabilities, but there are some unique features, as well. Designing and an-

alyzing IoT-Cloud environments still represent a great challenge (since many devices have to be managed at the same time), and these systems generally include a wide range of devices and utilize different data formats.

In the case of IoT-Cloud applications, data management operations are better placed close to their origins, thus close to the users, which results in better exploiting the edge devices of the network. Finally, as the latest step of this evolution, the group of such edge nodes formed the fog. Dastjerdi and Buyya defined fog computing as a distributed paradigm [103], where cloud storage and computational services are performed at the network edge. This new paradigm enables the execution of data processing and analytics application in a distributed way, possibly utilizing both cloud and nearby resources. The main goal is to achieve low latency, but it also brings novel challenges in real-time analytics, stream processing, power consumption, and security. Mukherjee et al. further detailed these challenges in [119], and stated that secure communication is a key issue and privacy-preserving data management schemes are needed.

The goal of my research was to support the proliferation of integrated IoT-Cloud technologies, by providing means for evaluating and demonstrating the inner workings of such ecosystems. By using my proposed tools, real things or devices can be substituted by mimicking their behavior, thus there is no need to perform initial investments by buying real sensors or devices to investigate IoT applications.

The scientific results I achieved and report in this dissertation are grouped into three theses, which are the followings:

- **Thesis I.** I designed and developed a mobile IoT simulator called MobIoTSim, which can simulate multiple IoT devices in a user-friendly way, and send the generated data on the real network to the cloud. I validated its usability, and evaluated its scalability for managing multiple devices simultaneously.
- **Thesis II.** I designed and developed a generic cloud gateway service, which can be deployed on any cloud service, and can manage IoT devices by receiving, storing and processing or visualizing data. I also validated its usability together with MobIoTSim, and evaluated its scalability.
- **Thesis III.** I designed and developed an IoT trace archiving service called SUMMON, which can periodically collect and store data from real IoT applications, even with using web crawling techniques. I demonstrated its operation with three use cases. The gathered datasets are shared in SUMMON to facilitate more realistic IoT simulations.

The structure of this dissertation also follows this grouping, hence it consists of three main parts. In the **first part**, I introduce the mobile IoT device simulator called MobIoTSim in **Chapter 2**. I provide an overview of cloud providers offering IoT features and define the main requirements for designing an IoT device simulator. Based on these requirements, we propose MobIoTSim for IoT device management, which is capable of simulating up to hundreds of IoT sensors and devices.

In the **second part**, which consists of two chapters, I propose a private cloud gateway service to manage both simulated and real IoT devices and sensors. In **Chapter 3**, I characterize the problem area of IoT data gathering and visualization, then describe how to develop a real, generic cloud gateway service to manage various devices composing a semi-simulated environment together with MobIoTSim. Finally, I evaluate

the performance and scalability of the proposed solution. In **Chapter 4**, I propose a specialized IoT-Cloud gateway for receiving, storing, and visualizing monitored environmental parameters of plants. This gateway can be used to receive data of sensors connected to a Raspberry Pi board forming a smart pot cluster to be used for low cost plant phenotyping. It is also able to perform image processing on the pictures of the plants to track plant growth. I have performed a detailed evaluation of our proposed platform by means of simulation, and exemplified real world utilization.

In the **third part**, which also consists of two chapters, I introduce an IoT trace archiving service, which can provide real world sensor data for MobIoTSim to perform realistic simulations. In **Chapter 5**, I analyze sensor data formats and samples of smart cities, and propose a data retrieval service for gathering and filtering open datasets. I analyze three smart city projects in detail, and show, how IoT data could be made available by SUMMON (Smart city Usage data Management and MONitoring) for simulating IoT-Cloud environments. In **Chapter 6**, I provide an extension to SUMMON with an automated web crawling service that gathers IoT and sensor data from publicly available dynamic websites, and exemplify its utilization with three use cases.

Finally, **Chapter 7** concludes the dissertation, while in Appendices A and B, I provide summaries in English and in Hungarian, highlighting the thesis points and the main contributions.

Over the past five years I had the opportunity to be part of a research team that started to build up a competence in combining Cloud and IoT technologies and providing simulation tools for investigating such ecosystems. This means that I achieved the results reported in this thesis not as an individual researcher, but as a team member. I am grateful to my colleagues for providing these collaboration opportunities. I will emphasize such team work in the body of my dissertation by detailing our joint results, meanwhile I will clearly state my own contributions at the end of the chapters, as well as in the thesis summaries.

Part I

MobIoTSim

2

A Mobile IoT Device Simulator for IoT-Cloud Systems

2.1 The need for a mobile simulator

The Cluster of European Research Projects on the Internet of Things [130] considers the Internet of Things (IoT) a vital part of Future Internet and they defined it as a dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols. Things in this network interact and communicate among themselves and with the environment by exchanging sensor data, they react to events autonomously and influence them by triggering actions with or without direct human intervention. According to another Gartner report [115] there will be 30 billion devices always online and more than 200 billion devices discontinuously online by 2020.

These trends and estimations call for an ecosystem that provides means to interconnect and control these devices. With the help of cloud solutions, user data can be stored in a remote location, and can be accessed from anywhere. The concept of cloud computing has been pioneered by commercial companies with the promise to allow elastic construction of virtual infrastructures, which attracted users early on. Its technical motivation has been introduced in [103]. Gubbi et al. [105] have identified that to support the IoT vision, the current computing paradigm needs to go beyond traditional mobile computing scenarios. Cloud computing has the potential to address these needs, as it is able to hide data generation, processing, and visualization tasks. For this reason, there are more and more Platform-as-a-Service (PaaS) cloud providers offering IoT specific services (e.g. Google Cloud Platform and IBM Bluemix Platform). Some of these IoT features are unique, but every IoT PaaS provider has the basic capabilities to connect and store data from devices. Many things have to be managed at the same time and a wide range of devices and data formats are available, therefore, creating and examining such applications is not trivial.

The aim of our research is to support the proliferation of IoT with the help of mobile and cloud technologies, thus to enable experimenting with complex systems

consisting of interdependent components that work together to enable the creation and management of user applications. To manage the heterogeneity of protocols and data structures used in these IoT-Cloud systems, smartphones and tablets can provide assistance. We discuss the design and development of a mobile IoT simulator called MobIoTSim that is capable of managing many things using the most popular IoT protocols (such as MQTT, HTTP) combined with different data structures/formats (e.g. JSON). It can substitute real things or devices by mimicking their behaviour. Therefore, the purpose of our proposed mobile IoT device simulator, is to help researchers and cloud application developers learn IoT device handling without buying real sensors, and to test and demonstrate IoT applications utilizing multiple devices.

Further advantages of this mobile approach are that mobile devices can act as micro-gateways to capture IoT data from sensors representing a semi-simulated environment, and outdoor IoT deployments can also be designed and evaluated in a more realistic way (e.g. using wireless mobile data communication instead of WiFi).

The goal of our research is to support the proliferation of integrated IoT-Cloud technologies, by providing means for evaluating and demonstrating the inner workings of such ecosystems. By using our proposed tools, real things or devices can be substituted by mimicking their behavior, thus there is no need to perform initial investments by buying real sensors or devices to investigate IoT applications.

The remainder of this chapter is as follows: Section 2.2 introduces an overview of related works and a comparison of simulators. Section 2.3 reveals properties of current IoT applications. Section 2.5 categorizes IoT-Cloud providers. Section 2.6 presents the architectural view of our proposal for simulating IoT-Cloud systems. Section 2.7 presents our proposed solution called MobIoTSim. In Section 2.8, we demonstrate the usability of MobIoTSim. In Section 2.9, we examine the device simulation scalability of the MobIoTSim application. Finally, the contributions are summarized in Section 2.10.

2.2 IoT simulators

Lately competition has started to increase among the leading cloud providers, like Amazon, Google and Microsoft, to attract users requiring services of these emerging technologies. This trend works against standardization, and results in incompatible interfaces and formats most of the time [114], both in the cloud and IoT fields.

There are already several related works available on categorizing IoT systems and applications. Gubbi et al. [105] were one of the pioneers by describing IoT systems in 2013, and they also emphasized the importance of the joint utilization of IoT and cloud systems. Botta et al. presented the main properties, features, and open issues of these systems in [92], while Nastic et al. [122] listed the numerous challenges of realizing IoT-Cloud systems in practice. These related works also serve as a motivation to our research by raising the need for managing a large number of protocols and data formats by means of simulation. Taylor et al. [131] identified that simulation projects of certain industrial applications require time consuming experimentation, therefore they proposed the CloudSME Simulation Platform to enable easier development and execution of commercial cloud-based simulations.

The existing simulators that examine distributed systems can be categorized as general network simulators, like NetSim [127], Qualnet [50] and OMNeT++ [133]. With these tools, IoT-related processes, such as device placement or message handling

can only be examined with great difficulty. The OMNeT++ discrete event simulation environment is a popular, generic tool for simulating communication networks and distributed systems [133]. It can be used in various domains ranging from queuing network simulations to wireless and ad-hoc network simulations.

There are also specialized IoT simulators addressing issues raised by this work. Chernyshev et al. [97] presented a recent survey of IoT simulators and testbeds. They argued that research in this area is especially challenging, and existing simulators are specialized in the sense that they focus on a particular architectural layer. One solution to overcome this could be to combine multiple specialized simulators at various levels. There are only a few large scale open testbeds to validate proof-of-concepts, therefore testbeds should become commoditized, making it possible to realize prototypes. Our research follows these findings and guidelines. Han et al. [107] designed DPWSim, which is a simulation toolkit to support the development of service-oriented and event-driven IoT applications with secure web service capabilities. SimIoT [129] was derived from the SimIC simulation framework [128]. It introduces several techniques to simulate the communication between an IoT sensor and the cloud, but it is limited to computing activity models.

Moschakis and Karatza [118] introduce several simulation concepts for IoT systems. First, they show how the interfacing between various cloud providers and IoT systems could be modeled (even including workload models) in a simulation. Compared to our proposal, they are focusing on the behavior of cloud systems that support the processing of data originating from an IoT system. Silva et al. [125] investigated the dynamic nature of IoT systems, hence they analyze fault behaviors and introduce a fault model for such systems. Although faults are important, the scalability of the introduced fault behaviors and concepts is insufficient for large scale systems.

Khan et al. [113] introduce a novel infrastructure coordination technique that supports the use of larger scale IoT systems. They build on top of CloudSim [96], and provide customizations that are tailored for their specific home automation scenarios, and as a result, they limited the applicability of their extensions. Zeng et al. [126] proposed a tool called IOTSim, which provides simulation capabilities for big data processing in IoT systems, but their solution is limited to applications using the MapReduce model.

The iFogSim [106] solution is also an extension to the CloudSim toolkit. It can be used to simulate IoT and fog environments by measuring resource management techniques with several metrics, such as latency, network congestion, energy consumption, and cost. They presented two case studies to demonstrate IoT modeling and resource management policies: the first is latency-sensitive online gaming, and the second is an intelligent surveillance application that is using distributed camera networks. The DISSECT-CF simulator [111] has also been extended with IoT features, and its operation is demonstrated with a meteorological case study [116]. Both approaches are simulating all components of an IoT-Cloud system, while our proposed device simulator can be connected to real cloud gateways, and sensor simulation can be done based on real world sensor data.

Concerning device simulation, the SimpleIoTSimulator [126] is an IoT sensor simulator, which can be used to set up and evaluate test environments having thousands of sensors and gateways, on a single computer. It supports many commonly used IoT protocols (e.g. CoAP, MQTT, HTTP). Its drawback is that it needs a specific, Red-Hat Linux environment, while our approach is more heterogeneous, and focuses on

IoT device simulation with mobile devices, which is easier to apply. Unfortunately, we found that the company is only engaging selected clients at this time, therefore, the tool is not yet available to the general public. The Atomiton simulator [58] seems to be very close to our concept based on its description. It manages virtual sensors, actuators, and devices with unique behaviors composing complex, dynamic systems. Unlike our mobile, open solution, however, it is commercial, and provides a web-based environment with limited documentation. Unfortunately, we were not able to perform a full registration to evaluate the solution (due to sign-up problems at their website).

Next, we present a comparison of the above mentioned simulators in Table 2.1. We used 6 categories to compare the main properties of these solutions. The "Cloud nodes / services" column uses the value "Simulated", if the application can only simulate cloud resources in a certain use case, or "Real", if the system can send data to a real application in the cloud, thus using a semi-simulated environment. The option "No inf." means we could not find any information about this feature. The "IoT sensors / devices" category has the "yes" value, when the actual tool supports the simulation of IoT devices. The "Artificially gen. sensor data" column denotes "yes", if the simulator can generate sensor data in an artificial way (e.g. using certain algorithms and deviation), and the "Real sensor / device data" provides the information about whether or not a real life IoT devices data can be used in the simulation. The "Learning curve" can be long, if the learning of the usage of the simulator and setting up a simulation environment takes a longer time, and short if the simulator can be used almost immediately after installation, in a self-explanatory way. The GUI support feature has the value "no", if there is no graphical interface provided, "limited", if the application requires code writing and editing for creating a visualized interface, and "yes", if it provides graphical interaction and configuration by default. We can see from this comparison that MobIoTSim brings novelty to the state-of-the-art with a graphical configuration and execution interface, offering a unique way for simulating real IoT sensor operation through loading trace data. Due to its mobile design, simulations can also be performed in wireless locations, composing a semi-simulated environment.

Table 2.1. Comparison of related simulators

Simulator	Cloud nodes / services	IoT sensors / devices	Artificially gen. sensor data	Real sensor / device data	Learning curve	GUI support
NetSim	Simulated	no	yes	no	long	yes
Qualnet	Simulated	no	yes	no	long	yes
OMNet++	Simulated	no	yes	no	long	limited
DPWSim	Simulated	yes	yes	no	long	yes
SimIoT	Simulated	yes	yes	no	long	limited
CloudSim	Simulated	yes	yes	no	long	no
IoTSim	Simulated	yes	yes	no	long	no
iFogSim	Simulated	yes	yes	no	long	yes
DISSECT-CF	Simulated	yes	yes	yes	long	no
SimpleIoTSim.	Real	yes	yes	limited	short	yes
Atomiton	No inf.	yes	yes	No inf.	short	yes
MobIoTSim	Real	yes	yes	yes	short	yes

2.3 A Survey of IoT Cloud Use Cases

In order to reveal properties of current IoT applications, we selected and investigated 20 IoT-Cloud use cases from various application areas coming from three main groups: smart home, smart city, and smart region. In this section we introduce and describe these relevant applications, and in the following section we study and gather real-world IoT properties of these cases, and compile them into a taxonomy.

2.3.1 Smart home applications

The Green Plug [10] is a smart home project that enables users to turn on and off appliances remotely. They also enable real time energy consumption by measuring current and voltage. A magnetic sensor is placed near the outlet, and the changes in the field can be used to calculate the electricity usage.

The Mimo [13] project develops smart home products that are created for the remote monitoring of babies. It measures the baby's body functions such as breathing, temperature, body position, and activity level. It can send occasional alerts and nightly reports to a smartphone by using an ultra low-power Bluetooth connection. Caretakers can see the sensor information in real-time. It also supports special features to interconnect other devices, e.g. if the baby's temperature is not optimal, a smart thermostat can change the room's temperature, or if the baby is moving, a webcam can be switched on to visually monitor it.

The Vitality GlowPack [22] solutions offer health-care smart home capabilities. They propose a smart pill bottle top as a replacement for the standard one. It connects via Bluetooth to the user's smartphone and an application reminds the patient to take the pills at the right time. There is a lamp unit feature of the product: the lamp and the bottle top start to flash if the bottle is not opened when the patient needs to take the medicine. If still unnoticed, it is able to play music or send an SMS, even initiating a phone call. Since pill taking is not such a frequent activity, there is no need for a high speed network connection for operation. There are additional features upon request: patients can be alerted if some pills cannot be taken at the same time, and the local pharmacy can also be interconnected to order additional medicine when the user runs out of them.

Nest Thermostat [14] is one of the Nest products for smart homes. With this smart thermostat one can control rooms' temperatures in a house. The use of a single thermostat does not require high-speed network connection. Their concept is that the IoT devices can interact without the need of a configuration. Supported devices like smart lamps can use the already implemented Nest Thermostat and use its data to learn from it. There are more and more new products with Nest support, which is a good example of more devices working together. The Nest API helps the interaction between different kinds of network capabilities to work together.

Smart Outlets [19] are designed to implement the smart electrical outlet concept. A user can remotely control the appliances, or set timers from a smartphone. The number of IoT devices can be varied, but in average we can say the application needs a medium sized environment. It can communicate with different networks, but WIFI is the default setting. Its monitoring and energy saving capabilities are very useful.

Key Finder Tags [12] are location sensors that can be attached to utilities as key fobs or stick-on tags. The default version has a Bluetooth connection and can make a

beeping sound or light signals to guide users. Other versions have their own cellular data connection and GPS sensors, so that they can also report exact localization information. Using this feature, smartphones can request devices to show their positions. Advanced tags support a useful reverse (actuator) function: if the user has the smart key fob, but their smartphone is missing, when pushing the fob, the phone will signal its position.

The Samsung company offers many smart home solutions. Their smart refrigerator [18] is a good example. It has a big touchscreen, where the family calendar can be shown, note taking and photo displays are also supported. It has three built-in cameras, therefore, every time the door closes, newly taken photos can be sent to registered smartphones.

2.3.2 Smart city applications

The SmartPile [21] system is an interesting solution for monitoring the condition and quality of the concrete in building constructions. It offers wireless devices to be built into the concrete and later gives information during curing (e.g. core and skin temperature and compression strain), transport, and installation (e.g. compression, strain and load capacity).

Wireless Plant Sensors [23] help with taking better care of plants. They can be composed to build indoor or outdoor sensor systems using WIFI connection to send status information about the monitored plants. They provide different algorithms for different plant types to water them. They also offer timers so users won't forget to water the plants. The size of the system may vary from few home plants to an industry sized system.

The Bigbelly [3] is a smart waste and recycling system that helps to figure out if a particular trash can needs to be emptied. It is a solar powered system, so no electricity is needed for fullness level sensing. It offers communication with the Bigbelly cloud. The system is designed to provide smart trash cans from home to city scale.

The Smart Parking [20] system can detect if the parking spot is reserved with a magnetic sensor. There is an option to send the data to the Save9 cloud through a cellular network and use it to provide smart parking solutions.

Echelon [7] offers an open standards-based approach to lighting control. Outdoor lighting is an important function for enterprises and cities to increase the efficiency and safety of their lighting systems.

The Optoi chemical sensors [16] are silicon devices mainly based on microelectrodes and specific sensing layers, such as silicon nitride, for pH measurement. These water monitoring devices can be integrated into a multiparametric microsystem together with Conductivity, Red-Ox potential, Temperature and other sensors for water monitoring applications.

2.3.3 Smart region applications

The project called Assessment of Landslides using Acoustic Real-time Monitoring Systems (ALARMS) [2] utilizes low cost, miniaturised acoustic sensors integrated with wireless networking capability (GSM) to provide an early warning of slope instability. If the acoustic soil activity reaches a predefined threshold, the system sends an SMS message. ALARMS uses a steel tube called a wave guide to conduct the signals

out of the ground. The steel tube is placed in a borehole in the ground that is filled with sand or gravel. The sand and gravel produces more energy when moved than the surrounding soil, which makes the signal easier to detect.

An example application for wildlife tracking is the Open Source Lion Tracking Collars (OSLTC) [15], which is an open source collar system to help conservationists protect the last lions living in the wild in Southern Kenya. It is also used to safeguard the Maasai herders' cattle, with the aim of restoring Maasai-land to a working ecosystem.

With the Phenonet Project [17], plant breeders can evaluate the performance of different wheat varieties using measurements taken from remote sensors. These sensors monitor properties like soil temperature, humidity, and air temperature and are often used for crop variety trials. This allows farmers to forecast harvest time, improve plant health, plan irrigation time, and determine frost and heat events.

Companies like Govtech [9], AquamatiX [1] and i2O [11] provide solutions for cities to better control the flow of water by embedding sensors in water pipes throughout the distribution network and connecting them to pump control systems. These sensors monitor water flow, feeding the data back to facilitate optimized water pumping throughout the system. By minimizing the amount of water in the pipes, cities can reduce the amount lost to leakage and prevent the formation of new leaks. In the process, the system also saves energy by reducing the need for pumping. Moreover, by distributing water monitoring throughout the network, these technologies can detect abrupt events, like bursts, facilitating faster response and minimizing water loss.

The latest GE Evolution Series Tier 4 Locomotive [8] is equipped with 250 sensors to measure a staggering 150,000 data points per minute. This data is combined with other incoming streams of data from informational and operating systems to monitor anticipated events and to support taking driving decisions in real time.

Caterpillar [4] is helping its dealers succeed with help of IoT for industrial analytics. The company is harnessing the data it collects from its industrial locomotives like engines, machines and tools and shares the analyzed data insights with its customers. It helps them anticipate problems, manage fleets and schedule maintenance proactively.

Based in Ireland, the CleanGrow's [5] project helps with monitoring the crop nutrients making use of a carbon nanotube-based sensor system. This information helps farmers to alter the maturity rate or color of the crop production. As opposed to the conventionally used analog devices, the CleanGrow device uses a nanotube sensor that detects quantity and presence of specific ion in the production.

OpenWeatherMap [46] is a service, which collects Weather data from thousands of remote ground sensors distributed all around the world and is expanding permanently. The collected data can be accessed with an API with more than 200,000 cities. A weather forecast service is implemented based on the sensor data.

2.4 A Taxonomy of IoT Application Properties

Based on the survey presented in the previous section, we gathered the most relevant properties of IoT applications and classified them into categories of our proposed taxonomy. We revealed five top-level categories: Context, Participants, Sensors, Network, and Other.

2.4.1 Context

Figure 2.1. The Context category of the taxonomy

We named the first top-level category *Context*, its structure is depicted in Figure 2.1. This is also the most usual viewpoint in earlier related works. Its categorization is based on the types and fields of the overviewed applications. From this category one can determine the key features of an application, considering its scale and area of utilization. One of its sub-categories is the *Body*, which contains use cases providing IoT sensors measuring body functions. The purpose of measuring can be related to health, e.g. when patients are examined by doctors, these cases are covered by the element *Health*. The *Sport* element is supposed to cover the training-related applications. Various sporting communities utilize these smart devices to optimize their trainings and track performance, but it is also very popular among amateurs, because of the social network sharing habits. *Sport* and *Health* use cases are close to each-other, but usually the precision is not critical in the first case, it is rather the durability that matters. The third element of the *Body* sub-category is *Comfort*, where corresponding applications help the everyday life to become more relaxed and efficient.

The next sub-category is *Building*, where the sensors are placed inside or attached to buildings. These applications generally utilize sensors, but there are cases where they use actuators to help the automation of smart homes. Their main purpose is to improve personal life (represented by element *Home*) – since these applications are targeting our homes –, or to improve professional life by supporting a home office environment (element *Office*). Smart homes are one of the first use cases of the IoT world, therefore, nowadays we can find some mature solutions in this field.

There are many use cases related to cities and municipalities, where data from a local geographical area is collected and analyzed – denoted by the *City* element. The main challenge is the growth of the distance between the sensors and gateways (which is also a decision point while planning IoT systems). The *Industry* sub-category is usually covered by factories that want to optimize resources and operational costs. The roots of IoT are in Wireless Sensor Networks, which solutions have been highly utilized in factories. The *Environment* sub-category aims to understand and manage natural resources like water, monitor wildlife, or operate wider-scale agricultural systems.

2.4.2 Participants

Figure 2.2. The Participants category of the taxonomy

Besides geographical coverage, the number of potential users and devices (or sensors) is also an important property of IoT applications – we named this category *Participants* (see Figure 2.2). By analyzing these numbers we can differentiate three category elements: *small-scale* applications, where there are generally less than 10 sensors utilized. *Medium-scale* applications are considering the sensor range of 10 to 100, where the handling of devices require special care, e.g. device grouping. Finally, *large-scale* applications contain more than 100 IoT components. This category covers use cases with a large number of devices, therefore, system scalability becomes critical here.

The data handling approaches depend on the number of users, which is covered by the element: *Number of Users*. Access control is usually not a critical issue nowadays, but the way the managed data is to be published/provided requires special attention. Similarly to the previous element, we can identify three main groups: *small-scale* with less than 5 users, *medium-scale* with the range of 5 to 50 users, and, if the data has to be available for more than 50 users, we are talking about *large-scale* applications.

Figure 2.3. The Sensors category of the taxonomy

2.4.3 Sensors

The biggest top-level category is the one called *Sensors*, which is used to define all sensor-related properties as shown in Figure 2.3.

The main purpose of sensors is to measure something to work with. We revealed four sub-categories: *Vision*, *Position*, *Physical* and *Other*. *Vision* contains sensors capable of monitoring visual data, hence we have the following elements: *Photo*, *Video* and *Light* sensors. The *Position* sub-category refers to sensor placement and physical location changes having the elements: *State*, *Local*, and *Global*. The open/close or tilt sensors are represented by *State* sensors. *Local* sensors provide information about position changes within a local environment, denoted by elements: *Motion*, *Velocity*, and *Displacement*. The *Global* sensor positioning contains the widely applied GPS sensor. The *Physical* sub-category covers sensors monitoring the physical world. It has the following elements: *Force*, *Load*, *Torque*, *Strain*, and *Pressure* sensors. Finally, the *Other* sub-category aims to cover the remaining, widely used sensor areas by the

Acoustic, Temperature, Flow, Humidity, Chemical, Electric, and Magnetic elements.

It is typical to combine more sensor types to create special-purpose devices for solving a problem. In the comparison table of use cases (to be shown later in Figures 2.2 and 2.3), we only highlight the most relevant sensors, but in the application descriptions, additional (composed) sensor types can also be found where applicable.

The type of the sensor and additional configuration information can be used in the cloud application for validating the data, noticing hardware errors or just filtering a few damaged data packets. For example, there are sensors which send data periodically and sensors that only send data if some event happens. If there is no data from a sensor that should send data in every second, the application can suspect some network error. Furthermore, the cloud application has to know the type of the sensor in order to know what kind of commands it can handle.

2.4.4 Network requirements

Figure 2.4. The Network requirements category of the taxonomy

Next to *Sensors*, the other most important category is *Network requirements*. This category has five sub-categories to cover network-related properties. The *Network Speed* requirement differentiates applications into three groups: *Slow*, *Medium*, and *Fast*. To be more precise on network requirements we can further categorize the applications based on *Message Size* and data generation *Frequency*. The *Message Size* sub-category

can be *Small*, *Medium*, or *Large* based on the size of one sensor data sample. The *Frequency* can be *Rare*, *Medium*, or *Often*, referring to the sampling timings. The *Network Speed* requirement usually correlates to the *Message Size* and the *Frequency* properties, but there are examples for unexpected setups. For example, if a medium size message is rarely sent to the cloud, the network speed requirement could be slow or medium, but if the data is important to be refreshed for a trigger, for example in a security device, the network speed should be fast. The *Connection type* sub-category contains many protocols, and they can be separated into two groups: *Short range* and *Long range* connections. The *RFID*, *NFC*, *Bluetooth*, *Zigbee*, and *Z-wave* are considered to be short range types, while *Cellular network*, *Wifi* and *TCP/IP* belong to the large range group. The final sub-category related to network requirements is the *Sensor availability*. We can talk about *Fixed devices*, where the location of the sensors is constant, or *Moving devices*, where the devices can change positions relative to each other. The other possibility is an *Ad-hoc* network, where the devices can connect and disconnect any time.

2.4.5 Other properties

Figure 2.5. The Other category of the taxonomy

The previously unclassified properties are grouped together into the *Other* category shown in Figure 2.5. Its elements differentiate IoT applications with three sub-categories: the *Connect to* sub-category defines where the IoT application sends its data to. It can be transferred directly to a smartphone or other *Mobile* device, to a *Local gateway* service (possibly running on a local, physical device), or to a cloud gateway running on a *Private cloud* or *Public cloud*.

The *Energy consumption* sub-category reveals information on power utilization, which can be *Low* or *High*. This is an important factor for cost efficient systems, and it

Table 2.2. Comparison table of IoT Use Cases - Part I

Name	Context	Nr. of Devices	Nr. of Users	Sensors	Connects to	Energy con.
Mimo [13]	Body / Health	Small	Small	Motion	Mobile	High
Vitality GP [22]	Body / Health	Small	Small	Open / Close	Mobile	Low
Nest Thermostat [14]	Building	Medium	Small	Temp.	Private cloud	Low
Smart Outlets [19]	Building / Office	Medium	Medium	Electric	Private cloud	Low
Key Finder Tags [12]	Building	Small	Small	GPS	Mobile	Low
W. Plant Sensors [23]	Industry	Medium	Medium	Photo	Public cloud	High
Bigbelly [3]	City	Medium	Small	Load	Private cloud	Low
Echelon [7]	City	Large	Small	Light	Private cloud	High
ALARMS [2]	Environment	Large	Large	Acoustic	Public cloud	Low
OSLTC [15]	Environment	Medium	Small	GPS	Private cloud	Low
GreenPlug [10]	Building	Medium	Small	Magnetic	Mobile	Low
Smart Parking S. [20]	City	Large	Medium	Magnetic	Private cloud	Low
Optoi [16]	City	Small	Small	Chemical	Private cloud	Low
Phenonet Project [17]	Environment	Medium	Small	Humidity	Public cloud	Low
GovTech [9]	City	Medium	Small	Flow	Private cloud	Low
AquamatiX [1]	City	Medium	Small	Flow	Private cloud	Low
i2O [11]	City	Medium	Small	Flow	Private cloud	Low
Smart Fridge [18]	Building	Small	Small	Photo	Private cloud	Low
SmartPile [21]	Building	Large	Small	Temp.	Private cloud	Low
Locomotive [8]	Industry	Large	Small	Electric	Private cloud	High
Caterpillar [4]	Industry	Medium	Medium	Electric	Private cloud	High
CleanGrow's [5]	Environment	Large	Medium	Chemical	Private cloud	Low
OpenWeather-Map [46]	Environment	Large	Large	Temp.	Public cloud	Low

Table 2.3. Comparison table of IoT Use Cases - Part II

Name	Security	Network speed	Message size	Freq.	Connec-tion	Sensor
Mimo [13]	Protected	Medium	Medium	Often	Short range	Moving devices
Vitality GP [22]	Protected	Slow	Small	Rare	Short range	Moving devices
Nest Thermostat [14]	Protected	Slow	Small	Rare	Short range	Fix dev.
Smart Outlets [19]	Protected	Slow	Small	Rare	Long range	Fix dev.
Key Finder Tags [12]	Secret	Slow	Small	Rare	Short range	Moving devices
Wireless Plant Sensors [23]	Public	Medium	Medium	Medium	Long range	Fix dev.
Bigbelly [3]	Public	Slow	Small	Rare	Long range	Fix dev.
Echelon [7]	Protected	Slow	Small	Medium	Long range	Fix dev.
ALARMS [2]	Public	Medium	Small	Often	Long range	Fix dev.
OSLTC [15]	Protected	Slow	Small	Medium	Long range	Moving devices
Green Plug [10]	Protected	Slow	Small	Medium	Short range	Fix dev.
Smart Parking [20]	Public	Slow	Small	Rare	Long range	Fix dev.
Optoi [16]	Protected	Slow	Small	Rare	Long range	Fix dev.
Phenonet Project [17]	Public	Slow	Small	Rare	Long range	Fix dev.
GovTech [9]	Protected	Medium	Small	Medium	Long range	Fix dev.
AquamatiX [1]	Protected	Medium	Small	Medium	Long range	Fix dev.
i2O [11]	Protected	Medium	Small	Medium	Long range	Fix dev.
Smart Fridge [18]	Protected	Medium	Medium	Rare	Long range	Fix dev.
SmartPile [21]	Protected	Slow	Small	Rare	Short range	Moving devices
Locomotive [8]	Protected	Medium	Medium	Often	Long range	Moving devices
Caterpillar [4]	Protected	Medium	Medium	Medium	Long range	Moving devices
CleanGrow's [5]	Protected	Slow	Small	Rare	Long range	Moving devices
OpenWeatherMap [46]	Public	Slow	Small	Rare	Long range	Fix dev.

reminds managers to calculate with battery lifetime for moving sensors or large-scale networks. The Security sub-category refers to the security-level of data handling. Here the data management can be *Secret*, *Protected*, or *Public*. If the data is *Secret*, it means that only the owner can access it, for example if it is a personal item, like a medicine handling device. The validity of the data is important, too. The *Protected* data means that the data can be shared with other people, but these people must identify themselves. The *Public* data does not require any authentication to access it.

2.4.6 Categorizing the overviewed IoT applications

Finally, we categorized the previously seen IoT applications into the presented taxonomy. In Figures 2.2 and 2.3, we map the IoT use cases introduced in the survey to the categories of the taxonomy.

It can be seen from the tables that the number of users is usually small-scale and the number of devices is usually medium-scale. This can be explained with the complexity of deploying large-scale systems. The sensor types and the context of the use cases are really diverse; this comes from the nature of these systems to be everywhere and help with our lives. Usually, we are talking about devices with low energy consumption, because they usually work with a battery. Regarding the network, we can say the frequency and size of the data is not too big, but if we take into account the type of the network, it can be seen that wireless networks are in the majority, and the bandwidth and error rate of these networks are not as good as in wired networks.

2.5 Cloud-based solutions for supporting IoT systems and applications

One of the main motivating forces behind our research is that several cloud providers have started to offer IoT specific services to ease the development of IoT-Cloud applications, but such systems, where many different things need to be operated and managed at the same time, are hard to realize. Just to mention a few, smart city application scenarios using SIGFOX [53] or LoRa [41] technologies are very expensive and time-consuming to deploy with real devices, since a base station costs up to a thousand Euros, and needs enormous configuration work. As a result, we propose to use simulated devices with real cloud gateways by creating a semi-simulated environment.

In an earlier work, we also presented a categorization, in which we highlighted IoT-specific features of (originally) cloud providers [P6]. Revisiting and extending these results, we gathered the main IoT-related properties of the most popular cloud providers in Table 2.4 and 2.5, which provide Platform-as-a-Service (PaaS) solutions for IoT applications. We used the following properties and notations in the tables to compare the selected providers. A *Provider* is the name of the actual IoT-Cloud service provider. With *Protocols*, we summarized the supported IoT device protocols for connections. *Data store* is supported, if the system can store data coming from devices, while *BLOB* means the system can store binary data, such as images. If the system supports storing and querying geolocation coordinates, it is denoted by *GEO*. Finally, we state whether or not *Push notification*, action *Triggering* and device data *Visualization* is supported by default.

Summarizing these comparison tables, Bluemix, Azure, Google, and Amazon have

Table 2.4. Data-related IoT-Cloud provider features.

Provider	Protocols	Data store	BLOB
Bluemix [36]	MQTT	+	-
Parse [47]	REST	+	+
Google [34]	REST	+	+
Amazon [26]	MQTT, REST	+	+
Azure [27]	MQTT, AMQP, REST	+	+
Heroku [35]	MQTT	+	+
CloudFoundry [6]	REST	+	+
Kinvey [32]	REST	+	+
DreamFactory [33]	REST	+	+

Table 2.5. Generic IoT Cloud provider features.

Provider	GEO	Push not.	Trigger	Visualization
Bluemix [36]	+	+	+	+
Parse [47]	+	+	+	-
Google [34]	+	+	+	+
Amazon [26]	+	+	+	+
Azure [27]	+	+	+	N/A
Heroku [35]	N/A	-	-	N/A
CloudFoundry [6]	N/A	-	-	-
Kinvey [32]	N/A	+	+	N/A
DreamFactory [33]	N/A	+	+	N/A

the highest variety of IoT-related services. The MQTT protocol is widely supported (which is also expected), but there are still some providers that only offer REST interfaces. From this survey, we can see that the most popular cloud providers have already realized the need for IoT support, and most of them provide reasonably good solutions for IoT application development. Nevertheless, interoperability issues still exist, and applications managing a large number of different IoT devices are hard to develop and evaluate.

By conducting this study of the available IoT-Cloud providers, we found IBM Bluemix [36] the most appealing by offering easy-to-use and well documented IoT services. Bluemix has also identified the need for a sensor or device simulator, but its proposed tool is meant to serve simple demonstration purposes, while our aim is to design a generic solution to this problem. Nevertheless, as we will see later, we tried to rely on and use some of the supporting services of Bluemix.

There are different kinds of IoT environments, hence their static or dynamic properties, and the number of utilized devices can affect the design of such a simulator. For example, a connected house can be regarded as a static environment, because its devices are usually in one place, possibly with a wired connection, providing reliable network stability. The dynamic environment is more complex to simulate, in such cases we would like to simulate a broader part of the environment considering WiFi interference, battery lifetime, and locations of the devices. From this classification we

can derive the basic requirements for an IoT-Cloud simulator. Since things are usually battery powered, it would be useful to represent energy metering or monitoring. Many sensors communicate through low-power wireless technologies, therefore, at least the most widespread ones need to be covered. Various resource constraints are applied to low-level devices (e.g. on processor, memory, or connectivity), which means we need to handle different device types. IoT systems are very dynamic as network conditions may change rapidly, and also heterogeneous at the same time.

We are not aiming at simulating all possible IoT systems and networks, but we still want to aid the design, development, and testing processes of these systems. Our goal is to develop a mobile IoT device simulator that can emulate real devices and sensors, thus it can be used in the previously mentioned processes instead of real resources.

The requirements for basic functionalities of such a simulator are to send and receive messages, generate sensor data (for one or more devices), and react to received messages. These capabilities are sufficient for the the simulator to be used in IoT system analysis. Requirements for advanced functionalities (such as simulating network errors, recording and replaying concrete simulation cases, and connecting real IoT devices to the simulator) can contribute to the analysis of more realistic systems.

We aimed at supporting the basic IoT functionalities. To this end, a simulated device should have a unique identifier, as well as tokens for registration and authentication. The generated sensor data should be made available in plain text, JSON, or binary format, including metadata information, such as date, time, and device state. Lastly, preferably MQTT, or REST communication protocols should be supported.

Figure 2.6. A framework for simulating IoT-Cloud systems

By examining related simulators in the IoT field, we found that we wanted to have a solution that puts an emphasis on mobility, and provides means for performing simulations at specific geolocations. This fact served as the major motivation for designing our simulator for mobile devices. The main purpose of our proposed mobile

IoT device simulator, called MobIoTSim, is to help cloud application developers learn IoT device handling without buying real sensors, and to test and demonstrate IoT applications utilizing multiple devices. The design of this application lets users create IoT environment simulations in a fast and efficient way with the option for custom settings. Since the first, initial version of MobIoTSim, we have made various feature improvements and releases to the simulator itself, and have developed enhanced, private cloud gateways. Next, we detail these contributions.

2.6 The architecture of the proposed semi-simulator environment

The architectural view of our proposal for simulating IoT-Cloud systems is depicted in Figure 2.6. Compared to traditional simulators, we propose a semi-simulated environment to stay as close to real world systems as possible. We started to gather real sensor trace files of public IoT initiatives, and made them available through an archive (I.), which can be connected to MobIoTSim (II.) to replay real device behavior. We also developed gateway services (III.) to manage IoT devices by processing and visualizing sensor data. These gateways can be instantiated and operated at private or public cloud or fog providers.

Our mobile IoT device simulator is designed to simulate up to hundreds of IoT devices, and it is implemented as a mobile application for Android platforms. Sensor data generation of the simulated devices can be done through artificially generated, random values in a range given by the user – this is the default option. For more advanced simulations, data from IoT-specific trace files can also be loaded and replayed. The data sending frequency can be specified for each device for both cases. The application uses MQTT protocol (by default) to send messages with the use of the Eclipse Paho library, which is opensource. The MQTT protocol is an established messaging protocol for IoT systems [121] and widely supported by IoT-Cloud services. Message data is represented in a structured JSON object in the simulator, compatible with the IBM IoT Foundation message format [37].

2.7 The structure of MobIoTSim

The basic usage of the simulator is to: (i) connect the application to a cloud, where the data is to be sent, (ii) create and configure the devices to be simulated, and (iii) start the (data generation of the) required devices. These main steps are represented by the three main parts of the application: the *Cloud settings*, the *Devices*, and the *Device settings* screens – as shown in Figure 2.7.

In the *Cloud settings* screen (Figure 2.7.a), the user can set the required information about the target cloud, where the data will be received and processed. Currently, we support two clouds: we developed gateways for the IBM Cloud (formerly Bluemix) and the Microsoft Azure providers. Concerning Bluemix, MobIoTSim users can use the so-called Quickstart application and a standard Bluemix IoT gateway service [36]. The Bluemix Quickstart is a public demonstration application, it can visualize the data received from a single, selected device. For a fast and easy solution, e.g. to try out the simulator for the first time, this type is recommended. The standard Bluemix

Figure 2.7. MobIoTSim GUI: a.) On the left - Cloud settings, b.) In the middle - Devices, c.) On the right - Device settings

IoT service can be used, once a user has an account for the Bluemix platform, and has already created an IoT service. This IoT service can be used to handle devices, which have been registered before. If a simulated device wants to send data to the Bluemix IoT service, it has to be registered beforehand. The registered devices have device IDs and tokens for authentication. The MobIoTSim application handles the device registration to the cloud with REST API calls, so the user does not have to register the devices manually on the graphical web interface. The main part of the IoT service is an MQTT broker; this is where device messages are received, and are possibly forwarded to other cloud services. Such cloud services or applications can process the data, react to it or just perform some visualization tasks. The required configuration parameters for the standard Bluemix IoT service in MobIoTSim are: the Organization ID, which is the identifier of the IoT service of the user in Bluemix, an authentication key, so that the user does not have to register the devices on the Bluemix web interface, and the command and event IDs, which are customizable parts of the used MQTT topics to send messages from the devices to the cloud and occasionally backwards. MobIoTSim can register the created devices with these parameters automatically, by using the REST interface of Bluemix. In some cases, the user may want to send sensor data with MobIoTSim not only to one, but to more cloud gateways at the same time. This is also possible by changing the organization ID attribute of a device to one of the already saved ones in the cloud settings.

In the *Devices* screen (Figure 2.7.b), we can create or import devices, and also view the list of existing devices, where every row is a device or a device group. These entities can be started and stopped by the user at will, both together or separately for the selected ones. Some devices have the ability to display warnings and notifications coming back from the gateway, backward communication is also supported. By clicking on the Add button we can create new devices or device groups. Predefined device templates can also be used for this purpose. These device templates help create often used devices, such as temperature sensors, humidity sensors, or a thermostat. The

import feature is applicable when we previously saved a device to a file with its sent messages for a certain simulation period.

New device creation and the editing of an existing one can be done in the *Device settings* (or *Edit device*) screen (Figure 2.7.c). If the user selects an existing template for the base of the device in the *Type* field, the message content and frequency will be set to some predefined values. For example, the *Thermostat* template has a temperature parameter, and the device turns on by reaching a predefined low-level value and turns off at reaching a high-level value. The On/Off state of the device is displayed on the screen of the application all the time. It is possible to select the *Custom* template to configure a unique device with various sensor parameters in detail. By default, random values will be generated within the set ranges as parameters. A device group is a group of devices with the same base template, and they can be started and stopped together. A group can be defined by setting the value of the *Number of devices* field (which is one by default).

Figure 2.8. Screenshots of MobIoTSim connected to the private gateway: a.) On the left: Cloud settings, b.) On the right: Devices screen showing a warning

Sensor data generation for the simulated devices can be done in different ways:

- The default way to generate device data is to create a custom device, where the user can define parameters with ranges, and a random value will be generated within the range (by default the *Generate random data* option is activated). To simplify this process the user can use device templates, which have predefined parameters and ranges, specific to the device type.
- The user can also choose from the predefined, already existing template devices. This way, the sensor parameters and their ranges are already set, and an artificial, random value generation will also be performed.
- If the *Generate random data* option is deactivated, the user can select a file to load the sensor values. In this way, we can replay previous experiments.

- Beside using random generation and replays, we can even load the trace files of real world applications gathered from public sources. Our first trace file example comes from the OpenWeatherMap public weather data provider [46], which monitors multiple cities and stores many parameters of them, including temperature, humidity, air pressure, and wind speed. Using the trace loader feature of MobIoTSim, the simulation can mimic a real life scenario.

2.8 Using MobIoTSim

To demonstrate the usability of MobIoTSim, we will show how to connect it to the default IBM Quickstart application having an MQTT server, running in the IBM Cloud. First, we need to create a device with the settings shown in Figure 2.7.a, then we can register it to the MQTT server of the IBM Cloud. Finally, we can start the device in MobIoTSim, like we did for MobIoT_test01, as shown in Figure 2.7.b. From this step forward, the data generated by the device is continuously sent to the demo gateway.

Figure 2.9. Data visualization in a private gateway service in Bluemix for a single device

Figure 2.10. Data visualization in the private gateway service in Bluemix for two groups of Thermostat devices showing median sensor values

We also developed a private gateway service for the IBM Cloud (see [48]). It is capable of managing numerous devices simultaneously, and can it can send notifications to simulated devices in MobIoTSim, e.g. by responding to critical sensor values. This gateway service is an extended version of the IBM visualization application [38]. It has a web-based, graphical interface to visualize sensor data coming from MobIoTSim, and it uses charts to depict the received sensor values as shown in Figure and 2.10. Messages (defined in JSON format) received from the simulated devices are managed by an MQTT server. It can also be used to send responses (or notifications) back to the simulated IoT devices in MobIoTSim.

We have also created a gateway service in Azure IoT Hub [27], and connected MobIoTSim to it. In this way, we can envision an inter-cloud scenario, in which simulated devices in MobIoTSim can send data to gateways in Bluemix and Azure clouds simultaneously.

Figure 2.11. MQTT scalability testing application

2.9 Evaluation of the multiple device simulation scalability of MobIoTSim

We examine the device simulation scalability of the MobIoTSim application itself. For this purpose, we created a simplified version of MobIoTSim (called MQTTDemo), containing only its core parts for device management functionalities. The screenshot of the application is shown in Figure 2.11. It enables access to low-level configurations, like specifying the number of threads used for managing the simulated devices. It also

collects detailed statistical information regarding the simulation by measuring elapsed times for executing certain functions. It can be connected to our private gateway deployed in the IBM Cloud, and can send messages to it using the MQTT protocol. By the time we performed the evaluations, the number of simulated devices were limited to 20 by the IBM Cloud platform.

Figure 2.12. Number of sent messages and time between send cycles with 10 devices

Figure 2.13. Number of sent messages and time between send cycles with 15 devices

In the Settings screen (as shown in Figure 2.11) we can define the number of devices to be simulated, the message frequency for sensor data generation, the number of used threads (*ThreadPool*), and the message type and contents. This type can be a simple JSON object with a random parameter, or an OpenWeatherMap device data in JSON format describing sensor values of certain smart cities. Other settings are hard-coded to the application for this testing, as well as the address of the MQTT broker of the IBM Cloud service. After performing a test, the resulting statistics can be accessed by pressing the STAT button.

Figure 2.14. Number of sent messages and time between send cycles with 20 devices

Figure 2.15. Results with 0.5 and 0.25 second frequency

The screen showing statistical data has two main parts: the Settings and the Statistics views. In the Settings view, we can find the number of used threads, the address of the MQTT broker, the number of simulated devices, the frequency of the messages, the generated clientID of one device, the MQTT topic where the messages were sent, the content type, and an example JSON message. The Statistical view displays the measured values, such as the duration of the simulation, the overall messages sent and the total number of errors in the initialization, connection, or send process of the devices. It also depicts the average times of the MQTT initialization, connection, and message send methods, and the average overall time cost of a message exchange cycle. An important measure is the real time difference between the process cycles of a device. If the message frequency is set to 1 second in the settings view, we should be able to verify it during testing. It can happen that during the simulation the simulated device cannot start the message sending method every second, because other devices hold the required resources (such as the CPU). As a result, the average time for creating a randomly generated message is also measured.

The evaluation was performed with a Samsung S5 device, and a node.js visualization gateway application running in the IBM Cloud. The Samsung S5 was released in 2014 having Snapdragon 801, Quad-core 2.5 GHz processor, 2 GB RAM and the OS is Android 4.4. The most informative measurements were made with 1 sec message

Table 2.6. Device group simulation performance using different number of threads

Used threads	Num. of devices	Message count	Send time	Time between cycles
3	10	441	405	1374
4	10	588	403	1023
6	10	610	431	991
8	10	603	435	993
12	10	610	462	999
3	15	432	406	2078
4	15	593	402	1531
6	15	863	411	1048
8	15	908	425	988
12	15	915	420	997
3	20	433	402	2766
4	20	583	401	2054
6	20	843	407	1402
8	20	1153	404	1030
12	20	1219	433	993

frequency, simulating 10 to 20 devices, and using 3 to 12 threads. The message contents were five randomly generated values in a JSON object (according to the Weather template). The duration of performing a test is around one minute.

The executed test results showed that the random data generalization consumed an almost negligible amount of time, so it did not interfere with the simulation. First, we started using three threads, because Samsung S5 has four cores, but from the results depicted in Table 2.6 we can see that the number of threads is an important factor. With 10 simulated devices, three threads had difficulties coping with simultaneous message sending, it required almost 1.5 seconds for the devices to start a message sending cycle again, instead of the one second specified in the settings (see Figure 2.12). The figure shows the number of threads used on the x axis, the number of sent messages with blue, and the time between cycles (in ms) with red color. When the red and blue lines become horizontal, the number of used threads is high enough to perform the required tasks. As we can see from the results, four threads are needed to solve the problem. If the number of simulated devices grow, it is obvious that more threads are needed to manage them. To manage 15 devices, four threads are few, but six threads would be enough to perform well. This fact can be seen in Figure 2.13, where the blue line represent the number of sent messages, and the red line denotes the time spent between the sending cycles. To manage 20 devices, we can see that six threads were too few for the acceptable performance, therefore, we had to use eight threads instead (as shown in Figure 2.14).

We also made additional experiments to find the limits of MobIoTSim using 20 simulated devices. The minimum time required to send a message is around half a second in general. For lower frequency (e.g. 0.25 second), there is no difference in the measured time even by using 16 threads as Figure 2.15 suggests. To conclude these experiments, we can state that our proposed simulator can send a total number of 2300 messages in a minute with 20 registered devices to the IBM Cloud.

2.10 Summary

In the phase of the latest Internet evolution, a high number of powerful devices appeared on the network, which led to the birth of the IoT paradigm. As a result, cloud technology providers responded to this trend, and broadened their services to attract users of IoT applications. In this new world, IoT developers are facing many challenges: they need to purchase smart devices to design and build-up a network of these components, to develop IoT applications, to test and evaluate these applications on the created system, and finally, to fine-tune the applications based on the evaluation results.

In this chapter I introduced a mobile IoT simulator called MobIoTSim. After creating a survey that gathers the relevant simulators and cloud solutions with IoT capabilities, I created the requirements for an IoT simulator. The simulator is capable of simulating multiple IoT devices in a user-friendly way. The data generation can be performed with artificially generated data, and it can also be based on real IoT sensor data, and it uses the real network layer to send the data to the cloud, therefore, the simulation becomes realistic. I examined the resource usage of the software to validate its capabilities. The results of this chapter belong to Thesis I, and were published in papers [P14], [P12], [P13], [P6], [P10], [P9], [P4], [P11], [P2] and [P5].

Part II

IoT Gateway

3

A Private Gateway for Investigating IoT Data Management

3.1 The need for private gateways in IoT systems

Internet of Things (IoT) systems follow a new trend, representing a dynamic global network infrastructure with self configuring capabilities [130], in which things can interact and communicate among themselves and with the environment through the Internet by exchanging sensor data, and can react autonomously to events and influence them by triggering actions with or without direct human intervention. Such systems can be utilized in many application areas, thus they may have very different properties. According to recent reports in the IoT field (e.g. [115]), there will be 30 billion devices always online and more than 200 billion devices discontinuously online by 2020. Such estimations call for smart solutions that provide means to interconnect and control these devices in an efficient way.

Cloud computing [95] enables flexible resource provisions that have become hugely popular for many businesses that want to respond quickly to customers' demands. Different cloud infrastructure solutions have been developed in the past decade to support the creation, deployment and management of private clouds. OpenStack [132] is the most popular and widespread solution for this, but other promising approaches have also appeared, such as CircleCloud [102] and OpenNebula [117]. There is also a growing number of cloud providers offering IoT-specific services, since cloud computing has the potential to serve IoT needs, such as hiding data generation, processing, and visualization tasks. With the help of these virtualized solutions, user data can be stored in a remote location and can be accessed from anywhere.

By realizing these capabilities, there are several, popular cloud platform providers available offering IoT specific services. Some of these IoT features are unique, but every IoT-Cloud provider has the basic capabilities to connect and store data from devices. Creating and examining such applications within IoT-Cloud environments represents a great challenge, since many things have to be managed at the same time, and a wide range of devices and data formats are available.

By addressing these needs, the main contributions of this chapter are the following: first, we characterize the problem area, then introduce how to develop real cloud gateway services to manage the simulated devices composing a complex semi-simulated environment. Finally, we evaluate the performance and scalability of the proposed solution.

The goal of our research with this tool is to support the proliferation of integrated IoT, mobile, and cloud technologies, and to enable experimenting with complex systems fulfilling the efficient management of user applications. By using our proposed tools, real things or devices can be substituted by mimicking their behavior, thus there is no need to perform initial investments by buying real sensors or devices to investigate IoT applications.

The remainder of this chapter is presented as follows: Section 3.2 discusses emerging technological improvements clarifying the research scope and introduces an overview of related works. Section 3.3 presents our proposed IoT-Cloud simulation environment, and also discusses the development possibilities of private cloud gateways for managing IoT devices, before it evaluates its scalability and device management features. Section 3.4 introduces a platform-independent gateway service, which can be deployed to any cloud later. Finally, the contributions are summarized in Section 3.5.

3.2 Related work in IoT Gateway solutions

Cloud computing has become a widespread and reliable solution over the past decade. Overcoming interoperability issues of public cloud providers and various middleware implementations, the process of creating and managing cloud federations is clarified and applied [112]. There are various reasons to optimize resource management in such federations: to serve more users simultaneously, to increase quality of service, to gain higher profit from resource renting, or to reduce energy consumption or CO_2 emissions.

Figure 3.1. Evolution of cloud systems

As depicted in Figure 3.1, in the past decade we experienced an evolution in Cloud Computing: the first clouds appeared in the form of a single virtualized datacenter, then broadened into a larger system of multiple interconnected datacenters. As the next step cloud bursting techniques were developed to share the resources of different clouds, then cloud federations were realized by interoperating formerly separate cloud systems. Once overall optimization issues in cloud federations addressing datacenter consolidation, operating costs, and energy efficiency were developed, further research directions started to use clouds to support newly emerging domains, such as the Internet of Things. In the case of IoT systems, data management operations were better placed close to their origins, thus close to the users, which resulted in better exploiting the edge devices of the network. Finally, as the latest step of this evolution, the group

of such edge nodes formed the fog. Dastjerdi and Buyya defined fog computing as a distributed paradigm [103], where cloud storage and computational services are performed at the network edge. This new paradigm enables the execution of data processing and analytics application in a distributed way, possibly utilizing both cloud and nearby resources. The main goal is to achieve low latency. However fog computing also brings novel challenges in real-time analytics, stream processing, power consumption and security.

There are many simulators available to examine distributed and specifically cloud systems. Nevertheless, there are some more specific IoT simulators addressing similar issues. Han et al. [107] have designed DPWSim, which is a simulation toolkit to support the development of service-oriented and event-driven IoT applications with secure web service capabilities. SimIoT [129] was derived from the SimIC simulation framework. It introduces several techniques to simulate the communication between an IoT sensor and the cloud, but it is limited to computing activity models. The SimpleIoTSimulator [126] is an IoT sensor or device simulator that is able to create test environments made up of thousands of sensors and gateways on a computer. It supports many of the common IoT protocols (e.g. CoAP, MQTT, HTTP). Its drawback is that it needs a specific, RedHat Linux environment, while our approach is more heterogeneous, and focuses on IoT device simulation with mobile devices, which is easier to apply. The Atomiton simulator [58] is probably the closest solution to our concept. It manages virtual sensors, actuators, and devices with unique behaviors. With this tool complex, dynamic systems can be created for specific applications. Unlike our open, mobile solution, it is a commercial, web-based environment with very limited documentation.

Kang et al. introduced the main types and features of IoT gateways in [110]. This work is a detailed study in this regard, and presents the state-of-the-art and research directions in this field. Our research does not aim to propose a generic solution for all needs of an IoT system, but to provide a gateway solution that can be used together with MobIoTSim to enable a comprehensive simulation environment for investigating IoT-Clouds.

3.3 Developing a private gateway

3.3.1 A comprehensive IoT-Cloud simulation environment

We have already introduced the architecture of our vision for simulating IoT-Cloud systems in the previous chapter (see Figure 2.6). Unlike traditional simulators, we try to stay as close to real world systems as possible. We take trace files saved from public applications, place them into an archive (I.), and use them in MobIoTSim (II.) to mimic real device behavior. We also develop gateway services (III.) to process and visualize sensor data and instantiate and operate them at private or public cloud providers.

The basic usage of the simulator is to: (i) connect the application to a cloud, where the data is to be sent, (ii) create and configure the devices to be simulated, and (iii) start the (data generation of the) required devices. These main steps are represented by the three main parts of the application: the *Cloud settings*, the *Devices* and the *Device settings* screens.

We also developed our own, private gateway service (available in [48]) for the IBM Cloud (i.e. Bluemix till 2017) platform that is able to manage several devices simultaneously, and can send a notification to the MobIoTSim device simulator by responding

to critical sensor values. This gateway service is basically an extended version of the IBM visualization application [38]. It has a web-based graphical interface to visualize sensor data coming from MobIoTSim. Messages (defined in JSON format) received from the simulated devices are managed by an MQTT server. It can also be used to send responses (or notifications) back to the simulated IoT devices in MobIoTSim.

It is easy to connect the simulator to such a private gateway. Since it has a predefined template called *Bluemix*, we only need to specify an organization identifier and the connection type (either TCP or TLS) to enable connection to the MQTT server. In the case of already available templates, the necessary URL is predefined. The application ID, the authentication key and token can be retrieved by registering to the gateway service, and these parameters can be used later to sign in to the data visualization site of the gateway. The simulated devices also need to be registered to the MQTT server of the gateway service – just like real devices –, by specifying their device and type identifiers and sensor data thresholds, which replies with their token identifiers (to be used for device setting). Once these settings are made, simulated devices can be defined and started in the same way as shown previously for the demo gateway. We can create advanced scenarios with this private gateway, such as managing more devices and responding to critical sensor data coming from the simulated devices.

Figure 3.2. Detailed, parameter-wise data visualization in the private gateway in the IBM Cloud sent by a group of devices in MobIoTSim

3.3.2 Evaluation of the proposed private gateway

As a next step, we further revised our private gateway for Bluemix to handle up to hundreds of devices at the same time. This way we can see the data sent by all devices in a single, real-time chart as shown in Figure 3.2. The node.js application managing the chart subscribes to the device topics with the MQTT protocol and waits for the messages. In this extended gateway, we use paging to overcome device management

limitations (25 devices at a time). In order to enhance and better visualize many devices data at the same time, we introduced device grouping for the chart generation.

Though the Bluemix platform provides some monitoring information for application services, custom Docker containers can be better monitored. For this purpose, we created the Docker version of our private gateway, which even became more portable this way.

We also examined and evaluated this gateway service. We deployed the private gateway in a Docker container as a micro application with the following parameters: OS - Ubuntu with Linux kernel 4.4.0, RAM - 256 MBs, CPU - Intel Xeon E5-2690 with 48 cores.

Figure 3.3. Initial evaluation with Thermostat devices

Figure 3.4. Evaluation results with CPU stressing

First, as an initial evaluation, we used the Thermostat template of MobIoTsim to create device groups of 100 and 450 simulated devices. We performed the simulations for 10 minutes and generated sensor messages in every second. We used the Grafana tool in Bluemix to monitor the resource consumption concerning CPU, memory and network usage. The results with median values can be seen in Figure 3.3. For the third case in the diagram, we reset the message generation interval to 0.5 seconds, therefore, we got 900 (denoted by 450×2) messages in every second.

Figure 3.5. Comparison of different device groups

Figure 3.6. Comparison of different data generation intervals

As we can see from the diagram we only experienced light load in all cases. The reason for this is that the gateway application only received the messages and forwarded them to a javascript-based chart generator run in a browser at the client side. In this case we performed no real data processing tasks, though in real world we may do that.

As a result, we extended our private gateway by introducing stressing processes to support more realistic IoT operation. For CPU stressing, we implemented a parameterizable Fibonacci number generation algorithm. We used the setting to count the 20th Fibonacci number upon each received message multiplied by a score representing the size of the message (as we supposed that larger messages needed more processing computation).

For the next experiment, we enabled stressing, and created 10, 100, and 250 thermostat devices in groups. The averages of the measured results can be seen in Figure 3.4. Now we can see that the load of the CPU is increasing as we manage a higher number of devices at the same time. Therefore, we extended this measurement and compared 10, 100, 250, and 450 devices in groups. Figure 3.5 shows the average results for the device groups and the appropriate resources. Next, we re-executed this scenario by reducing the data generation frequency to 0.5 seconds. The comparison of continuous measurements for 100, 250, and 450 devices of the 1 and the 0.5 seconds message generation can be seen in Figure 3.6.

Figure 3.7. Comparison of message data sizes

Table 3.1. Results of measurements with the Weather template.

No. of devices	10	100	250	450
CPU util. (%)	1.59	12.27	29.53	52.29
Memory (MB)	110.07	110.22	110.35	110.05
Network (B/s)	890.66	881.34	855.16	853.60
Message size (KB)	2468	24695	61666	111110
No. of messages	6000	60046	149940	270165

Once we got some impression of working with a simple thermostat device, we chose a more complex one. As one of the earliest examples of sensor networks are from

Figure 3.8. Comparison of CPU utilization of the two device templates

the field of meteorology and weather prediction, we chose to model a meteorological service and gathered real data from OpenWeatherMap [46]. This service provides a global geospatial platform supporting the development and operation of data-driven products for agriculture, transportation, and alike. It monitors and publishes actual weather conditions and forecasts for more than 200 thousand cities using data from more than 40 thousand weather stations. Their database includes historical data, which is accessible through APIs.

As a final evaluation step, we used the Weather template of MobIoTSim to create device groups of 100 and 450 simulated devices. By using this template, we can set up weather station parameters following the OpenWeatherMap format, or load previously saved OpenWeatherMap traces of certain cities. For this experiment, we randomly picked weather data of cities (one city for one simulated device) from earlier traces. Figure 3.7 shows the difference between the two applied templates in sizes, while Table 3.1 provide the detailed median measurement results. We can see that as the number of devices grows, the resource utilization also gets higher. Figure 3.8 compares the CPU utilization of the two device types.

3.4 Platform-independent gateway

We also created a platform-independent gateway service, which can be later deployed to any cloud. Such a service contains an MQTT broker to help the communication between the IoT environment and the cloud application as shown in Figure 3.9. A graphical interface for a thing database is required to register IoT devices and generate API keys for cloud applications. The device registration makes the communication more secure.

We have already evaluated existing, opensource MQTT brokers, which is the base

Figure 3.9. Platform-independent gateway architecture

Table 3.2. Comparison of MQTT brokers

	Verne-MQ	Moquette	Rabbit-MQ	Active-MQ	Joram-MQ	Mosquitto
Price	free	free	free	free	free	free
Opensource	yes	yes	yes	yes	yes	yes
License	Ap. 2.0	Ap. 2.0	Moz. Pub.	Ap. 2.0	LGPL	Ap. 2.0
Custer	yes	no	yes	yes	yes	no
Linux	yes	yes	yes	yes	yes	yes
MacOS	yes	yes	yes	yes	na	na
Windows	no	yes	yes	yes	yes	yes
Language	Erlang	Java	Erlang	Java	Java	C

of a platform-independent gateway service. Our criteria for an appropriate MQTT broker was to be opensource, scalable, and extendable for dynamic device registration and management. In Table 3.2, we provide an overview of the most popular MQTT brokers we started to investigate. These brokers are the following: VerneMQ [57] [56], Moquette [43] [42], RabbitMQ [52] [51], ActiveMQ [25] [24], JoramMQ [40] [39], Mosquitto [45] [44].

3.5 Summary of contributions related to the generic IoT Gateway

The IoT paradigm has appeared as the latest Internet revolution, generating a huge amount of powerful devices for the online world. By responding to this trend, several cloud providers have started to offer specific management services for this new world. In this currently forming IoT ecosystem, users and system developers not only have to buy and configure devices, but they also have to choose the right IoT-Cloud provider, offering the combination of protocols and data structures fitting their applications.

To support users and researchers in this field, I have shown how to develop a generic cloud gateway service to manage simulated devices composing a complex semi-simulated environment. It can be deployed on any cloud service, and can receive, store

and process/visualize data. I have also evaluated the performance and scalability of this IoT gateway service. Using these solutions, researchers and developers can investigate the behavior of IoT systems, and develop and evaluate IoT-Cloud applications more efficiently in a convenient, cost effective way. The results of this chapter belong to Thesis II, and were published in papers [P14], [P12], [P13], [P6], [P10], [P9], [P4] and [P2].

4

Designing an IoT-Cloud Gateway for the Internet of Living Things

4.1 The IoLT project and the need for a specialized gateway

According to recent reports in the field of the Internet of Things (IoT) (e.g. [109]), there will be 25 billion connected things by 2021. These estimations call for smart solutions that provide means to connect, manage, and control these devices efficiently. IoT can be envisioned as a dynamic network with self-configuring capabilities, in which devices (that are called things) can interact and communicate among themselves and with the environment by exchanging sensor data. Such systems can be utilized in many application areas, thus they may have very different properties.

Smart farming is also a rapidly growing area within smart systems that need to respond to great challenges in the near future. By 2050, it is expected that the global population will grow to 9.6 billion as the United Nations Food and Agriculture Organisation predicts. A recent Beecham Research report [28] also states that food production has to respond to this growth by increasing it with 70% till 2050. This report also states that agriculture is responsible for a fifth of greenhouse gas emissions and 70 percent of the world's fresh water usage, which strives for a reform. IoT supported by cloud services has the potential to implement the required changes [92].

Plant phenotyping [123] also evolves rapidly and provides high throughput approaches for monitoring the growth, physiological parameters, and stress responses of plants with high spatial and temporal resolution. Recent advances use the combination of various remote sensing methods that can exploit IoT and cloud technologies. In the past, typical plant phenotyping platforms used very expensive instrumentation to monitor several hundreds, even few thousands of plants. Although these large infrastructures are very powerful, they have high cost, ever reaching to a staggering few million euros per platform, which limits their widespread, everyday use. Due to recent ICT developments we can apply novel sensor and IoT technologies to provide a

promising alternative, called affordable phenotyping. Our research goals also point to this direction, and we propose a low cost plant phenotyping platform for small sized plants, which enables the remote monitoring of plant growth in a standard greenhouse environment.

The main contributions of this chapter are the design and implementation of the IoLT Smart Pot Gateway for managing smart pot clusters by monitoring their environmental parameters. This IoT-Cloud platform is capable of collecting, storing, and visualizing sensor data, as well as performing leaf area calculations with image processing to allow plant growth tracking. We also evaluate the proposed solution with scalable simulations, and exemplify real world utilization.

The remainder of this chapter is as follows: Section 4.2 introduces related approaches for smart farming and plant monitoring, and Section 4.3 highlights our research aims and discusses the proposed smart pot solution. Section 4.4 presents a detailed evaluation of our gateway framework by means of simulation, Section 4.5 shows real world utilization. Finally, we conclude this part in Section 4.6.

4.2 IoT solutions for monitoring plants

Smart system design and development have started to flourish. Smart farming is also getting very popular; there are many commercial solutions and products in household areas.

Concerning indoor plants, many tools are available for monitoring temperature, humidity, light, water level, and salt content of the plant soil, and these tools are able to communicate with nearby devices. Some advanced systems are capable of automatic watering or provide notifications or even remote control through mobile applications. Table 4.1 shows a comparison of the available solutions, and we briefly introduce them in the following sections.

PlantRay [79] is a really basic smart pot; it has a soil moisture sensor, and it changes color and beeps if watering is required. The battery may last for a year. AeroGarden [62] has many different sized products, able to hold from two to 24 plants. They have LED lights, but only the bigger ones have automatic LED control. The most advanced one can be connected with Amazon Echo. Xiaomi Flora [76] has an indicator board for providing information coming from the sensors, and it has a dedicated application for remote controlling the management of the pot. It is able to monitor the moisture and salt content of the soil. GAIAA [67] is a solution from sPlant, which is capable of managing four plants at a time. It also has a remote app control, and provides automatic watering, light supplement, and WiFi communication with a cloud server. Tregren [87] produces 3 different sized products; they can handle 3, 6 or even 12 plants. The watering and the light control are automatic, it can usually be left alone for 21 days. The SmartPot [80] is just a prototype, but it has temperature, humidity, light, soil moisture sensors, and a water pump. It can be controlled by a mobile application. The Parrot pot [78] includes a self-watering system and four built-in sensors. Unfortunately, its production has been suspended. Click & Grow [65] has two similar products, they only differ in size. The smaller one is for three plants, the bigger one can handle 9 plants. CitySens [64] is a vertical pot system with auto-watering and variable pot numbers with an option to communicate with a mobile application via wifi. LeGrow [71] creates modules for a smart pot system. Currently, they sell lamp, humidifier,

power, and pot modules. Odyseed [75] is a smart pot solution that uses time schedules for automatic irrigation and lighting.

There are some other interesting products, like Lua [72], the flowerpot with 15 different animated emotions. The emotions representing the status of the plant are based on the moisture, light, and temperature sensors. The Vincross [88] company has an AI robot called HEXA, and it can move the plant to get enough sunshine. The TOKQI [86] smart pot can detect if the user pets the plant and starts playing music. It is a decorative item with RGB lighting and it can be used as a regular bluetooth speaker, too. For professional usage, there are only very few commercially available platforms for affordable phenotyping (e.g. PhenoBox [100]).

Table 4.1. Comparison of commercial smart pot solutions

Product	Main features	No. of plants	Price (EUR)
Xiaomi Flora	indicator board, salt content mon.	1	40
Parrot pot	self-watering system, 4 sensors	1	50
sPlant	light supplement	4	90
PlantRay	soil moisture, color change, beep	1	19
Tregren	control light, water and nutrients	3-6-12	90
Odyseed	automatic irrigation and lighting	2	70
Click & Grow	water plants automatically, LEDs	3-9	100-200
CitySens	self-watering, app	3-5	176
LeGrow	modular	1+	40
AeroGarden	LEDs, Amazon Echo	2-24	40-450
SmartPot	many sensor prototype	1	NA
Lua	15 different universal animated emotions	1	100
TOKQI	pet plant to play music, bluetooth speaker	1	12
HEXA	AI robot moving to sunshine	1	950

Concerning generic IoT gateways, Kang et al. [110] introduced the main types and features of IoT gateways in a detailed study, which presents the state-of-the-art and research directions in this field. This solution is also too generic for our needs.

Focusing on the development of a smart farming environment, Dagar et al. [101] proposed a model of a simple smart farming architecture of IoT sensors capable of collecting information on environmental data and sending them to a server using wireless connection. There are also generic solutions to monitor agricultural applications using IoT systems, such as the Kaa IoT Platform [70]. It is a commercial product that is able to perform sensor-based field and remote crop monitoring. It also has an open source version called the Kaa Community Edition. Such generic toolkits are quite complex and heavy-weight, so they are not well suited to specific needs.

In contrast to these solutions, our approach aims to provide a low-cost solution using the latest IoT and Cloud techniques to enable a robust and scalable solution to be used for groups of plants with user friendly management.

Figure 4.1. IoLT Smart Pot prototype (left), and one pot of the cluster (right)

4.3 The design of a Smart Pot for the IoLT project

The Internet of Living Things (IoLT) project was started in 2017 with the aims to integrate IoT technological research with applied biological research, and to develop IoT applications for three target fields: complex plant phenotyping, actigraphy for psychosocial treatments, and Lab-on-a-chip systems for microfluidic diagnostics. IoLT is also forming a Network of Excellence of researchers of corresponding disciplines working at the University of Szeged and the Biological Research Centre of the Hungarian Academy of Sciences. An opensource IoLT platform is under development to enable the execution of applications on cheap, low capacity IoT devices, providing easy to use programming interfaces based on Javascript.

In the research field of plant phenotyping, we planned to design and develop a scalable, low-cost automation system, called IoLT Smart Pot, using IoT and cloud technologies, to monitor the effect of various stress factors of plants (drought, nutrition, salt, heavy metals, etc.), as well as the behavior of various mutant lines. For the first prototype depicted in Figure 4.1, the biologists designed a hardware for hosting 12 small sized plant pots (for *Arabidopsis* plants) organized in a 4x3 matrix. To monitor plant growth, an RGB camera and a LED-based illumination system for additional lighting

Figure 4.2. The architecture of the IoLT Smart Pot Gateway

are installed above the plant cluster. The relevant environmental parameters are light intensity, air and leaf temperature, relative air and soil humidity, which are monitored by sensors placed above and into the pots. To govern the monitoring processes, a Raspberry Pi board is placed beside the cluster. The monitored sensor data is stored locally on the board, and is accessible through a wired connection on the same network. The initial configuration for performing periodical monitoring was set to 5 minutes concerning the sensor readings, and 1 hour to take pictures of the cluster of pots.

4.3.1 Implementation of the IoLT Smart Pot Gateway

The architecture of our initially proposed IoLT Smart Pot Gateway can be seen in Figure 4.2. It has a modular setup, it consists of three microservices, and its source code can be found on GitHub [69]. The microservices are realized by Docker containers [66], which are composed of the gateway application that can be deployed to a virtual machine (VM) of a cloud provider. Special monitoring scripts are used to track and log the resource utilization of the containers for performance measurements. The users can access the gateway through a web interface provided by a Node.js portal application. It can be used to group and manage pots and users with projects created by administrators. Projects need to have start and finish dates, associated users and a short description. Pots can also be registered by them and linked to projects. In this way, registered and connected smart pots can send sensor data to them, which can be visualized in the portal. A sample view of such a portal web interface can be seen in Figure 4.3. It displays a project named *Real BRC Smartpot test (1 week)* managing a smart pot registered as *BRC_Smartpot_1*. The chart depicts values (y axis) of seven

Figure 4.3. Historical sensor data visualization in the IoLT Smart Pot Gateway

sensor types with timestamps (X axis) for a week of utilization. On the chart interface, a user can tick or untick certain sensors, and change the time interval below the chart using a sliding bar. Once a setup is done, the depicted sensor datasets can be downloaded (in CSV format) by clicking on the "Download" button.

The Node.js portal application is built upon two other microservices. In the middle of the architecture in Figure 4.2, we can find the Mosquitto MQTT Broker service, which is built on the open-source Mosquitto tool [45] to store the received sensor values of the pots using a MongoDB [73] database. The monitored seven sensor types of a pot are described by a JSON document (see later in Figure 4.9), which should be regularly updated and sent in a message by an MQTT client of a smart pot to the MQTT broker running in this service. The sensor readings on the Raspberry Pi board are performed by a python script using an MQTT client package configured with a pot identifier, sensor value sampling frequencies, and picture taking frequencies. The third microservice on the bottom is called the Apache Web Server, which is responsible for saving the pictures of the plants of the pots. The python scripts of the boards use SFTP file transfers to send the pictures stored by this service.

4.3.2 A solution for monitoring and analyzing plant growth over time

After the initial version of the gateway portal was released, the biologists started to use it for monitoring Arabidopsis plants. As mentioned in the previous section, the gateway stores regularly updated sensor values and periodically taken pictures of the smart pot cluster. The portal can be used to query, visualize, and download a set of sensor values for a certain period, and the created pictures.

Besides viewing these monitored results, the biologists had to perform post-processing tasks on the monitored data after downloading them from the gateway portal. One

Figure 4.4. Real and segmented pictures of the Smart Pot cluster taken at 2019.01.01

```
{
  Time:"2019-06-01 15:46:30",Slot_1:8.37,Slot_2:11.76,Slot_3:9.57,
  Slot_4:11.54,Slot_5:10.73,Slot_6:16.39,Slot_7:23.87,Slot_8:22.39,
  Slot_9:1.88,Slot_10:21.33,Slot_11:20.02,Slot_12:20.69
}
```

Figure 4.5. Projected leaf area values to be stored in the database of the gateway

Sensor data query

You can request measured data through this page.
It currently saves every data that has been measured on the project.

Name of the project:

From: Until:

Figure 4.6. The gateway screen for querying detailed leaf area values

of these tasks is to calculate the growth speed of the plants, which is generally performed by calculating the projected leaf area visible on the taken pictures, and filing them to a time series document, later depicting them in diagrams. At first, such a task had to be done manually, taking valuable time from away the researchers.

Responding to their needs, we extended the gateway with a new functionality. After a new picture is uploaded to the gateway, a python script is triggered to perform the segmentation of the picture. The segmented pictures are also stored in the gateway server in a subfolder to allow verification from the researchers. After segmentation, the projected leaf area is calculated for all 12 plants visible on the segmented picture (also with image-processing algorithms in a python script), then saved to the database of the gateway in the format shown in Figure 4.5 (in cm^2).

Figure 4.7. Selection of a pot from the cluster

Figure 4.8. Detailed leaf area values over time for a pot

In order to access the results, a researcher should log in to the portal web interface of the gateway, and select a registered project with a time interval, as shown in Figure 4.6. For the next step, one can select one of the plants (represented with a slot id) of the smart pot cluster associated with the given project (as depicted by Figure 4.7). Finally, as Figure 4.8 shows, we can see the chart of the calculated values that represent the time course of the projected leaf area of the selected plant in a pot of the cluster. The curve nicely reveals a circadian oscillation pattern due to periodic leaf movement (flattening in the dark and erection in the light period).

```
{
  "Project": "SampleProject",
  "Soil-sensor II": "434.437",
  "Full light intensity [lux]": 16901.38,
  "Time": "2019-01-14 14:02:56",
  "Humidity [%]": "41.1",
  "Soil-sensor I": "594.940",
  "IR light intensity [lux]": 15865.80,
  "Temperature [C]": "21.8",
  "Visible light intensity [lux]": 1035.58
}
```

Figure 4.9. Sample JSON message of seven sensor values of a pot

Figure 4.10. CPU and memory usage measurement results for 250 pots

4.4 Evaluation of the Smart Pot Gateway

In order to evaluate our proposed solution, we instantiated an IoLT Smart Pot Gateway service in the MTA Cloud [74] with a small VM flavor with a single virtual CPU core and two GBs of memory. The MTA Cloud is an OpenStack-based, national community cloud financed by the Hungarian Academy of Sciences that provides cloud infrastructure services for scientists from the academy.

4.4.1 Simulations with a python tool

First, we performed a throughout evaluation by means of simulation. After executing some initial measurements, we found out the exact, real data value ranges for the installed sensors of the smart pot. Based on these values, we designed a simulated smart pot represented by python scripts capable of sending generated sensor data via the MQTT protocol. Figure 4.9 depicts a generated sample JSON file for the revealed sensor types.

First, we created 250 simulated pots with scripts that sent generated sensor data

Figure 4.11. Network and I/O usage measurement results for 250 pots

Figure 4.12. CPU and memory usage measurement results for 50 pot clusters

to our IoLT Smart Pot Gateway service (running at MTA Cloud) for 30 minutes. We divided the experiment' total time-frame to the following periods:

- in the first 10 minutes, we applied a sensor data generation frequency of 30 seconds (which means that each pot sent a message of 7 sensor values every 30 seconds);
- in the second 10 minutes, we applied a sensor data generation frequency of 10 seconds;
- in the following 5 minutes, we applied a sensor data generation frequency of 2 seconds;
- and in the last 5 minutes, we applied a sensor data generation frequency of 10 seconds, again.

We also developed a special monitoring script for the gateway (as shown in Figure 4.2) to track its resource consumption. The resource usage sampling of the script

Figure 4.13. Network and I/O usage measurement results for 50 pot clusters

Figure 4.14. CPU and memory usage measurement results for 100 pot clusters

was set to 10 seconds. They queried CPU, memory, network and input/output (I/O) resource utilization for all containers, and we added up these values to get the total resource consumption of the composed service (running in a VM). We can see the measurement results for this initial round simulating 250 pots in Figure 4.10 and Figure 4.11. The x axis denotes the timestamps of resource usage monitoring, while the y axis denotes the resource usage values (in percentage or in kB or MB). We can see that there are some spikes in the resource usage percentages after the first 10 minutes, when we start to send more messages, and from the 20th minute the utilization has an increasing trend. Nevertheless, we have to mention that the resource usage sampling is less frequent than the arrival rate of the messages, which results in an incomplete curve (the resource utilization is not tracked between the sampling intervals). The network and I/O utilization were visible only for the first 10 minutes, possibly due to the initialization phase of the script.

Next, we set the simulation parameters in a way to mimic future, real world uti-

Figure 4.15. Network and I/O usage measurement results for 100 pot clusters

Figure 4.16. CPU and memory usage measurement results for 250 pot clusters

lization. Our proposed IoLT Smart Pot is basically a cluster of 12 pots, as shown in Figure 4.1. To evaluate the scalability of our gateway solution, we performed three simulation measurements with 50, 100, and 250 clusters (composed of 600, 1200, and 3000 pots respectively). In all cases, we performed the measurements for 30 minutes, and the simulated smart pot platform sent sensor values with the following setup:

- in the first 10 minutes, we applied a sensor data generation frequency of 5 minutes (which means that each pot sent a message of 7 sensor values every 5 minutes: resulting 2 messages in this period per pot);
- in the second 10 minutes, we applied a sensor data generation frequency of 1 minute;
- and in the last 10 minutes, we applied a sensor data generation frequency of 5 minutes, again.

Figure 4.17. Network and I/O usage measurement results for 250 pot clusters

Table 4.2. Comparison of the four evaluation rounds

No. of pots		250	600	1200	3000
CPU	AVG	6.22	1.19	7.06	8.65
CPU	MAX	26.64	13.42	30.91	39.29
MEM	AVG	14.28	17.79	15.55	21.72
MEM	MAX	16.14	18.94	15.95	24.73

In the first simulation for 50 clusters, we set the sampling of resource usage (processor and memory usage) to every 10 seconds, while for the second and the third one (100 and 250 clusters) we set it to two seconds (to have a better resolution of resource loads).

We can see the measurement results for the first round simulating 50 clusters with 600 pots in Figure 4.12 and Figure 4.13 for 30 minutes. Here we can see that the average CPU load varies between one and two percent, and the memory usage fluctuates between 15 and 19 percent. The network and I/O utilization are slowly, but constantly growing. In this experiment we also observed that the time of an actual data processing (receiving a message and writing its contents to the database) and the time of the resource usage sampling are rarely matched. One matching example can be seen right after the third minute in Figure 4.12, which shows a spike with almost 14 percent of CPU utilization.

For the second round, we doubled the number of clusters to 100, and performed the simulation for only five minutes with detailed resource usage sampling of two seconds. We can see the measurement results for this round simulating 100 clusters with 1200 pots. In Figure 4.14, we can see that the results reveal a periodic resource usage fluctuation denoting the data processing activities. For the network and I/O utilization shown in Figure 4.15 we can still see a constant grow.

Finally, for the largest experiment, we further increased the number of pot clusters to 250 arriving to a total number of 3000 simulated pots. For this third round, we performed the simulation for 30 minutes, again, with the same periods as defined for the first round (of 50 clusters). We can see the measurement results in Figure 4.16 and Figure 4.17. If we take a look at the middle 10 minute period, we can see the periodic

Table 4.3. Comparison of the 30 min. evaluation rounds

No. of pots			600	3000
CPU	AVG	0:00 - 0:10	1.37	6.65
		0:10 - 0:20	1.09	14.11
		0:20 - 0:30	1.11	5.01
CPU	MAX	0:00 - 0:10	13.42	39.29
		0:10 - 0:20	1.25	36.43
		0:20 - 0:30	1.27	31.78
MEM	AVG	0:00 - 0:10	17.44	22.67
		0:10 - 0:20	17.88	22.60
		0:20 - 0:30	18.05	19.81
MEM	MAX	0:00 - 0:10	18.94	24.47
		0:10 - 0:20	18.00	24.73
		0:20 - 0:30	18.11	20.74

resource usage spikes for CPU and memory, as in the previous round. And we can also observe the utilization growth in network and I/O data transfers.

Figure 4.18. Data visualization of a real world measurement in the IoLT Smart Pot Gateway

To summarize our investigations, Table 6.3 compares the average and maximum resource utilization values measured during the experiments. We can see that by increasing the number of pots to be managed by the gateway service, the utilization also increases. As expected, the CPU utilization was the highest in the third round for managing 3000 pots at the same time with almost 40 percent. The memory utilization is also the highest in this case with almost 25 percent. Table 4.3 shows a detailed comparison for the longest experiments denoting the different phases of the measurements. This table highlights that the CPU utilization generally reaches its maximum in the first phase, then it generally drops, while memory utilization shows a quite balanced

load all over the three phases. Finally, we can state that these results prove that we can easily serve numerous phenotyping projects monitoring up to thousands of pots with a single gateway instance in a Cloud.

Figure 4.19. Sensor data of a day in the IoLT Smart Pot Gateway

4.5 Real world measurements

We have seen in the previous section that our gateway service is scalable enough to manage a few thousand pots in a single cloud VM. To exemplify real world utilization, we connected the gateway to the IoLT Smart Pot prototype. It is able to hold 12 Arabidopsis plants in small pots organized into a cluster (as shown in Figure 4.1). We configured the python scripts of a Raspberry Pi board placed beside the pot cluster to perform sensor readings periodically, and send the environmental values to the IoT-Cloud gateway service. The wiring of the smart pot cluster formed a single IoT device with a camera and 7 sensors (attached to some of the 12 pots).

We performed the monitoring of the growth of Arabidopsis plants under standard greenhouse conditions for several periods, taking up around 2-3 months in total. RGB image taking was performed every hour, and the sensor sampling frequency was set to five minutes (to generate a JSON message). Figure 4.18 depicts a query at the gateway portal resulting in a chart of the sensor and leaf area values for over a month of monitoring. As we can see from the chart, the smart pot was disconnected for a certain period after around 20 days). If we zoom in by using the bar below the chart, we can view the detailed results. Figure 4.19 shows the sensor values of a day of utilization of the smart pot cluster.

4.6 Summary of contributions related to the IoLT gateway

Smart farming approaches are meant to revolutionize agriculture to improve productivity and reduce waste by exploiting the latest ICT technologies and trends. Affordable

phenotyping has the goal to provide low cost and easily scalable solutions to create the greenhouses of the future.

In this chapter, I aimed to contribute to this field by proposing the IoLT Smart Pot Platform and Gateway that can be used to manage smart pot clusters by monitoring environmental parameters. This solution is capable of collecting, storing and visualizing sensor data, as well as performing leaf area calculations with image processing to allow plant growth analysis. I also evaluated the proposed solution with scalable simulations, and exemplified real world utilization in standard greenhouse conditions. The results of this chapter belong to Thesis II, and were published in papers [P9], [P11], [P2] and [P5].

In our future work, we plan to redesign the smart pot and revise its corresponding gateway service with a solar cell system to enable portable installations and remote monitoring at outdoor locations.

Part III

SUMMON

5

Analyzing IoT-Cloud Environments Using Real Sensor Data

5.1 The need for open data reuse of smart systems

Internet of Things (IoT) solutions represent a new trend for forming a dynamic global network infrastructure with self configuring capabilities where, things can interact and communicate among themselves and with the environment through the Internet by exchanging sensor data, and react to events autonomously and they can be triggered with or without direct human interaction. We have already referred recent estimations that call for smart solutions that provide means to interconnect and control these devices in an efficient way.

Cloud computing [95] has the potential to serve these IoT needs since it enables flexible resource provisions to quickly respond to dynamic conditions, and it is able to store and process sensor data in a remote location accessed from anywhere. In the past decade, we experienced an evolution in Cloud Computing: they formed cloud federations to interoperating formerly separate cloud systems. Later, with Fog Computing service latency could be reduced and the service quality improved by keeping and processing the data close to the user. To analyze these complex systems we need tools to simulate them, and open datasets of real world IoT applications to enable realistic simulations.

The European Commission has published a guideline document [99] for encouraging open access to reuse digital research data generated by Horizon 2020 projects. In this report they defined the so-called FAIR data principles to help Horizon 2020 beneficiaries to make their research data findable, accessible, interoperable and reusable. Following these guidelines, we aimed to create an open IoT trace archive to facilitate data search and accessibility, and to reuse sensor data for experimenting with IoT-Cloud systems.

In this chapter, we (i) analyze sensor data formats and datasets of three smart city projects, (ii) propose a data retrieval service for gathering and filtering open datasets, and (iii) discuss how IoT data could be made available by this tool for simulating IoT-Cloud systems, and we also (iv) exemplify its utilization in an open source IoT device

simulator. The remainder of this chapter is as follows: Section 5.2 discusses related approaches in this field, while Section 5.3 analyzes the available open datasets of three smart city initiatives. Section 5.4 introduces our proposed data gathering tool, and exemplifies its utilization with MobIoTSim.

5.2 Simulations supported by real-world data

In IoT-Cloud environments, data is generated by certain sensors or things, and collected, stored, and processed by local or remote gateway services depending on the application of Fog or Cloud services, respectively. The design and application of such gateway solutions is not trivial. Kang et al. introduced the main types and features of such IoT gateways in [110], representing the state-of-the-art and research directions in this field. Besides real-world solutions, in many cases we need to use simulations to investigate the operation of such gateways, and to understand the inner workings of complex IoT-Cloud systems. There are many simulators available to examine distributed and specifically Cloud and IoT systems.

Moving closer to real-world applications, we may choose to simulate a part of the ecosystem, e.g. the devices that produce the data to be processed. For such purpose, we may use the SimpleIoTSimulator [126], which is an IoT sensor or device simulator that is able to create test environments made up of thousands of sensors and gateways on a computer, supporting many of the common IoT protocols. Its drawback is that it needs a specific, RedHat Linux environment. The Atomiton simulator [58] manages virtual sensors, actuators, and devices with unique behaviors. With this tool, complex, dynamic systems can be created for specific applications, but it is a commercial, web-based environment with very limited documentation. A more advanced and generic solution is MobIoTSim that can be used to simulate any device, and it can be connected to any cloud. It has predefined device templates, and a built in connection the IBM Cloud. When we use it for simulating IoT systems, working with real data could result in more convincing and more reliable applications. In research works addressing distributed, Grid and Cloud systems, two main sources are used for this purpose: the Grid Workloads Archive [108] and the Parallel Workloads Archive [104]. These archives contain trace files of real-world utilization. Our goal is to show our first attempt at creating a similar archive for the IoT research community.

5.3 Analyzing Sensor Data Formats and Contents of Smart Cities

As the first step of this research, we searched the Internet for publicly available projects of various IoT application areas. We found three smart city initiatives that met our criteria, and publish open data of IoT sensors and devices. In this section we analyze these projects in detail.

5.3.1 Sensor Data in the SmartME project

The SmartME project [93] was initiated in 2015 by a group of researchers in the Mobile and Distributed Systems Lab at the University of Messina. Their goal was to set up

a crowdfunded system using the IoT paradigm in the the municipality area of the city of Messina, Italy. During the planning phase of the system, they found cloud services very useful to be integrated with the IoT components to enable a virtual infrastructure management framework. They used OpenStack to implement the required cloud services. They named this approach a Software Defined City, where techniques of computer science, networking, data storage and sensing can work and interact together. This solution also serves as a reference architecture for building up smart cities, which represents a unified environment of static sensor network components and dynamically connecting mobile devices and objects, where contextualization is an important enabler of geo-awareness.

The project has an illustrative website [55], where the installed sensors and devices can be overviewed in a map. The IoT system is composed of low-cost microcontroller boards equipped with sensors and actuators and installed on buses, traffic lamps and buildings over the municipality area of Messina. This sensor network can be used to collect data and information of the urban area to design advanced services for citizens. In the SmartME project they use a collection of sensors to monitor environmental properties, traffic conditions and energy utilization.

Some of the Smart City data is published in an Open Data platform [54] with the use of an open source data portal software called CKAN [31], to store and visualize the collected data in a standard open data format. It has a layered structure to categorize and store data, which are accessible in different formats through an API. To collect the sampling data from the installed sensors and devices and send them to the CKAN datastore, they developed specific plugins running asynchronously on the IoT devices. The plugins periodically check the voltage levels of the connected pins of the sensors, through which they obtain the environmental data to be sent to the datastore, together with a timestamp and the geographical position of the device, which are Arduino YUN boards. It is an embedded board based on Atmel and Atheros micro-controllers running Linino OS, a Linux OpenWrt distribution based on OpenWrt. The boards have built-in Ethernet and WiFi support, as well as micro USB connections. To host a set of low-cost sensors, a Tinkerkit Shield is used attached to a YUN board. They use a Tinkerkit thermistore, Ldr and Mpl3115 sensors to monitor temperature, brightness and pressure properties. A Honeywell HIH-4030 sensor is used to monitor humidity, and an Arduino KY38 to track the environmental noise level. Finally, a Groove MQ9 sensor is planned to be installed for monitoring air quality, but such data is not available by the time of writing.

CKAN provides a quite flexible way for collecting, storing and accessing data, but after analyzing the SmartME repository we found it hard to compose a group of sensor data to be used for a specific simulation. This strengthened our research hypothesis that there is need for a mediator service, which is able to gather and filter data automatically, tailored for a specific requirement.

Now, taking a closer look at the SmartME website [55], we can see that there are 21 sensor groups (or nodes) within the Messina region (by the time of writing). Each group collect sensor data at a given location about the following environmental properties: temperature, brightness, humidity, pressure and noise. Table 5.1 summarizes these properties and the sensor and device types used to monitor them. After analyzing the open datasets we can state that property values are sampled every 10 minutes, which are varying based on the device location and the actual weather conditions. They were monitored between 2016 and 2017. In the following we present the

statistical result of our investigation, in which we analyzed 5 datasets for each sensor type.

Table 5.1. Sensor types in the Smart City of Messina

Property	Model	Unit	Device
Temperature	Thermistor	°C	TinkerKit
Brightness	LDR	Lux	TinkerKit
Humidity	HIH-4030	%	Honeywell
Pressure	mpl3115	hPa	TinkerKit
Noise	KY-038	Amp	Keyes

To measure temperature values in the SmartME project, Tinkerkit Thermistore sensors are used. The voltage level of the module varies between 0 and 5 V, depending on the outdoor temperature. The sensed value range is between 0 and 1023, these are converted to °C. Table 5.2 shows statistical summaries for the examined 5 datasets (denoted by a prefix of their original identifier).

Table 5.2. Temperature measurements in the Smart City of Messina

Database ID	Interval	Freq. (sec)	No. of samples	Values (in °C)
2f4bbebc-...	2017.05-06.	605	117	28 - 32
cf5c179a-...	2016.04 - 2017.09	605	6548	6 - 29
6f88f1eb-...	2017.05-09	605	10458	18 - 54
606903a8-...	2016.03 - 2017.09	605	46120	3 - 41
fdcf57f7-...	2016.04 - 2017.09	605	63515	5 - 33

A sample temperature measurement from these datasets in JSON format can be seen in Figure 5.1. To measure brightness, a TinkerKit Light Dependant Resistor (LDR) module is used, its resistance is decreased with higher brightness (it outputs 5 V under no light, and 0 V for the highest brightness). Table 5.3 shows the summary of brightness data in SmartME.

```
{
  "_id": 1,
  "Date": "2017-05-19T10:51:03.278000",
  "Temperature": 28.9273120410964,
  "Altitude": 5.35,
  "Latitude": 38.19401,
  "Longitude": 38.19401
}
```

Figure 5.1. Sample temperature sensor data of SmartME

Table 5.4 shows a statistics of humidity environmental properties for 5 datasets in the SmartME project. They were measured with Honeywell HIH-4030 sensors, which are ideal for battery powered application.

Table 5.3. Brightness measurements in the Smart City of Messina

Database ID	Interval	Freq. (sec)	No. of samples	Values (Lux)
c54a6a42-...	2017.05-06.	605	117	1 - 368
af811db2-...	2016.04 - 2017.09	605	6544	0 - 754
407eea34-...	2017.05-09	605	10452	1 - 1231
84e60570-...	2016.03 - 2017.09	605	46126	0 - 916
f8c80ed5-...	2016.04 - 2017.09	605	63515	0 - 2562

Table 5.4. Humidity measurements in the Smart City of Messina

Database ID	Interval	Freq. (sec)	No. of samples	Values (%)
9124f050-...	2017.05-06.	605	117	31 - 56
ea82840b-...	2016.04 - 2017.09	605	6544	17 - 110
337adbaa-...	2017.05-09	605	10449	18 - 113
bde446bc-...	2016.03 - 2017.09	605	46125	10 - 95
2be73f19-...	2016.04 - 2017.09	605	63518	10 - 90

To monitor air pressure, MPL3115 sensors are used with TinkerKit. They provide high resolution and support power saving mode, and their operating range is between 20 kPa to 110 kPa. Table 5.5 provides pressure-related statistics.

Table 5.5. Pressure measurements in the Smart City of Messina

Database ID	Interval	Freq. (sec)	No. of samples	Values (hPa)
e025857e-...	2017.05-06.	605	117	1005 - 1010
c1f5e699-...	2016.04 - 2017.09	605	6573	946 - 1007
e42905aa-...	2017.05-09	605	10457	1001 - 1024
4f185156-...	2016.03 - 2017.09	605	46123	3 - 1033
bd648c7f-...	2016.04 - 2017.09	605	63515	495 - 1033

For sensing noise, they use Arduino KY-038 microphone sound sensor modules providing both analog and digital outputs (with adjustable sensitivity). Table 5.6 summarizes noise measurement statistics for five datasets.

Besides environmental property monitoring, prototypes have been developed for smart energy, parking and traffic management. According to the information provided in the project websites [55] and [54], these parts of the SmartME project are still under development and testing, therefore partial datasets are available, and no real-time measurements are performed currently. Nevertheless we found it interesting and useful to examine the open data on traffic light lamps. The summary of these measurements published in a single dataset is shown in Table 5.7. In this set 10 lamps were monitored for 10 days, and their actual working hours and states were saved. A sample traffic light measurement from this dataset in JSON format can be seen in Figure 5.2.

Finally, some measurements on smart parking showing parking lot space information are also available (in the dataset with prefix: 298b8ab1-), but this set is refreshed unpredictably, only in a monthly basis.

Table 5.6. Noise measurements in the Smart City of Messina

Database ID	Interval	Freq. (sec)	No. of samples	Values (Amp)
a61d33f0-...	2017.05-06.	605	124	0 - 3
24ef5d60-...	2016.04 - 2017.09	605	6546	0 - 65
356224e4-...	2017.05-09	605	10451	0 - 73
9bf842ca-...	2016.03 - 2017.09	605	46120	0 - 55
5c1c9944-...	2016.04 - 2017.09	605	63518	0 - 151

Table 5.7. Traffic light measurements in the Smart City of Messina

Database ID	Interval	No. of samples	Working hours	Values
54b1f2ee-...	2017.06.17-26.	5713	0 - 3825	True / False

5.3.2 Sensor Data in the CityPulse project

CityPulse was a former European FP7 project [91] run between 2014 and 2016. As stated in their website [98], its goal was to develop innovative, reliable and real-time smart city applications by adopting and integrating the Internet of Things paradigm, knowledge-based computing and reliability testing. The project designed a flexible and extensible smart city data analytics framework. They developed software components for real-time data stream publication and annotation, installed at two locations: in Aarhus, Denmark, and in Brasov, Romania. The project team composed of citizens, stakeholders and technical colleagues, has developed over a hundred smart city scenarios that were published and made available online. The set of key datasets are published as CityPulse Reference Datasets that consists of over 180GB of time series data and live data feeds according to [29].

The CityPulse Dataset Collection containing various semantically annotated datasets is available in [30]. In this subsection we summarize the contents of these datasets archiving weather observations and parking-related monitoring information. The collection also includes datasets of specific events, which we found less relevant for further investigations and simulations, therefore we excluded them from our work. Most measurements are available in raw JSON format and in annotated TTL formats. Some measurements are also available in CSV format.

In the CityPulse project the following environmental properties were monitored both in Aarhus and Brasov: temperature and dew point (in °C), pressure (in mBar), humidity (in %), wind direction (in °) and wind speed (in kph). We could not find detailed information on the exact sensors used to perform these measurements. By examining the datasets we can see that there is no more real-time monitoring, and the publicly available data dates back to 2014, containing only some months of measurements. The data structure is different than the one used in the previously seen SmartME project. The frequency of sampling is also different: generally 3 measurements are performed in an hour (after 10, 20 and 30 minutes in a loop).

We examined in detail the two available datasets of weather observations from the city of Aarhus. The summary of these measurements are given in Table 5.8. From this table we can see that similar sampling is used for all properties, with the same measurement dates.

```
{
  "on": true,
  "working_hours": "3404",
  "id_lamp": "lamp-001",
  "timestamp": "2017-06-17T23:45:12",
  "count_power_on": "440",
  "lat": "37.55560",
  "lng": "14.98149",
  "_id": 1
}
```

Figure 5.2. Sample traffic light sensor data of SmartME

Table 5.8. Measurements in the Smart City of Aarhus

Property	Interval	Freq. (sec)	No. of samples	Values
Dew point (°C)	2014.02-06.	600-1800	8352	-9 - 15
Dew point (°C)	2014.08-09.	600-1800	3782	3-19
Pressure (mBar)	2014.02-06.	600-1800	8353	986-1038
Pressure (mBar)	2014.08-09.	600-1800	3782	992-1030
Humidity (%)	2014.02-06.	600-1800	8353	15-94
Humidity (%)	2014.08-09.	600-1800	3782	16-94
Temperature (°C)	2014.02-06.	600-1800	8353	-3 - 25
Temperature (°C)	2014.08-09.	600-1800	3782	2-27
Wind direction (°)	2014.02-06.	600-1800	8353	0-360
Wind direction (°)	2014.08-09.	600-1800	3782	0-360
Wind speed (kph)	2014.02-06.	600-1800	8353	0-63
Wind speed (kph)	2014.08-09.	600-1800	3782	0-39

Besides weather observations, we can find a parking-related dataset as well. It contains sensed information of 8 different parking lots in the city of Aarhus. The published data covers a period of 6 months. The statistical summary of this dataset is shown in Table 5.9.

Finally, pollution monitoring was supposed to complement parking space monitoring in the CityPulse project. Since no such sensor infrastructure was installed within the project, they decided to simulate this data and to generate artificial datasets for pollution information. The dataset was generated randomly (within a predefined range) to cover 3 months with 449 observation points, including the following properties: ozone, particulate matter, carbon monoxide, sulfur dioxide and nitrogen dioxide.

5.3.3 Sensor Data in the Smart City of Surrey

The government of the city of Surrey, Canada, has defined its Open Data Program in 2014 [36]. Their original goal was to set up an open, transparent and accessible government, and to empower citizens and to help small businesses to create value.

Table 5.9. Parking Data in the Smart City of Aarhus

Parking lot	Interval	Freq. (min)	No. of samples	Total spaces	No. of cars
8 places	2014.05-11.	30	55303	65 - 1240	0-1236
8 places	2015.02-10.	30	55303	65 - 1240	0-1236

The open data published through its projects is freely available for everyone to use and republish according to the Open Government Licence of Surrey (details in [61]). At the time of writing, 359 datasets are available through its website [60]. It has many dataset categories, e.g. land use, infrastructure, transportation, environmental services, business, economy and culture. One of the earliest datasets in this database is the Historical Climate Data, which contains environmental measurements sampled between 1929 and 2007. It contains information on measured minimum and maximum temperature values, and information on rainfall, snowfall and gust. Table 5.10 presents statistical information on this dataset.

Table 5.10. Historical Climate Data of Surrey

Type	Interval	Freq.	No. of samples	Values
Temperature	1929.01-2007.02.	Monthly	658	-9.7 – 25.2 °C
Rainfall	1929.01-2007.02.	Monthly	493	0 – 333 mm
Snowfall	1929.01-2007.02.	Monthly	496	0 – 92.7 mm

There are also many water-related measurements. Many of these samples are viewable and searchable through a service called COSMOS (City of Surrey Mapping Online System) which has a graphical interface [59]. They are able to monitor residential water consumption, and have access to various valves, mains, fittings and drainage monitoring devices. For example, water meters are specific devices used to measure the amount of water flowing through water connections. This dataset contains 65798 samples with various parameters, such as identifier, location (street, number), status and the geolocation of the device. Water pipes underground are used to transfer drinking water. A dedicated dataset is used to monitor their availability and properties including 28100 samples. Another possibly reusable dataset is the container of water levels of the river Fraser that flows nearby Surrey. The set contains measurements for two months in 2015 at two locations. Statistical data on this dataset is depicted in Table 5.11. Detailed statistical sample values of this dataset is shown in Table 5.12.

Table 5.11. Statistics on the river Fraser water levels dataset

Name	Interval	Freq.	No. of samples	Sample Size (kB)
waterlevels-2015march	2015.03.	60	1488	45
waterlevels-2015april	2015.04.	60	1433	43

In the group of environmental datasets we can find further information e.g. on tree planting (Park Specimen Trees). This sample set contains information on trees living in the parks, along the major roads or on specific properties of Surrey. In the transportation category we can find information related to cars, transit, routes and gas emissions. Using the Real Time Transit dataset, there are live transit information

Table 5.12. Statistics on the river Fraser water level monitoring stations

Name	Station	Value (m)
waterlevels2015march	Manson	-0,899 – 2,022
	192 ST	-0,649 – 2,062
waterlevels2015april	Manson	-0,752 – 1,673
	192 ST	0,3288 – 1,8522

including stops, stop estimates, buses, routes and statuses, which are also available in a map representation.

5.4 Filtered Datasets for Experimenting with IoT-Cloud Environments

In the previous sections, we introduced the datasets of two smart cities, and shown how to reuse them in simulations provided by SUMMON and MobIoTSim. There are many Smart City projects in the world, and this number is expected to grow in the near future. Hopefully, more and more datasets will be accessible publicly, and contain useful sensor information. To efficiently and easily work with several datasets located at different sources, we need a service which gathers and filters such data. There are three main functions that should be implemented in such a service:

- List the datasets
- Show detailed information about a specific dataset
- Query / filter a dataset

Figure 5.3. Opening page of SUMMON

For browsing and searching for the required sensor dataset for further reuse, a webpage is the best way to provide these features. To help other applications reuse the

data and the filtering capabilities, an API is also preferable. The dynamic list of the available datasets can be collected in different ways, based on the data source. A data source can be online accessible through its own API, or it can be downloadable stored in a file or in a local database. This list should contain basic information about the datasets, such as the name, source, categories and size. This list could even be searched, filtered and ordered to help finding the right dataset for the user or application. After selecting a specific dataset, more details can be obtained, like the date interval of the data, field names, types and ranges, with an example data sample. Even different statistics can be calculated and displayed in a table or with a chart. The query/filter should include three main parts:

- the first one is the opportunity to include or exclude fields from the results;
- the second part is responsible for the data filtering based on the field types. For example, if the field is an integer, the minimum and maximum values can be given, similar if it is a date type, just in a valid date format. In cases where the field contains a name or ID, the tool should display the possible distinct values. If the field type is boolean, the user could select a criterion to select only the true or false values, or both.
- The third main part is the specification of the result format. It can be a preview, or a downloadable file in JSON format or even a stream.

Figure 5.4. The basic operations of SUMMON

To implement the above mentioned requirements, we propose a tool called SUMMON, which is an abbreviated name coming from: Smart city Usage data Management and MONitoring. Its opening page can be seen in Figure 5.3, and its main features and operations are depicted in Figure 5.4. In the first prototype of the proposed tool we implemented some basic features. It is a website implemented in nodeJS and uses the CKAN system API. In case of the SmartME smart city project, a base URL for the API requests is also available. The CKAN system has a hierarchical structure, where the so called organizations are on the top. It is possible to list all organizations with the `http://smartme-data.unime.it/api/action/organization_list` call,

we can even request more information with the `all_fields=true` parameter, but unfortunately the datasets (or in CKAN dialect: packages) of an organization cannot be listed, which is the reason why we used a different API call, to query the datasets. We can get more detailed information about a dataset, like the licence, maintainer, author. SUMMON uses an API call to receive the schema of the dataset, so it can generate a form based on that. After the user is finished with the form, the settings are used to generate an SQL statement, which is run with the following call: `http://smartme-da-ta.unime.it/api/action/datastore_search_sql?sql=<SQL>` The results are displayed as a preview, embedded in an html code with the possibility to modify the form and see some example results, or in json format to be downloadable.

Figure 5.5. IoT environment simulation with SUMMON and MobIoTsim

In order to demonstrate the applicability of SUMMON, we use a mobile IoT device simulator called MobIoTsim. It was developed to target researchers and future IoT application users with the aim to exemplify IoT device handling without owning or buying real sensors. It can be used to test and demonstrate the inner workings of IoT-Cloud systems utilizing multiple devices. It can be easily connected to a private gateway service (e.g. the one available in [49] we formerly developed for the IBM Cloud platform). It is able to receive process sensor data coming from several devices simultaneously. It has a web-based graphical interface to visualize sensor data coming from MobIoTsim. Messages (defined in JSON format) sent by the simulated devices are managed by an MQTT server. It can also be used to send responses (or notifications) back to the simulated IoT devices in MobIoTsim.

As Figure 5.5 shows, the simulation of a whole IoT environment scenario starts with the filtered JSON data from SUMMON imported into the MobIoTsim simulator. When the user starts a simulation with MobIoTsim, the sensor data are sent in the

Summon

Welcome to SmartME data

Include/Exclude ☐ ☒

☐ _id

☐ id_lamp

On

Lamp id

Timestamp min

Timestamp max

Limit

JSON ☐

Pretty ☐

search

Figure 5.6. A form generated by SUMMON to filter a dataset

form of messages to the MQTT Broker of the IoT Gateway service. After it receives these real-life values, the broker forwards them to a gateway application. The gateway application shows a real-time chart with the data, even from several devices simultaneously. In cases, when we need just specific values from the dataset, or just a specific number or value of records, SUMMON can generate a form, where the required criteria can be added, as shown in Figure 5.6.

Edit Device

Type ID: TestType

Device ID: MobloTSimDevice02

Token: 8f3n4rE?mA-rCF-vR

Organization ID: hqst3j

Type: Custom

Number of devices: 1

Frequency: 2.0

Generate random data ☒

ADD

Parameter: parameter1 ✗

Min parameter: 10

Max parameter: 25

OK

Figure 5.7. Importing a file to MobIoTSim

The downloaded JSON data from SUMMON can be imported into the MobIoTSim simulator. The process can be done similarly to creating a new device – shown in Figure 5.7 –, but instead of generating random data, we can select the downloaded file to read

the values from it. This way, the created simulated device will send real-life data to the cloud. Even more simulated devices can be added, with the same or different data source imported from SUMMON.

5.5 Conclusion

Following the recent technological trends in ICT, IoT environments generate unprecedented amounts of data that should be stored, processed, and analyzed. Cloud systems already started to dominate the Internet with the appearance of things of the IoT paradigm, IoT-Cloud systems are formed. Fog Computing can be used to reduce service latency and to improve service quality by keeping and processing the data close to the user. IoT applications can also be supported by Fog technologies to achieve these goals, resulting in IoT-Cloud systems having many open issues.

In this chapter, I addressed some of these issues, and analyzed sensor data formats and datasets of three smart city projects. Based on the results of these investigations, I proposed a tool called SUMMON to provide means for gathering and filtering open datasets. I also discussed how filtered data of real IoT applications could be made available by this service to perform IoT-Cloud investigations in simulated environments. Finally, I demonstrated its utilization with an opensource IoT device simulator called MobIoTSim. The results of this chapter belong to Thesis III, and were published in papers [P1], [P9], [P7], [P11], [P8] and [P2].

Our future work will address the extension of our proposed data retrieval service by supporting additional sensor datasets from different IoT application areas, and we also plan to enable additional APIs for communicating with our service.

6

A Crawling Approach to Facilitate Open IoT Data Archiving

6.1 The need for gathering dynamic IoT and sensor data

According to recent reports in the field of the Internet of Things (IoT) (e.g. [109]), there will be 25 billion connected things by 2021. Such estimations call for smart solutions that provide means to connect, manage, and control these devices efficiently. IoT defines a global, dynamic network having self-configuring capabilities, in which devices (called things) can interact and communicate among themselves and with the environment by exchanging sensor data. Such systems can be utilized in many application areas, thus they may have very different properties.

Cloud computing [92] has already proven its potential to serve IoT needs such as hiding and performing data generation, processing, and visualization tasks by forming IoT-Cloud systems. Smart cities and smart regions can be realized by these complex systems, and designing and developing IoT applications on top of these environments represent a great challenge, since many things have to be managed at the same time, and a wide range of devices and data formats are available. As a result, their design and analysis should be supported by means of simulation.

To address this challenge, we already proposed a solution called Smart system Usage data Management and MONitoring (SUMMON), which provides an IoT and sensor data trace archive. It aims to contribute to the proliferation of integrated IoT and cloud technologies, by enabling simulated experimentation with applications using real data. Besides filtering and archiving datasets of smart city projects, it would be beneficial to reuse data from social networking sites and from such dynamically refreshed websites that provide visualized information of crowd-sourced smart systems that do not share their sensed data. To gather data from such services, we designed a web crawling approach to facilitate open IoT data archiving and reuse.

By simulating IoT-Cloud systems, working with real, open data could result in more

convincing and more reliable experimentation. In research works addressing distributed systems we can find data sources used for this purpose, e.g. in the Grid Workloads Archive [108], which contains trace files of job executions in real-world Grid systems. Our aim is to propose a way to build up a similar archive for the IoT-Cloud research community to facilitate simulation experiments.

The main contributions of this chapter are: (i) the design of an automated web crawling service that gathers and stores historical IoT and sensor data in SUMMON, and (ii) its application to three real use cases of smart cities and smart regions.

The remainder of this chapter is presented as follows: Section 6.2 details the latest extension of our trace archive service called SUMMON, and details the development, operation and evaluation of our proposed web crawler service that automatically gathers and imports IoT traces to SUMMON. Finally, the contributions are summarized in Section 6.3.

Figure 6.1. An environment for simulating IoT-Cloud systems

6.2 Extending SUMMON with a web crawling service

The architecture of our vision for simulating IoT-Cloud systems is depicted in Figure 6.1. Compared to traditional simulator environments, we try to stay as close to real world systems as possible. We gather trace files saved from public applications with a crawler service (1), make them available through an open trace archive (2), and use them in a device simulator (3) to mimic real device behavior. We also developed different gateway services (4) to process and visualize sensor data and instantiate and operate them at private or public cloud providers.

The aim of our last extension to our semi-simulation framework is to provide a solution for component I., in the form of an automated web crawler service. For archiving

and open data access we use SUMMON, for device simulation we use MobIoTSim, while to perform measurements, visualize and store simulated behavior and data we use the generic IoT gateway for MobIoTSim.

The basic usage of the simulator environment is to: (i) connect to the trace archive service to retrieve sensor data for the simulations, (ii) connect the application to a real cloud, where the historical data is to be sent, (iii) create and configure the devices to be simulated, and (iv) start the data generation of the required devices.

SUMMON was created in 2017 with the aim of making real-world IoT sensor data easily accessible to researchers. In 2018 we made revisions and further developments to ease the gathering, presentation and accessibility of the trace data. The updated source code of the service is available in [68].

The screenshot displays the SUMMON web interface for importing trace data. At the top, there are two dropdown menus: 'City of Surrey' and 'Internet Connectivity'. Below these, the 'Request URI' field contains 'https://api.spacexdata.com/v3/launches/latest', with 'TEST API' and 'TOGGLE' buttons to its right. The 'Name' field is set to 'SpaceX API', and the 'Frequency' dropdown is set to 'Every Hour'. Two toggle switches are present: 'Enable switch to run the query and save it to the database at the selected interval' (turned On) and 'Enable switch so the saved queries will be downloadable' (turned On). Below the toggles are 'NEW ELEMENT' and 'REMOVE LAST ELEMENT' buttons. A note states: 'There is no validation for the queries, for more information use this documentation.' The 'Query Key - 0' section shows 'rocketID' as the key and 'rocket.rocket_id' as the value. The 'Query Key - 1' section shows an error: 'Key can't be empty'. On the right, a JSON preview shows the structure of the data being imported, with a 'Query' field containing the string 'falcon9'.

Figure 6.2. Importing trace data to SUMMON through a REST API

The architecture of SUMMON builds on the latest standards and web development approaches. The service is composed of three micro-services implemented in Docker containers. We use MongoDB to store sensor and IoT data in JSON format. The backend is a Node.js Express server application, which is responsible for managing the database in case of storing additional data, or retrieving already stored trace items. The communication among the components is performed by secure web tokens (using the bcryptjs package) through restful APIs. The frontend is implemented with the React package in Node.js. It is responsible for graphical presentation and navigation.

Figure 5.3 shows the starting page of the portal. Beside the basic information displayed here, the left upper menu can be used to select the appropriate trace logs in the form of datasets, assigned to certain categories. Once a dataset is selected, one can view its content through dynamic charts based on its attributes. In this page we can also find information on the data provider or project that originally hosted the data, as well as about licensing information to reuse the data.

Figure 6.3. Visualizing trace data gathered from OpenWeatherMap through the REST API

Placing datasets into SUMMON can be done in two different ways. The first, simpler one is to register a new dataset through the administrative interface (reachable with admin role), and upload a previously downloaded trace file as static data. In this case there will be no charts available to get insights on the data values, but we have the possibility to manually filter and reformat the data to store each element in the database to enable detailed queries for chart generation. The second option is to use the special REST API to connect SUMMON to a server and let it regularly query and automatically store data by predefined attributes to the database in JSON format, as depicted in Figure 6.2. In this way we can automate data retrieval and storage, and chart generation will be performed automatically, once an attribute of the dataset is selected. Figure 6.3 displays a dataset gathered through the REST API from the OpenWeatherMap portal [46] including information on the weather properties of the city of Szeged. By the end of 2018, we set up a prototype portal, and uploaded 42 datasets from three smart city projects, and a dataset from a weather prediction

service.

Next, we planned to provide a way to gather and store sensor and IoT data scattered over websites of various projects. We also planned to exploit the REST API-based, automatic data retrieval of SUMMON for this purpose. To achieve this, we had to develop a special service that performs website crawling and makes the collected data available to SUMMON.

Figure 6.4. Architecture of the proposed crawler service

To create such a service we used Scrapy [82], which is an opensource python-based framework for data mining. Its main components are the following:

- Engine: it is a central entity in the system that is responsible for communication among the system components.
- Scheduler: it performs scheduling (ordering) for the requests coming from the Engine.
- Spider: it is represented by a class defined by the user. It should include the logic for extracting data (named by items) from the responses resulting from crawling websites, and should perform further requests, if needed.
- Downloader: this component performs the crawling and sends the results to the Spider through the Engine.
- Item Pipeline: it processes the items (data) extracted from the responses of the Downloader by the Spider. It further filters and validates the items and stores them in the database.
- Middleware: it is represented by special components standing between the Engine and the Downloader or between the Engine and the Spider. They generally manage requests and responses between these components.

We also used Scrapy [83] to create our crawler service, which is an application for managing multiple processes of the spiders' requests in a parallel way. It also provides a JSON-based web service for managing multiple Scrapy projects at the same time. We realized the service as a container-based Docker application as depicted in Figure 6.4. A Flask python server application is responsible for regularly (2 hours interval is predefined) starting Spider jobs and managing requests among the (i) Scrapy micro-service, where the crawling logic is defined, the (ii) MongoDB database micro-service, where the gathered data are (temporarily) stored, and (iii) SUMMON, which is the external trace archive.

Table 6.1. Internals of the crawler service

Endpoint	Description
/	main page
/scrapy/jobs	table of registered Scrapy jobs
/scrapy/start/{ <i>Project</i> }	to start a Spider in a <i>Project</i> ; Spider name in GET parameter
/data/{ <i>Spider</i> }/keys	to query the keys as document fields
/data/{ <i>Spider</i> }/query?q={query}	to list <i>Spider</i> attributes with keys; additional GET parameters: from - starting date; to - end date;

Table 6.1 summarizes the REST endpoints of the crawler service revealing its potential internal operations: we can list the actually running Scrapy jobs, add further jobs, and query gathered data by the jobs from the database. An example for the query is: "/data/idokep/query?q=location=Szeged&wind=7 m/s". This means that we query the results of the Spider called "idokep", and would like to list the messages sensed in "Szeged" (location) having 7 m/s wind speed.

Project	Spider	Job	PID	Start	Runtime	Finish	Log
Pending							
Running							
livetraffic	livetraffic	4041838a799d11e982680242ac130003	11	2019-05-18 18:46:31	0:00:17		Log
parking	parking	49061bb6799d11e982680242ac130003	100	2019-05-18 18:46:46	0:00:02		Log
Finished							
idokep	idokep	44cf970c799d11e982680242ac130003		2019-05-18 18:46:41	0:00:04	2019-05-18 18:46:45	Log

Figure 6.5. Web interface of the proposed crawler service

We deployed a prototype instance in the Google Cloud, and connected a running SUMMON prototype to it through a REST API connection. In this way the data gathered by our crawler service can be regularly retrieved and stored in SUMMON. The web interface of the service depicting three Scrapy jobs (two actually running, one is finished) can be seen in Figure 6.5.

6.2.1 Case study 1: Idokep.hu

The downloadable static data and dynamic data streams through APIs can be easily imported to SUMMON. IoT data gathering is only a problem for projects and websites

publishing data with graphical elements of websites, or with any form of items regularly refreshed on a page (meaning that the expired data is deleted or overwritten). In these cases there are no APIs or links provided to get the sensed or monitored information. We found that, in general crowdsourcing and social networking platforms, and in some cases smart city project sites belong to this category.

Table 6.2. The structure of the analyzed website

Fields	XPath
location	<code>./a[@class="helyseg"]/@href</code>
time	<code>./div[@class="mikor"]/text()</code>
sky_picture	<code>./div[@class="sor"][1]/div[2]</code>
temperature	<code>./div[@class="fok"]/text()</code>
air_pressure	<code>./div[text()='Légnyomás: "]/strong/text()</code>
humidity	<code>./div[text()='Páratartalom: "][2]/strong/text()</code>
max_temperature	<code>./div[text()='Tmax: "]/strong/text()</code>
rain	<code>./div[text()='Csapadék: "]/strong/text()</code>
wind	<code>./div[text()='Szél: "]/img/@src</code>

```
class IdokepItem(scrapy.Item):
    location = scrapy.Field()
    sky_picture = scrapy.Field()
    time = scrapy.Field()
    temperature = scrapy.Field()
    air_pressure = scrapy.Field()
    humidity = scrapy.Field()
    max_temperature = scrapy.Field()
    rain = scrapy.Field()
    wind = scrapy.Field()
```

Figure 6.6. Data fields of a sensing message originated from Idokep.hu

As one of the earliest examples of sensor networks are coming from the field of meteorology and weather prediction, we chose to provide trace gathering for the crowd-sourced meteorological service of Hungary called Idokep.hu [85], which can be considered as a smart region type IoT application. It has been established in 2004, and it is one of the most popular weather prediction sites in Hungary. Since 2008, weather conditions of Croatia and even some parts of Germany are available. Detailed information of its system architecture and operation can also be found on the website: more than 400 stations send sensor data to their system (including temperature, humidity, barometric pressure, rainfall and wind properties), and the actual weather conditions are refreshed every 10 minutes. They also provide detailed forecasts for up to a week. To ease crowdsourcing, they produce and sell sensor stations that can be deployed by individuals to extend their sensor network to improve their weather predictions. Mobile sensing is also supported, users can post messages of the sensed weather conditions including location, temperature, cloudiness, wind direction and speed, and rainfall. This option is the most interesting for us: once we could save such posting messages,

we could also reuse them to simulate crowdsourcing systems of different scales and behavior.

Since we needed a crawling solution it is obvious that these posting messages cannot be downloaded in batch files, they are only viewed at their website (with the URL: "https://www.idokep.hu/eszlel"). Here only the posts from the past 30 minutes are visible, and each post contains the nickname of the user, the time of sensing, an icon of the sensed weather condition, optional parameters for weather properties, and possibly some clarifying text. We used the ChroPath [63] tool, a Chrome browser plugin, to analyze the website and determine the XPath structure and contents – shown in Table 6.2. To save the post messages to the MongoDB database, we use the data format shown in Figure 6.6. The field names are self-describing, we use the "sky_picture" field for storing the used icon. To get the values of the fields, we had to perform various searching and filtering methods on the HTML source of the site. We also pseudonymised the nicknames for privacy issues.

```
class LivetrafficItem(scrapy.Item):
    location = scrapy.Field()
    date = scrapy.Field()
    title = scrapy.Field()
    desc = scrapy.Field()
```

Figure 6.7. Data fields of a sensing message originated from theaa.com

6.2.2 Case study 2: AA Traffic news

For the second use case we have chosen a smart city application. Fortunately, there are some smart city projects that make their sensed data downloadable in historical batch files, such as the SmartME project, which we used earlier to save and upload to SUMMON. But there are many other related projects that do not share trace files and do not provide APIs to access their data. The Automobile Association Developments Ltd. operates a website for route planning and live traffic news [81], which can be considered as a service of smart cities. The Automobile Association (AA) was created in 1905 by local drivers in London, and has grown over 15 million users to date. Their main goal was to provide breakdown cover insurance, but nowadays they have many services, some are operated with crowdsourcing methods.

On their webpage we can view a map of the United Kingdom, and by zooming in we can find special icons for posts of live traffic information. They differentiate six icon types corresponding to traffic conditions: incidents, roadworks, congestion, road closures, lane closed, and weather incidents. The icons to be viewed on the map are generated dynamically, and by clicking on one of them a popup window comes up for detailed information. Once a new report is viewed, the earlier disappears, which makes the site hard to be crawled. To gather the sensed messages from this site, we used the Selenium tool [84], another Chrome extension, for crawling over the popup windows of the site. Figure 6.7 shows the data fields and format we used to save these post messages to our database – we tried to save all information of a post, "desc" stores the actual traffic condition.

Project	Spider	Job	PID	Start	Runtime	Finish	Log
Pending							
Running							
Finished							
idokep	idokep	513b9dd8cde911e9a0d30242ac120002		2019-09-03 01:22:49	0:00:03	2019-09-03 01:22:52	Log
livetraffic	livetraffic	513d5772cde911e9a0d30242ac120002		2019-09-03 01:22:44	0:00:09	2019-09-03 01:22:53	Log
parking	parking	14c89a52cdfa11e9a0d30242ac120002		2019-09-03 03:22:39	0:00:11	2019-09-03 03:22:50	Log
idokep	idokep	14c50360cdfa11e9a0d30242ac120002		2019-09-03 03:22:49	0:00:04	2019-09-03 03:22:53	Log
livetraffic	livetraffic	14c6efaecdffa11e9a0d30242ac120002		2019-09-03 03:22:44	0:00:10	2019-09-03 03:22:54	Log
parking	parking	d8507c1ece0a11e9a0d30242ac120002		2019-09-03 05:22:39	0:00:11	2019-09-03 05:22:50	Log
idokep	idokep	d84d5f70ce0a11e9a0d30242ac120002		2019-09-03 05:22:49	0:00:03	2019-09-03 05:22:52	Log
livetraffic	livetraffic	d84f0640ce0a11e9a0d30242ac120002		2019-09-03 05:22:44	0:00:10	2019-09-03 05:22:54	Log
parking	parking	9bda3c00ce1b11e9a0d30242ac120002		2019-09-03 07:22:39	0:00:11	2019-09-03 07:22:50	Log
idokep	idokep	9bd702f6ce1b11e9a0d30242ac120002		2019-09-03 07:22:49	0:00:04	2019-09-03 07:22:53	Log
livetraffic	livetraffic	9bd8d162ce1b11e9a0d30242ac120002		2019-09-03 07:22:44	0:00:10	2019-09-03 07:22:54	Log
parking	parking	5f61f874ce2c11e9a0d30242ac120002		2019-09-03 09:22:39	0:00:10	2019-09-03 09:22:49	Log
idokep	idokep	5f5f1f14ce2c11e9a0d30242ac120002		2019-09-03 09:22:49	0:00:04	2019-09-03 09:22:53	Log
livetraffic	livetraffic	5f60a708ce2c11e9a0d30242ac120002		2019-09-03 09:22:44	0:00:09	2019-09-03 09:22:53	Log
parking	parking	22eaf272ce3d11e9a0d30242ac120002		2019-09-03 11:22:39	0:00:10	2019-09-03 11:22:49	Log
idokep	idokep	22e80a6cce3d11e9a0d30242ac120002		2019-09-03 11:22:49	0:00:04	2019-09-03 11:22:53	Log
livetraffic	livetraffic	22e9b7aece3d11e9a0d30242ac120002		2019-09-03 11:22:44	0:00:09	2019-09-03 11:22:53	Log
parking	parking	e6738df2ce4d11e9a0d30242ac120002		2019-09-03 13:22:39	0:00:11	2019-09-03 13:22:50	Log
idokep	idokep	e670a448ce4d11e9a0d30242ac120002		2019-09-03 13:22:49	0:00:04	2019-09-03 13:22:53	Log
livetraffic	livetraffic	e67237a4ce4d11e9a0d30242ac120002		2019-09-03 13:22:44	0:00:10	2019-09-03 13:22:54	Log
parking	parking	a9fd8ca4ce5e11e9a0d30242ac120002		2019-09-03 15:22:39	0:00:11	2019-09-03 15:22:50	Log
idokep	idokep	a9fa5b74ce5e11e9a0d30242ac120002		2019-09-03 15:22:49	0:00:04	2019-09-03 15:22:53	Log
livetraffic	livetraffic	a9fc2224ce5e11e9a0d30242ac120002		2019-09-03 15:22:44	0:00:10	2019-09-03 15:22:54	Log
parking	parking	6d863e36ce6f11e9a0d30242ac120002		2019-09-03 17:22:39	0:00:10	2019-09-03 17:22:49	Log
idokep	idokep	6d8303c4ce6f11e9a0d30242ac120002		2019-09-03 17:22:49	0:00:04	2019-09-03 17:22:53	Log
livetraffic	livetraffic	6d84c600ce6f11e9a0d30242ac120002		2019-09-03 17:22:44	0:00:10	2019-09-03 17:22:54	Log
parking	parking	310de236ce8011e9a0d30242ac120002		2019-09-03 19:22:39	0:00:10	2019-09-03 19:22:49	Log
idokep	idokep	310b19a2ce8011e9a0d30242ac120002		2019-09-03 19:22:49	0:00:04	2019-09-03 19:22:53	Log
livetraffic	livetraffic	310c9fcace8011e9a0d30242ac120002		2019-09-03 19:22:44	0:00:10	2019-09-03 19:22:54	Log
parking	parking	f497d454ce9011e9a0d30242ac120002		2019-09-03 21:22:39	0:00:10	2019-09-03 21:22:49	Log
idokep	idokep	f494c868ce9011e9a0d30242ac120002		2019-09-03 21:22:49	0:00:04	2019-09-03 21:22:53	Log
livetraffic	livetraffic	f4968a4ace9011e9a0d30242ac120002		2019-09-03 21:22:44	0:00:10	2019-09-03 21:22:54	Log
parking	parking	b8203b7ccea111e9a0d30242ac120002		2019-09-03 23:22:39	0:00:10	2019-09-03 23:22:49	Log
idokep	idokep	b81d5876cea111e9a0d30242ac120002		2019-09-03 23:22:49	0:00:04	2019-09-03 23:22:53	Log
livetraffic	livetraffic	b81ef604cea111e9a0d30242ac120002		2019-09-03 23:22:44	0:00:11	2019-09-03 23:22:55	Log

Figure 6.8. Executed measurements in the web interface

```

class ParkingItem(scrapy.Item):
    time = scrapy.Field()
    address = scrapy.Field()
    location = scrapy.Field()
    used = scrapy.Field()

```

Figure 6.9. Data fields of a sensing message originated from parkme.com

In this case we also had to perform an XPath analysis and source page filtering. For the defined spider we set the London region to be monitored (with the URL: "https://www.theaa.com/route-planner/traffic-news/London, UK") – containing 100 posts generally, in average.

6.2.3 Case study 3: ParkMe

Finally, for the third use case we chose the ParkMe project [77] of the Inrix company founded in 2005, which is also a smart city application. It provides a cloud-based parking management solution to simplify the monitoring and management of parking. It offers a dashboard for browsing parking places in a map to be displayed by a searching for a city. Once a parking spot is selected, it opens a popup window for viewing its detailed information: it shows the time, location, address, actual occupancy (in %), and optionally actual prices in local currency. The data fields and format we use in this case can be seen in Figure 6.9. We can see that we store a timestamp, the address and location of the parking place, and the current occupancy (with the "used" field). Again, here we had to perform XPath analysis and source page filtering, as well. For the defined spider we set a part of the downtown Budapest region to be monitored (with the URL: "https://www.parkme.com/map?q=#budapest") – containing 33 parking spots (with the map resolution we used).

6.2.4 Evaluation

We wanted to evaluate the operation and usability of our proposed crawler service. We can also use the results to get an impression on the difficulty and the sophistication needed for gathering sensor and IoT data from the three different application areas. We deployed the crawler service to a small, standard VM (1 virtual CPU core, 3.75 GB virtual memory, 10 GB HDD) with Ubuntu 18.04, started in the Google Cloud Platform. We scheduled to perform measurements for crawling the above mentioned three websites by executing a job of each spider of the registered projects every two hours. Figure 6.8 displays the successfully executed crawling tasks on a day (3rd September, 2019).

Table 6.3. Comparison of the three use cases

Project	Time (sec)	Data (KB)	No. of messages
idokep	4	20.24	50
livetraffic	30	17.7	47
parking	10	7.48	33

Table 6.3 shows the average values for the execution times, saved data sizes and number of messages for the spider jobs run on this day. From these results we can see that the web page layouts of the corresponding smart systems vary highly, therefore no generic spider can be used to crawl them. *idokep* has a fixed number of posts/messages to be saved, due to its refreshing method. *livetraffic* and *parking* use maps for visualization, and they can be zoomed in to specific regions having different numbers of posts. While *parking* has a fix number of parking places for a preset region, in case of *livetraffic* the number of posts can vary even for a preset region. For the *livetraffic* case, we unfortunately found that its website was not always reliable, a few times our crawler service encountered timeout errors.

6.3 Conclusion

The IoT paradigm has appeared as the latest technological evolution on the Internet coupling a huge amount of powerful devices together in the online world. By responding to this trend, several cloud providers have started to offer specific management services to form complex IoT-Cloud ecosystems that are hard to be analyze without simulation tools.

To support users and researchers in this field, we proposed SUMMON, an IoT trace archiving service that can be used to gather real world datasets, and to reuse them for simulated experiments. We continued this research and provided an extension to SUMMON with an automated web crawling service that gathers IoT and sensor data from publicly available websites. These tools can be used by simulators to retrieve real-world sensor data for realistic evaluations. We introduced the architecture and operation of this approach in detail, and exemplified its utilization with three, real world use cases of smart systems.

The results of this chapter belong to Thesis III, and were published in papers [P11], [P8], [P2] and [P3].

“Unless you do the work,
the world will never see your vision.”

— Jocko Willink

7

Conclusions

In the phase of the latest Internet evolution, a high number of powerful devices appeared on the network, which led to the birth of the IoT paradigm. As a result, cloud technology providers responded to this trend, and broadened their services to attract users of IoT applications. In this new world, IoT developers are facing many challenges: they need to purchase smart devices, to design and build-up a network of these components, to develop IoT applications, to test and evaluate these applications on the created system and finally, to fine-tune the applications based on the evaluation results.

In this work I provided means to overcome some of these challenges. I developed a complex, semi-simulation environment composed the following tools: an Android-based, mobile device simulator called *MobIoTSim* capable of simulating up to hundreds of IoT devices, a customizable cloud gateway service that is able to manage the simulated devices, and an IoT trace archive service called *SUMMON*, which can be used to gather real-world sensor data to be fed to the simulation experiments. I exemplified the usage of this complex simulation environment, and evaluated its device management scalability and responsiveness. I do believe that these tools can contribute to a more cost efficient IoT system planning, development and evaluation, even for different academic research communities and industrial stakeholders.

Concerning future works, though the results of this dissertation represent a complete ecosystem for investigating IoT-Cloud systems, there is still some room for improvements and extensions.

In case of the *MobIoTSim* simulator, I plan to develop further predefined templates for public cloud providers for the Cloud Settings view to facilitate easier configurations. I also plan to perform some re-engineering on its source code, and make it available on Google Play Store. Regarding the *IoT gateway*, I plan to investigate scalability issues of the underlying MQTT broker, and to provide means for switching between different MQTT implementations. Finally, for data archiving in *SUMMON*, we are keep on looking for additional IoT and smart application areas to enlarge the number of traces available in the archive for enabling various simulations with a wider set of experimental data.

Most of the research I performed in the past five years has resulted from active involvement in the Internet of Living Things (GINOP-2.3.2-15-2016-00037), Smart Systems Research Institute (20391-3/2018/FEKUSTRAT and TUDFO/47138-1/2019-ITM), EFOP-3.6.1-16-2016-00008, and OTKA FK 131793 national, and EU COST Action IC1304 (ACROSS) and EU COST Action CA17136 (INDAIRPOLLNET) international projects.

The scientific results presented in this dissertation have been published in numerous journals, conference and workshop papers and have been presented in various scientific forums. These publications have inspired further research, generated collaborations, and are well represented by many independent citations – which are summarized in Table 7.1. The components of the proposed semi-simulation environment are under continuous development and improvement, their source codes are open and available on GitHub at the following links:

- <https://github.com/sed-szeged/MobIoTSim/>
- <https://github.com/sed-szeged/MobIoTSimBluemixGateway>
- <https://github.com/sed-szeged/iotgateway>
- <https://github.com/sed-inf-u-szeged/IoLT-Smart-Pot-Gateway>
- <https://github.com/sed-szeged/SUMMON>

Table 7.1. Publications, Theses and Independent citations

	Thesis I.	Thesis II.	Thesis III.	Independent citations	
				Google Scholar	MTMT
[P14]		◆		9	4
[P12]		◆			
[P13]		◆		1	
[P6]	◆	◆		35	21
[P1]			◆	1	
[P10]	◆	◆		12	6
[P9]	◆	◆	◆	1	1
[P4]	◆	◆		2	
[P7]	◆		◆	5	
[P11]	◆	◆	◆	1	1
[P8]	◆		◆	1	1
[P2]	◆	◆	◆	12	8
[P3]			◆		
[P5]		◆			
Sum	8	10	7	80	42

Appendices

Summary in English

A.1 Introduction

The Internet of Things (IoT) is one of the latest trends of the current ICT evolution, represented by a huge amount of powerful smart devices that have started to appear on the Internet. According to reports in this field, Gartner [124] predicted that there will be 30 billion devices online, and more than 200 billion devices discontinuously online by 2020, while Huawei foresees 100 billion new IoT connections till 2025 [90]. By responding to this new trend, many cloud providers have started to offer services for IoT management. Recent advances have already shown that Cloud Computing can be used to serve IoT needs by performing data generation, processing and visualization tasks. Gubbi et al. [105] were one of the pioneers by describing IoT systems in 2013, and they also emphasized the importance of the joint utilization of IoT and clouds. Botta et al. presented the main properties, features, and open issues of such combination in [92], while Nastic et al. [122] listed the numerous challenges of realizing IoT-Cloud systems in practice. These related works also served as motivations for my research by raising the need for managing a large number of protocols and data formats by means of simulation.

To create such ecosystems, IoT system developers need to purchase, connect and configure these devices, and they also have to choose the right infrastructure provider offering the combination of protocols and data structures fitting their applications. To ease these tasks, I propose a semi-simulation environment in this dissertation that can be used to investigate the inner workings of complex, IoT-Cloud systems. This environment consists of three components that represent the main contributions of my work: (i) an Android-based, mobile IoT device simulator called MobIoTSim that can mimic real world sensors and devices; (ii) a customizable cloud gateway to manage IoT devices by receiving, processing and visualizing sensor data; and (iii) an IoT trace archiving service called SUMMON that can be used to gather real world sensor data.

One of the main motivating forces behind my research was that several cloud providers had started to offer IoT-specific services to ease the development of IoT-Cloud applications, but such cases, where many different things needed to be operated

and managed at the same time, were hard to realize. Just to mention a few, smart city application scenarios using SIGFOX [53] or LoRa [41] technologies are very expensive and time-consuming to be deployed with real devices, since a base station costs up to a thousand euros needing enormous configuration work. In my thesis I respond to these challenges by designing and developing solutions for simulating IoT devices and managing the simulated applications with real world cloud services.

Concerning state-of-the-art solutions, some of the the existing simulators for examining distributed systems can be categorized as general network simulators, and there are also specialized, IoT simulators addressing issues raised by this work. After examining the related simulators in the IoT field, I found wanting to have a solution that puts an emphasis on mobility, and provides means for performing simulations at specific geolocations. This fact served as a major motivation for designing a simulator for mobile devices. The main purpose of such a mobile IoT device simulator is to help cloud application developers to learn IoT device handling without buying real sensors and to test and demonstrate IoT applications utilizing multiple devices.

A.2 New scientific results

Thesis I. I designed and developed a mobile IoT simulator called MobIoT-Sim, which can simulate multiple IoT devices in a user-friendly way, and send the generated data on the real network to the cloud. I validated its usability, and evaluated its scalability for managing multiple devices simultaneously.

My proposed mobile IoT device simulator is designed to simulate up to hundreds of IoT devices, and it is implemented as a mobile application for Android platforms. Sensor data generation of the simulated devices can be done by artificially generated, random values in a range given by the user – this is the default option. For more advanced simulations, data from IoT-specific trace files can also be loaded and replayed. The data sending frequency can be specified for each device for both cases. The application uses MQTT protocol (by default) to send messages with the use of the Eclipse Paho library, which is open source. The MQTT protocol is an established messaging protocol for IoT systems [120] and widely supported by IoT-Cloud services. Message data is represented in a structured JSON object in the simulator, compatible with the IBM IoT Foundation message format.

The basic usage of the simulator is to connect the application to a cloud, where the data is to be sent, to create and configure the devices to be simulated, and to start the (data generation of the) required devices. These main steps are represented by three main parts of the application: the *Cloud settings*, the *Devices* and the *Device settings* screens.

Sensor data generation for the simulated devices can be done in different ways. The default way to generate device data is to create a custom device, where the user can define parameters with ranges, within the random value will be generated (by default the *Generate random data* option is activated). To simplify this process the user can use device templates, which has predefined parameters and ranges, specific to the device type, and also an artificial, random value generation will be performed. If the *Generate random data* option is deactivated, the user can select a file to load the sensor values. In this way, we can replay previous experiments. Beside using random generation and replays, we can even load trace files of real world applications gathered

from public sources. An example file from the OpenWeatherMap public weather data provider [46] can be used, which monitors many cities and stores many parameters of them, including temperature, humidity, air pressure and wind speed. Using the trace loader feature of MobIoTSim, the simulation can mimic a real life scenario.

I analyzed the device simulation scalability performance of the MobIoTSim application. For this purpose I created a simplified version of MobIoTSim (called MQTTDemo), containing only its core parts for device management functionalities to enable automated, script-based message generation. It allows for accessing low-level configurations, like specifying the number of threads used for managing the simulated devices. It also collects detailed statistical information of the simulation by measuring elapsed times for executing certain functions. It can be connected to the private gateway deployed in the IBM Cloud, and can send messages to it using the MQTT protocol. By the time I performed the evaluations, the number of simulated devices were limited to 20 by the IBM Cloud platform. In the Settings we can define the number of devices to be simulated, the message frequency for sensor data generation, the number of used threads (*ThreadPool*) and the message type and contents. This type can be a simple JSON object with a random parameter, or an OpenWeatherMap device data in JSON format describing sensor values of certain smart cities. Other settings are hard-coded to the application for this testing, as well as the address of the MQTT broker of the IBM Cloud service.

The evaluation results proved that the random data generalization consumed an almost negligible time, so it did not interfere with the simulation. To manage 20 devices (the highest number for IBM Cloud, i.e. Bluemix till 2017), we can see that six threads were too few for an acceptable performance, therefore we had to use eight threads instead.

Thesis II. I designed and developed a generic cloud gateway service, which can be deployed on any cloud service, and can manage IoT devices by receiving, storing and processing or visualizing data. I also validated its usability together with MobIoTSim, and evaluated its scalability.

After examining the IoT services of public cloud providers, I performed a requirement analysis for a service that is easy to use and able to handle both simulated and real world IoT devices. Based on these results, I developed a customizable, private gateway that can be deployed to any cloud provider to handle up to hundreds of different devices at the same time.

This generalized gateway is a node.js application that is able to manage multiple device topics with the MQTT protocol, and it has a web-based graphical interface to visualize sensor data coming from MobIoTSim. Messages (defined in JSON format) received from the simulated devices are managed by an MQTT server. It can also be used to send responses (or notifications) back to the simulated IoT devices in MobIoTSim. To support multiple device handling, it uses paging to overcome device management limitations introduced in the free version of the IBM Cloud. In order to enhance and better visualize heterogeneous device data simultaneously, I introduced device grouping for the chart generation. This way we can see the data sent by all devices in a single real-time chart. Though the IBM Cloud provides some monitoring information for deployed applications, we can have access to more detailed monitoring data using custom Docker containers. As a result, I placed my extended gateway solution to a Docker container, which has also become more portable at the same time.

I evaluated the scalability of this generic gateway service. I deployed the gateway

in a Docker container as a micro application to the IBM Cloud. I performed multiple rounds of evaluation. In the initial phase evaluation, I used the Thermostat template of MobIoTSim to create 900 simulated devices in groups. These results indicated that even with this high number of devices only a little load could be observed. For the second round of experiments, I introduced CPU stressing, and created device groups for 10, 100, 250 and 450 devices in MobIoTSim, and re-executed the simulation. The results showed that for doubled messages (i.e. 0.5 second frequencies) the CPU utilization is also doubled. In the last evaluation round I also created 10, 100, 250 and 450 devices in four groups, using the Weather template of MobIoTSim. Since MobIoTSim is capable of loading previously saved IoT trace data, I used this feature for loading real data provided by OpenWeatherMap. I performed the experiments for all device groups with one second frequency. The results showed that the Weather template uses bigger messages, which also results in higher CPU load.

Smart farming is also a rapidly growing area within smart systems, that need to respond to great challenges of the near future. Due to recent ICT developments we can apply novel sensor and IoT technologies to provide a promising alternative, called affordable phenotyping. To address these aims, in the frame of the Internet of Living Things project [89], I participated in developing a low cost plant phenotyping platform for small sized plants, which enables the remote monitoring of plant growth in a standard greenhouse environment, called the IoLT Smart Pot. To manage this smart pot cluster, I designed and participated in the implementation of the IoLT Smart Pot Gateway for monitoring its environmental parameters, based on the results of the generic gateway solution. I also participated in the evaluation of the proposed solution with scalable simulations, and exemplified its real world utilization.

Thesis III. I designed and developed an IoT trace archiving service called SUMMON, which can periodically collect and store data from real IoT applications, even with using web crawling techniques. I demonstrated its operation with three use cases. The gathered datasets are shared in SUMMON to facilitate more realistic IoT simulations.

The European Commission has published a guideline document [99] for encouraging open access to reuse digital research data generated by Horizon 2020 projects. In this report they defined the so-called FAIR data principles to help Horizon 2020 beneficiaries to make their research data findable, accessible, interoperable and reusable. Following these guidelines, I aim to create an open IoT trace archive to facilitate data search and accessibility, and to reuse sensor data for experimenting with IoT-Cloud systems.

In the first versions of MobIoTSim one could only use artificially generated values for sensor simulation. Later, I developed a trace loading feature to support easier configuration of experiments, and to replay real world sensor data in certain simulated scenarios. I introduced a Weather template for representing weather monitoring features of smart city devices, for which I used data from the OpenWeatherMap service. To further extend the supported application areas, I envisioned a public IoT trace archive to be used together with MobIoTSim. For this aim, I developed a solution called Smart system Usage data Management and MONitoring (in short SUMMON), which is a web service for gathering and filtering data from these projects. With this service we can list the available datasets, and query and filter a selected dataset to be loaded by MobIoTSim for large scale experiments. I gathered and filtered real sensor data of smart cities, such as the SmartME [93] and the CityPulse [91] projects, and archived it in SUMMON.

Besides filtering and archiving datasets of smart city projects, it would be beneficial to reuse data from social networking sites and from such dynamically refreshed websites that provide visualized information of crowd-sourced smart systems that do not share their sensed data. To gather data from such services, I proposed a web crawling approach to facilitate open IoT data archiving and reuse. This proposal extends the previous solutions, and the designed crawling service can provide additional datasets to perform various experiments with the semi-simulation framework.

I participated in the development of the crawling service, for which we used Scrapy, which is an open-source python-based framework for data mining, and Scrapyd, which is an application for managing multiple processes of the spiders' requests in a parallel way. We realized the service as a container-based Docker application. A prototype instance is deployed and available in the Google Cloud, which is connected to a running SUMMON prototype through a REST API to evaluate their interaction. In this way the data gathered by our crawler service can be regularly retrieved and stored in SUMMON. I also participated in the evaluation of its utilization with three smart city use cases. From the results we could see that the web page layouts of the corresponding smart systems varied highly, therefore no generic spider could be used to crawl them.

Conclusions

In the current IoT world, developers are facing many challenges: they need to purchase smart devices, to design and build-up a network of these components, to develop IoT applications, to test and evaluate these applications on the created system and finally, to fine-tune the applications based on the evaluation results. In this work I provided means to overcome some of these challenges. I developed a complex, semi-simulation environment composed the following tools: an Android-based, mobile device simulator called MobIoTSim capable of simulating up to hundreds of IoT devices, a customizable cloud gateway service that is able to manage the simulated devices, and an IoT trace archive service called SUMMON, which can be used to gather real-world sensor data to be fed to the simulation experiments. I exemplified the usage of this complex simulation environment, and evaluated its device management scalability and responsiveness. I do believe that these tools can contribute to a more cost efficient IoT system planning, development and evaluation, even for different academic research communities and industrial stakeholders.

The scientific results presented in this dissertation have been published in numerous journals, conference and workshop papers and have been presented in various scientific forums. These publications have inspired further research, generated collaborations, and are well represented by many independent citations. Most of the research I performed in the past five years has resulted from active involvement in the Internet of Living Things (GINOP-2.3.2-15-2016-00037), Smart Systems Research Institute (20391-3/2018/FEKUSTRAT and TUDFO/47138-1/2019-ITM), EFOP-3.6.1-16-2016-00008, and OTKA FK 131793 national, and EU COST Action IC1304 (ACROSS) and EU COST Action CA17136 (INDAIRPOLLNET) international projects.

Magyar nyelvű összefoglaló

B.1. Bevezetés

Az információ- és kommunikációtechnológia fejlődésének egyik legújabb területe a Dologok Internete (Internet of Things, IoT), melyet hatalmas mennyiségű okoseszköz interneten való megjelenése jellemez. Gartner [124] előrejelzések szerint 2020-ban 30 milliárd eszköz állandóan, 200 milliárd eszköz pedig időszakosan fog az internetre csatlakozni. A Huawei becslése szerint 100 milliárd új eszköz felkapcsolódására lehet számítani 2025-ig. Erre az új trendre reagálva, sok felhő szolgáltató kezdett el támogatást nyújtani az IoT rendszerek kezelésére. Az eddigi tapasztalatok alapján bizonyosságot nyert, hogy a felhő szolgáltatások képesek kielégíteni az IoT rendszerek elvárásait az adatok feldolgozásával és vizualizációjával kapcsolatban. Gubbi és társai, az IoT rendszereket az elsők között leíró ausztrál kutatócsoport már 2013-ban is hangsúlyozta az IoT rendszerek és a felhők kapcsolatát [105]. Botta és társai pedig bemutatták a két terület együttműködésének főbb tulajdonságait, jellemzőit és a felmerülő problémákat [92], valamint további kutatások ismertették az IoT-felhő rendszerek gyakorlatban történő megvalósítása közben felmerülő számtalan kihívást [122]. Ezek a kutatások is motivációként szolgáltak kutatásaimhoz, mivel felvetették a számtalan protokoll és adatformátum kezelésének nehézségeit.

Az IoT rendszerekkel foglalkozó szakembereknek speciális eszközöket kell beszerezniük, amelyeket csatlakoztatniuk kell a hálózatra és be kell üzemelniük őket. Emellett ki kell választaniuk egy megfelelő infrastruktúra szolgáltatót, mely támogatja a szükséges protollokat és adatformátumokat. Ezekben a kihívásokban próbál segítséget nyújtani az általam kidolgozott hibrid szimulációs rendszer. A rendszer három fő komponensből épül fel, melyek munkám főbb hozzájárulását képviselik: (i) egy MobIoTSim nevű Android-alapú mobil IoT eszközszimulátor, mely valós eszközök viselkedését képes modellezni, (ii) egy testreszabható felhős gateway alkalmazás az IoT eszközök kezeléséhez, az eszközöktől érkező adatok fogadásához, feldolgozásához és megjelenítéséhez; és (iii) egy SUMMON nevű IoT adattároló szolgáltatás, mely valós IoT környezetek adatait gyűjti be és teszi elérhetővé.

A kutatásaim egyik fő motivációját az szolgáltatta, hogy több felhő szolgáltató kez-

dett el IoT-specifikus szolgáltatásokat kínálni, hogy megkönnyítse az IoT-felhő rendszerek fejlesztését, azonban a megvalósítás nehéz olyan esetekben, ahol egyszerre több dolgot kell működtetni és kezelni. Ilyen esetekre lehet példa a különböző okosváros alkalmazások fejlesztése SIGFOX [53] vagy a LoRa [41] technológiák használatával, melyek esetében különösen költséges és időigényes a valós eszközök kitelepítése, hiszen egy bázisállomás beüzemelése több ezer eurós költséggel és hatalmas mennyiségű beállítási teendővel jár. Ezekre a kihívásokra nyújtok megoldást az értekezésemben IoT eszközök szimulációjára alkalmas szolgáltatások tervezésével és fejlesztésével, valamint a szimulált eszközöket valós felhő szolgáltatásokkal való kezelésével.

Ami a jelenlegi megoldásokat illeti, az elosztott rendszerek vizsgálatára szolgáló meglévő szimulátorok egy része általános hálózati szimulátor, míg egy másik része IoT-specifikus szimulátor, melyek az említett problémákkal is foglalkoznak. Miután megvizsgáltam ezeket az IoT szimulátorokat, én is készítettem egy bővebb felmérést, melyben kiemeltem az egyes szimulátorok különbségeit. Arra a következtetésre jutottam, hogy egy olyan szimulátorra lenne szükség, ami mobilis, így különböző helyekről képes szimulációkat végezni.

B.2. Új tudományos eredmények

I. Tézis. Megterveztem és megvalósítottam a MobIoTSim nevű mobil IoT eszközszimulátort, amely felhaználóbarát módon képes IoT eszközöket szimulálni, és a generált adatokat a valós hálózaton továbbítani. Elemeztem a használhatóságát, és kiértékeltem a skálázhatóságát több eszköz egyidejű szimulációjával.

Az általam javasolt mobil IoT eszközszimulátor egy Android operációs rendszerre készített mobilalkalmazás, amely képes több száz IoT eszköz egyidejű szimulálására. A szenzoradatok generálására lehetőség van mesterségesen, a felhasználó által megadott intervallumból véletlenszerű értékek generálásával. Továbbá lehetőség van valós IoT adatokat betölteni és lejátszani a szimuláció során a valósághűbb modellezés érdekében. Az eszközök adatküldési gyakorisága mindkét esetben tetszőlegesen állítható. Az alkalmazás MQTT protokoll használatával küld üzeneteket az Eclipse Paho nyíltforrású programkönyvtár segítségével. Az MQTT protokoll egy IoT-specifikus protokoll [120], mely széles körben támogatott az IoT-felhő alkalmazások körében. Az üzenet tartalma JSON formátumban van kódolva, az IBM IoT Foundation üzenetküldési formátumával kompatibilis módon.

A szimulátor általános használata során először egy felhőhöz kell csatlakoztatni azt, ahová az adatokat szeretnénk küldeni. Ezt követően lesz lehetőségünk létrehozni és beállítani a szimulálni kívánt eszközöket, majd elindítani az eszközök szimulálását. Ezeket a lépéseket az alkalmazás három fő része valósítja meg: a *Felhő beállítások* (Cloud settings), az *Eszközök* (Devices) és az *Eszköz beállítások* (Device settings) nézetek. A szimulált eszközökhöz való szenzoradatok generálása többféleképpen is megtörténhet. Alapértelmezetten egy egyedi eszköz létrehozásakor a felhasználó által megadott intervallumból véletlenszerűen kerül egy érték kiválasztásra (alapértelmezetten a *Véletlenszerű adatgenerálás* (Generate random data) opció van kiválasztva). Az egyszerűség kedvéért a felhasználó használhat eszközsablonokat is, melyeknél előre beállított paraméterek és intervallumok szerepelnek, az eszköz típusának megfelelően. A *Véletlenszerű adatgenerálás* (Generate random data) opció kikapcsolásakor a felhasználó kiválaszthat egy fájlt, melyből szenzoradatok tölthetők be. Ezzel a módszerrel korábbi szimulációk is

újra elvégezhetők. Továbbá ezzel a módszerrel akár valós IoT alkalmazásokból származó adatokat is be lehet tölteni. Az OpenWeatherMap nevű publikus időjárási adatokat nyújtó szolgáltatás példafájlja is használható, mely több város különféle paramétereit tartalmazza, például a hőmérsékletet, páratartalmat, légnyomást és szélesebséget. A fájlbetöltő funkció segítségével a MobIoTSim képes valós eszközök valósághű modellezésére is.

Megvizsgáltam a MobIoTSim alkalmazást a szimulált eszközök skálázhatósága szempontjából. Erre a célra készítettem egy egyszerűsített verziót a MobIoTSim-ből (MQTT-Demo néven), mely csak az eszközkészítéshez szükséges funkciókat tartalmazza, így automatikusan, szkriptek használatával történhet az üzenetgenerálás. Olyan alacsony szintű beállítások is elérhetővé váltak így módon, mint a szimulált eszközök által használt szálak száma. Továbbá ez a megoldás részletes statisztikai információt is gyűjt a szimulációról, és méri az egyes funkciók végrehajtásához szükséges időt. Képes az IBM Cloud felhőbe telepített gatewayre csatlakozni és az MQTT protokoll segítségével üzeneteket küldeni neki. Amikor a kísérleteket végrehajtottam, a szimulált eszközök száma 20-ra volt korlátozva az IBM Cloud platform által. A beállításokban a szimulált eszközök számát, az adatküldés gyakoriságát, a használt szálak számát és az üzenet típusát és tartalmát lehet beállítani. Az üzenet típusa lehet véletlenszerű paramétereket tartalmazó egyszerű JSON objektum, vagy használható az OpenWeatherMap több városról időjárási adatokat tartalmazó archív fájlja, mint forrás. A többi beállítás beégetésre került a kiértékeléshez, ahogyan az MQTT broker elérhetősége is. A vizsgálatok kiértékelése után megbizonyosodtam róla, hogy a véletlenszerű adatgenerálás elenyésző időt igényel, tehát nem zavarja a szimulációt. Az IBM felhő által engedélyezett maximális, 20 eszköz szimulálása esetén 8 szál használatával volt optimális a teljesítmény.

II. Tézis. Megterveztem és megvalósítottam egy általános célú felhő gateway szolgáltatást, amely bármely felhőbe telepíthető, és IoT eszközök kezelésére képes a tőlük származó adatok fogadásával, tárolásával és feldolgozásával vagy megjelenítésével. Validáltam a MobIoTSim-el való együttes használhatóságát, és kiértékeltem a skálázhatóságát.

Miután megvizsgáltam több különböző felhő szolgáltatás IoT megoldásait, feltérképeztem a követelményeit egy könnyen használható és szimulált, valamint valós IoT eszközök kezelésére is alkalmas gatewaynek. Ezek alapján kifejlesztettem egy testreszabható gatewayt, amely szinte bármelyik felhő szolgáltatónál telepíthető, hogy akár több száz különféle eszközt lehessen egyszerre kezelni. Ez az általános gateway egy node.js alkalmazás, amely képes egyszerre több eszköz kezelésére MQTT protokoll segítségével, és egy grafikus felületen jeleníti meg a MobIoTSim-ből érkező szenzoradatokat, ahogy a 3.2. ábra mutatja. A szimulált eszközöktől érkező üzeneteket (JSON formátumban) egy MQTT szerver fogadja. Lehetőség van a szimulált eszközök felé is üzenetet (értesítést) küldeni. A valós idejű diagram rajzolásánál az eszközök csoportosításával egyetlen diagramon is áttekinthető az összes eszköztől küldött adat. Habár az IBM Cloud felhő is mutat némi erőforrás-használati információt, Docker konténerek használatával részletesebb adatok érhetőek el az alkalmazással kapcsolatban. Ecélből egy Docker konténerbe csomagoltam a továbbfejlesztett gatewayt, így növeltem hordozhatóságát.

Megvizsgáltam ennek az általános gatewaynek a skálázhatóságát. Kitelepítettem a Docker konténerbe csomagolt gatewayt az IBM Cloud felhőbe, majd több kísérletet végeztem el. Az első körben a MobIoTSim termosztát sablont használtam, hogy 900 szimulált eszközt vizsgáljak. Az eredmények azt mutatták, hogy az alkalmazást csak kis

mértékben terhelte le a nagy számú eszköz kezelése. A második körben a processzorra külön terhelést helyeztem (egy nagyobb számítási igényű algoritmussal) és megismétltem a méréseket 10, 100, 250 és 400 eszközzel. Az eredmények azt mutatták, hogy a dupla mennyiségű üzenettel (azaz 0.5 másodpercenkénti üzenetküldéssel), a processzor használata is duplájára emelkedett. Az utolsó körben a MobIoTSim időjárás sablonját használtam. Mivel a MobIoTSim képes fájlból beolvasott adatok alapján eszközöket szimulálni, így ezt a funkciót használtam, hogy az OpenWeatherMap adatokat használja. Mind a négy eszközcsoportha újra lefuttattam a tesztek egy másodperces frekvencián. táblázatban láthatók. Az eredmények azt mutatták, hogy az időjárási adatokat tartalmazó nagyobb méretű üzenetek nagyobb processzor terheltséget idéztek elő.

Az okos agrárium is egy gyorsan növekvő terület az okos rendszereken belül. A jelenkori információ- és kommunikációtechnológia fejlődésnek köszönhetően az IoT technológiák segíthetik ezt a területet is, melyet megfizethető fenotipizálásnak (affordable phenotyping) neveznek. Ehhez kapcsolódóan részt vettem az Élő Dolgok Internete (Internet of Living Things) projektben, ahol IoLT Okos Cserép (IoLT Smart Pot) néven megfizethető fenotipizálási platformot készítettünk kis méretű növények számára, amely képes a növények növekedésének távoli megfigyelésére egy átlagos üvegházban. Egy ilyen okos cserepekből álló klaszter kezeléséhez az IoLT Okos Cserép Gateway (IoLT Smart Pot Gateway) alkalmazást terveztem meg, és vettem részt kifejlesztésében az általános gateway fejlesztési eredményeim és tapasztalataim alapján. Ezt követően részt vettem a megoldás szimulációs skálázhatósági vizsgálatainak elvégzésében, és megmutattam helyes működését valós környezetben.

III. Tézis. Megterveztem és megvalósítottam a SUMMON nevű IoT adattároló szolgáltatást, amely képes valós IoT alkalmazásokból rendszeresen adatot gyűjteni és tárolni, akár az ezt közvetlenül nem támogató alkalmazásoktól is. Megmutattam hatékony működését három használati eseten keresztül. Az összegyűjtött adathalmazokat elérhetővé tettem a SUMMON szolgáltatás keretében, melyek valósághűbb IoT szimulációkat tesznek lehetővé.

Az Európai Bizottság 2016-ban kiadott egy útmutató dokumentumot [99] melyben a Horizon 2020 projektek által generált kutatási adatok nyílt hozzáférhetőségét és digitális újrahasználatosságát ösztönözte. Ebben a dokumentumban definiálták az úgynevezett FAIR adatkezelési irányelveket, hogy segítsék a Horizon 2020 résztvevőket a kutatási adatok megtalálásában, hozzáférésében és újrahasznosításában. Ezt az útmutatót követve tűztem ki célul, hogy egy nyílt IoT adattároló szolgáltatást készítek, mely az adatok közti keresést, hozzáférést és újrahasználatosságot segíti elő IoT-felhőkkel kapcsolatos szimulációs vizsgálatokhoz.

A MobIoTSim első verzióiban csak mesterségesen generált értékek voltak használhatóak a szenzor szimulációk során. Később a fájlbetöltés funkcióval bővítettem az alkalmazást, hogy könnyebben konfigurálható vizsgálatokat lehessen végezni és valós IoT adatokat is fel lehessen használni a szimulációkhoz. Létrehoztam egy időjárási sablont, amely okos városok időjárásfigyelő eszközeit reprezentálja, amihez az OpenWeatherMap szolgáltatás adatait használtam. A további alkalmazási területek támogatásához egy nyílt IoT adattároló szolgáltatást terveztem használni a MobIoTSim-mel együtt. Ezért fejlesztettem ki a SUMMON (Smart system Usage data Management and MONitoring) nevű megoldást, ami egy adatok összegyűjtésére, szűrésére és tárolására alkalmas szolgáltatás. A szolgáltatásban kilistázhatóak az elérhető adathalmazok, melyek lekérdezés és szűrés után a MobIoTSim szimulátorba tölthetőek nagyméretű vizsgálatokhoz. Több

okos város valós adatait gyűjtöttem össze, így például a SmartME [93] és a CityPulse [91] projektekből.

Az okos városok adatainak összegyűjtése és szűrése mellett, olyan közösségi módon működő szolgáltatásoktól is fontos lehet adatokat kinyerni a szimulációs környezetek számára, melyek dinamikusan frissülő weboldallal rendelkeznek és közvetlenül nem osztják meg a begyűjtött és megjelenített adatokat. Az ilyen weboldalakból történő adatok kinyeréséhez webes feltérképezés (web crawling) módszerét javasoltam az IoT adatok kigyűjtéséhez és újrahasználásához. Ez a javaslat kiegészíti a korábbi megoldást, és a webes feltérképező szolgáltatás segítségével további adathalmazokkal bővíti a SUMMON adattár szolgáltatást.

Részt vettem a webes feltérképező szolgáltatás fejlesztésében, mely során a Scrapy nevű nyílt forráskódú python-alapú keretrendszert használtuk az adatok kinyerésére és a Scrapyd nevű alkalmazást a webes feltérképező pókok (spider) több processzben, párhuzamosan való kezelésére. A szolgáltatás prototípusát egy Docker konténerbe csomagoltuk, majd a Google felhőbe telepítettük és a SUMMON prototípusával kötöttük össze egy REST API segítségével, hogy megvizsgáljuk az együttműködést. Így a webes feltérképező szolgáltatás által kinyert adatok rendszeresen lekérhetők és tárolhatók a SUMMON-ban. Továbbá részt vettem a használhatósági vizsgálatának három okos város esetében is. Az eredmények azt mutatták, hogy az egyes okos rendszerek weboldalainak elrendezése nagyon eltérő, így egy általános pók nem tudná feltérképezni őket. Részt vettem a javasolt webes feltérképező szolgáltatás kifejlesztésében, mely során a Scrapy nevű nyílt forráskódú python-alapú keretrendszert használtuk az adatok kinyerésére és a Scrapyd nevű alkalmazást a webes feltérképező pókok (spider) párhuzamos kezelésére. A szolgáltatás prototípusát egy Docker konténerbe csomagoltuk, majd a Google felhőbe telepítettük és a SUMMON prototípusával kötöttük össze egy REST API segítségével, hogy megvizsgáljuk az együttműködésüket. A megoldás kifejlesztése után részt vettem három okos város alkalmazás feltérképezésének használhatósági vizsgálatában. Az eredmények azt mutatták, hogy az egyes okos rendszerek weboldalainak elrendezése nagyon eltérő lehet, így egy általánosított pók (azaz feltérképező algoritmus) nem tudná egységesen elemezni őket (tehát egyedi megoldásokra van szükség).

Összefoglalás

Az IoT rendszerekkel foglalkozó szakemberek több kihívással is küzdenek: okoseszközöket kell vásárolniuk, meg kell tervezni és meg kell valósítani az azokat kezelő infrastruktúrát, IoT alkalmazásokat kell fejleszteniük, tesztelniük kell ezeket az alkalmazásokat, ki kell értékelniük, végül finomhangolni kell azokat. Ebben az értekezésben olyan eszközöket dolgoztam ki, melyek az említett kihívások közül néhányat segítenek leküzdeni. Kifejlesztettem egy komplex, hibrid szimulációs környezetet, amely a következő komponensekből áll: egy MobIoTSim nevű Android-alapú mobil eszközzsimulátor, amely képes több száz IoT eszköz egyidejű szimulációjára; egy testreszabható felhő gateway szolgáltatás, amely képes kezelni a szimulált IoT eszközöket; és egy SUMMON nevű IoT adattároló szolgáltatás, amely valós IoT alkalmazásoktól gyűjt adatokat, hogy ezzel valósághűbb szimulációt segítsen elő. Bemutattam a használatát ennek a komplex szimulációs környezetnek, és kiértékeltem a skálázhatóságát. Meggyőződésem, hogy ezek az eszközök hozzájárulnak az IoT rendszerek költséghatékonyabb tervezéséhez, fejlesztéséhez és kiértékeléséhez, mind akadémiai kutatócsoportok, mind ipari szereplők számára.

A tézisek tudományos eredményeit számos nemzetközi folyóiratban, konferencia és workshop cikkben publikáltam, és különféle tudományos fórumokon adtam elő. A disszertáció publikációi több későbbi kutatás alapjául szolgáltak, tudományos együttműködések hoztak létre, amelyet a sok független hivatkozás is mutat. Az értekezésben bemutatott kutatási eredmények nagy része az Internet of Living Things (GINOP-2.3.2-15-2016-00037), a Smart Systems Research Institute (20391-3/2018/FEKUSTRAT és TUDFO/47138-1/2019-ITM), az EFOP-3.6.1-16-2016-00008, és az OTKA FK 131793 hazai, és az EU COST Action IC1304 (ACROSS) és EU COST Action CA17136 (IN-DAIRPOLLNET) nemzetközi projekteken való aktív részvétel során keletkeztek.

*“God, grant me the serenity
to accept the things I cannot change,
courage to change the things I can,
and wisdom to know the difference.”*

— Reinhold Niebuhr

Acknowledgement

Over the past five years I had the opportunity to be part of a research team that started to build up a competence in combining Cloud and IoT technologies and providing simulation tools for investigating such ecosystems. This means that I achieved the results reported in this thesis not as an individual researcher, but as a team member. I am grateful to my colleagues for providing these collaboration opportunities.

I would like to thank my supervisor, Dr. Attila Kertész, for the guidance, the helpful pieces of advice, the time and the energy he spent on me. I am grateful that he started me on my research career and that we always had great experiences on shared professional trips.

I would also like to thank Prof. Dr. Tibor Gyimóthy, the former head of the Software Engineering Department, and Dr. Rudolf Ferenc, the current head of the Software Engineering Department, for helping me advance in my PhD studies. Many thanks to my coauthors for giving me a lot of great ideas and helping me overcome the technical challenges, and to Edit Szűcs for proofreading the English version of my dissertation.

I am grateful to my colleagues, especially András Márkus, Dr. Zoltán Tóth, Dr. Dénes Bán and Dr. István Kádár for I could count on them.

Last but not least, I would like to thank my parents for their care and encouragement. I feel very fortunate that my family members are setting a good example for me: my mother with reliability, my father with tirelessness and wide range of interests, my grandmothers with aptitude and perseverance, my grandfathers with dedication and cheerfulness, my uncle with creative thinking, my godmother with kindness, my godfather with entrepreneurial spirit. Thank you Kata, from the bottom of my heart, for being by my side in the moments of struggle, for being my partner in the wonderful experiences, but most of all, for being with me in simple everyday life. I am grateful to Dóri and Dani for being like my real siblings. Thank you to my friends for the cheerful moments, love and understanding.

Research reported in this thesis was supported by the Hungarian Government and the European Regional Development Fund under the grant number GINOP-2.3.2-15-2016-00037 (Internet of Living Things).

Bibliography

- [1] AquamatiX. <http://www.aquamatix.net/>, 2016. Accessed: 2016-12-10.
- [2] Assessment of Landslides using Acoustic Real-time Monitoring Systems (ALARMS). www.bgs.ac.uk/research/tomography/alarms.html, 2016. Accessed: 2016-12-10.
- [3] Bigbelly. <http://bigbelly.com/solutions/stations/>, 2016. Accessed: 2016-12-10.
- [4] Caterpillar. http://www.cat.com/en_US.html, 2016. Accessed: 2016-12-10.
- [5] CleanGrow. <http://www.cleangrow.com/>, 2016. Accessed: 2016-12-10.
- [6] CloudFoundry. <http://cloudinary.com/>, 2016. Accessed: 2016-04-10.
- [7] Echelon. <http://www.echelon.com/applications/pl-rf-outdoor-lighting>, 2016. Accessed: 2016-12-10.
- [8] GE Evolution Series Tier 4 Locomotive. <http://www.getransportation.com/locomotives>, 2016. Accessed: 2016-12-10.
- [9] Govtech. <http://www.govtech.com/fs/Internet-of-Things-Helps-Cities-Manage-Water.html>, 2016. Accessed: 2016-12-10.
- [10] Green Plug. http://www.crocus-technology.com/crocus_applications_sensor-internet-of-things.php, 2016. Accessed: 2016-12-06.
- [11] i2O. <http://en.i2owater.com/>, 2016. Accessed: 2016-12-10.
- [12] Key Finder Tags. <http://postscapes.com/wireless-key-locators>, 2016. Accessed: 2016-12-30.
- [13] Mimo. <http://mimobaby.com/>, 2016. Accessed: 2016-12-30.
- [14] Nest Thermostat. <https://nest.com/works-with-nest/>, 2016. Accessed: 2016-12-30.
- [15] Open Source Lion Tracking Collars. "<http://home.groundlab.cc/lion-collars.html>", 2016. Accessed: 2016-12-10.
- [16] Optoi Chemical Sensors. "<http://www.optoi.com/en/products/details/chemical-physical-sensors-mems>", 2016. Accessed: 2016-12-10.

- [17] Phenonet Project. <http://www.csiro.au/en/Research/D-61/Areas/Robotics-and-autonomous-systems/Internet-of-Things/Phenonet>, 2016. Accessed: 2016-12-10.
- [18] Samsung refrigerator. <http://www.samsung.com/us/explore/family-hub-refrigerator/>, 2016. Accessed: 2016-12-10.
- [19] Smart Outlets. <http://www.postscapes.com/smart-outlets/>, 2016. Accessed: 2016-12-02.
- [20] Smart Parking Sensor. <http://www.save9.com/home/products-and-services/internet-and-wireless-networks/wireless-sensor-networks/>, 2016. Accessed: 2016-12-10.
- [21] SmartPile. <http://smart-structures.com/technology/EDC-embedded-data-collector/>, 2016. Accessed: 2016-12-10.
- [22] Vitality GlowPack. <http://www.vitality.net/products.html>, 2016. Accessed: 2016-12-30.
- [23] Wireless Plant Sensors. <http://postscapes.com/wireless-plant-sensors>, 2016. Accessed: 2016-12-10.
- [24] ActiveMQ source code. <http://activemq.apache.org/source.html>, 2017. Accessed: 2017-11-15.
- [25] ActiveMQ website. <http://activemq.apache.org>, 2017. Accessed: 2017-11-12.
- [26] Amazon Web Services. <http://aws.amazon.com/>, 2017. Accessed: 2017-03-02.
- [27] Azure IoT Hub. <https://azure.microsoft.com/en-us/services/iot-hub/>, 2017. Accessed: 2017-03-12.
- [28] Beecham Research, Smart Farming Sales Brochure. Online: <http://www.beechamresearch.com>, 2017. Accessed: 2017-03-19.
- [29] CityPulse EU FP7 Project. <http://www.ict-citypulse.eu>, 2017. Accessed: 2017-12-05.
- [30] CityPulse Project Final Report. "http://www.ict-citypulse.eu/page/sites/default/files/citypulse_d1.4_final_report_final.pdf", 2017. Accessed: 2017-12-07.
- [31] CKAN website. <https://ckan.org/>, 2017. Accessed: 2017-12-01.
- [32] CloudFoundry. <http://www.kinvey.com/>, 2017. Accessed: 2017-03-19.
- [33] Dreamfactory. <http://www.dreamfactory.com/>, 2017. Accessed: 2017-03-19.
- [34] Google Cloud Platform. <https://cloud.google.com/solutions/iot/>, 2017. Accessed: 2017-03-02.
- [35] Heroku. <https://www.heroku.com/>, 2017. Accessed: 2017-03-12.

-
- [36] IBM Bluemix website. <https://www.ibm.com/cloud/>, 2017. Accessed: 2017-03-02.
 - [37] IBM IoT Foundation message format. <https://docs.internetofthings.ibmcloud.com/>, 2017. Accessed: 2017-03-19.
 - [38] IBM Watson IoT Platform visualization application. <https://github.com/ibm-watson-iot/rickshaw4iot>, 2017. Accessed: 2017-03-30.
 - [39] JoramMQ source code. <http://joram.ow2.org>, 2017. Accessed: 2017-11-13.
 - [40] JoramMQ website. <http://www.scalagent.com/en/jorammq-33/products/overview>, 2017. Accessed: 2017-11-13.
 - [41] LoRa Technology. <https://www.lora-alliance.org/What-Is-LoRa/Technology>, 2017. Accessed: 2017-03-20.
 - [42] Moquette source code. <https://github.com/andsel/moquette>, 2017. Accessed: 2017-11-25.
 - [43] Moquette website. <http://andsel.github.io/moquette/>, 2017. Accessed: 2017-04-20.
 - [44] Mosquitto source code. <https://github.com/eclipse/mosquitto>, 2017. Accessed: 2017-11-17.
 - [45] Mosquitto website. <https://mosquitto.org/>, 2017. Accessed: 2017-11-17.
 - [46] OpenWeatherMap. <http://www.openweathermap.org>, 2017. Accessed: 2017-03-19.
 - [47] Parse. <https://parse.com/products/iot>, 2017. Accessed: 2017-01-02.
 - [48] Private Gateway Service for MobIoTSim. <https://github.com/sed-szeged/MobIoTSimBluemixGateway>, 2017. Accessed: 2017-04-14.
 - [49] Private Gateway Service for MobIoTSim. <https://github.com/sed-szeged/MobIoTSimBluemixGateway>, 2017. Accessed: 2018-06-17.
 - [50] QualNet communications simulation platform. <http://web.scalable-networks.com/content/qualnet>, 2017. Accessed: 2017-03-08.
 - [51] RabbitMQ source code. <https://network.pivotal.io/products/pivotal-rabbitmq>, 2017. Accessed: 2017-11-14.
 - [52] RabbitMQ website. <https://www.rabbitmq.com/>, 2017. Accessed: 2017-11-10.
 - [53] SIGFOX. <http://www.sigfox.com/en/#!/connected-world>, 2017. Accessed: 2017-03-21.
 - [54] SmartME Project datasets. <http://smartme.unime.it/dataset>, 2017. Accessed: 2017-12-10.
 - [55] SmartME Project website. <http://smartme.unime.it/>, 2017. Accessed: 2017-12-12.

- [56] VerneMQ source code. <https://github.com/erlio/vernemq>, 2017. Accessed: 2017-04-30.
- [57] VerneMQ website. <http://vernemq.com>, 2017. Accessed: 2017-11-12.
- [58] Atomiton IoT Simulator. <http://atomiton.com/simulator.html>, 2018. Accessed: 2018-06-20.
- [59] COSMOS website. <http://cosmos.surrey.ca/external/>, 2018. Accessed: 2018-02-08.
- [60] Open Data documents of the city of Surrey. <http://data.surrey.ca/dataset/open-data-documents>, 2018. Accessed: 2018-02-08.
- [61] Open Data Program of the city of Surrey. <http://data.surrey.ca/pages/about>, 2018. Accessed: 2018-02-08.
- [62] AeroGarden. "<https://www.aerogarden.com>", 2019. Accessed: 2019-06-10.
- [63] chroPath tool. <https://autonomiq.io/chropath/>, 2019. Accessed: 2019-05-12.
- [64] CitySens. <https://www.citysens.com>, 2019. Accessed: 2019-06-20.
- [65] Click & Grow. <https://www.clickandgrow.com>, 2019. Accessed: 2019-06-22.
- [66] Docker Container Environment. <https://www.docker.com/>, 2019. Accessed: 2019-06-26.
- [67] GAIAA sPlant smart pot. <http://www.splant.com.cn>, 2019. Accessed: 2019-06-15.
- [68] GitHub repo of SUMMON. <https://github.com/sed-szeged/SUMMON/>, 2019. Accessed: 2019-05-05.
- [69] IoT Smart Pot Gateway on GitHub. <https://github.com/sed-inf-u-szeged/IoLT-Smart-Pot-Gateway>, 2019. Accessed: 2019-06-28.
- [70] Kaa project website. <https://www.kaaproject.org/documentation/>, 2019. Accessed: 2019-01-26.
- [71] LeGrow. <https://www.legrow.co/>, 2019. Accessed: 2019-06-20.
- [72] Lua. <https://www.indiegogo.com/projects/lua-the-smart-planter-with-feelings/>, 2019. Accessed: 2019-06-20.
- [73] MongoDB website. <https://www.mongodb.com/what-is-mongodb>, 2019. Accessed: 2019-06-30.
- [74] MTA Cloud service. <https://sztaki.cloud.mta.hu>, 2019. Accessed: 2019-06-29.
- [75] Odyseed. <https://www.odyseed.com>, 2019. Accessed: 2019-06-20.
- [76] P. Sharma. Xiaomi Smart Flower Pot Flora Review. <https://www.parrot.com/global/connected-garden/parrot-pot>, 2019. Accessed: 2019-06-15.

-
- [77] ParkMe project. <https://www.parkme.com/>, 2019. Accessed: 2019-05-12.
 - [78] Parrot Pot. <https://www.parrot.com/global/connected-garden/parrot-pot>, 2019. Accessed: 2019-06-20.
 - [79] PlantRay. "<https://www.plantray.com>", 2019. Accessed: 2019-06-10.
 - [80] SmartPot. <https://www.smart-pot.net/>, 2019. Accessed: 2019-06-21.
 - [81] The AA Traffic News website. <https://www.theaa.com/route-planner/traffic-news>, 2019. Accessed: 2019-07-10.
 - [82] The Scrapy tool. <https://scrapy.org/>, 2019. Accessed: 2019-05-12.
 - [83] The Scrapy tool. <https://github.com/scrapy/scrapyd>, 2019. Accessed: 2019-05-13.
 - [84] The Selenium tool. <https://www.seleniumhq.org/>, 2019. Accessed: 2019-05-10.
 - [85] The website of Idokep.hu. <http://www.idokep.hu>, 2019. Accessed: 2019-06-08.
 - [86] TOKQI. <https://www.tokqismartflowerpot.com>, 2019. Accessed: 2019-06-20.
 - [87] Tregren. <https://www.tregren.com/>, 2019. Accessed: 2019-06-19.
 - [88] Vincross HEXA. <https://mymodernmet.com/little-robot-smart-planter/>, 2019. Accessed: 2019-06-25.
 - [89] GINOP Internet of Living Things (IoLT) Project. "<https://www.sed.inf.u-szeged.hu/iolt>", 2020. Accessed: 2020-10-03.
 - [90] Huawei GIV 2025. "https://www.huawei.com/minisite/giv/Files/whitepaper_en_2018.pdf", 2020. Accessed: 2020-10-03.
 - [91] Stefan Bischof, Athanasios Karapantelakis, Cosmin-Septimiu Nechifor, Amit P Sheth, Alessandra Mileo, and Payam Barnaghi. Semantic modelling of smart city data. 2014.
 - [92] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, 56:684–700, 2016.
 - [93] Dario Bruneo, Salvatore Distefano, Francesco Longo, Giovanni Merlino, and Antonio Puliafito. Turning messina into a smart city: The# smartme experience. In *Smart Cities in the Mediterranean*, pages 135–152. Springer, 2017.
 - [94] Wojciech Burakowski, Andrzej Beben, Hans van den Berg, Joost W Bosman, Gerhard Hasslinger, Attila Kertesz, Steven Latre, Rob van der Mei, Tamas Pflanzner, Patrick Gwydion Poullie, et al. Traffic management for cloud federation. In *Autonomous Control for a Reliable Internet of Services*, pages 269–312. Springer, Cham, 2018.

- [95] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.
- [96] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.
- [97] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 2017.
- [98] EU CityPulse. Fp7 project. *Online: <http://www.ict-citypulse.eu>. Accessed in December*, 2017.
- [99] European Commission. Guidelines on FAIR Data Management in Horizon 2020. Version 3.0, July 2016. http://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/hi/oa_pilot/-h2020-hi-oa-data-mgt_en.pdf. Accessed: 2018-08-20.
- [100] Angelika Czedik-Eysenberg, Sebastian Seitner, Ulrich Güldener, Stefanie Koemeda, Jakub Jez, Martin Colombini, and Armin Djamei. The ‘phenobox’, a flexible, automated, open-source plant phenotyping solution. *New Phytologist*, 219(2):808–823, 2018.
- [101] Rahul Dagar, Subhranil Som, and Sunil Kumar Khatri. Smart farming–iot in agriculture. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 1052–1056. IEEE, 2018.
- [102] Sandor Guba Imre Szeberenyi Daniel Bach, Eva Geist. Cloud infrastructure for research computing and laboratory environment. In *3rd International IBM Cloud Academy Conference (ICACON 2015)*, 2015.
- [103] Amir Vahid Dastjerdi and Rajkumar Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116, 2016.
- [104] Dror G Feitelson, Dan Tsafirir, and David Krakov. Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982, 2014.
- [105] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.
- [106] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

-
- [107] Son N Han, Gyu Myoung Lee, Noel Crespi, Kyongwoo Heo, Nguyen Van Luong, Mihaela Brut, and Patrick Gatellier. Dpwsim: A simulation toolkit for iot applications using devices profile for web services. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 544–547. IEEE, 2014.
- [108] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick HJ Epema. The grid workloads archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.
- [109] N Jones. Top strategic iot trends and technologies through 2023, gartner report. Online: <https://www.gartner.com/en/documents/3890506>, 2018.
- [110] Byungseok Kang, Daechon Kim, and Hyunseung Choo. Internet of everything: A large-scale autonomic iot gateway. *IEEE Transactions on Multi-Scale Computing Systems*, 3(3):206–214, 2017.
- [111] Gabor Kecskemeti. Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.
- [112] Attila Kertesz. Characterizing cloud federation approaches. In *Cloud Computing*, pages 277–296. Springer, 2014.
- [113] Amin M Khan, Leandro Navarro, Leila Sharifi, and Luís Veiga. Clouds of small things: Provisioning infrastructure-as-a-service from within community networks. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 16–21. IEEE, 2013.
- [114] Guilherme Sperb Machado, David Hausheer, and Burkhard Stiller. Considerations on the interoperability of and between cloud computing standards. In *27th open grid forum (OGF27), G2C-Net workshop: from grid to cloud networks*, 2009.
- [115] J Mahoney and H LeHong. The internet of things is coming, gartner report. Online: <https://www.gartner.com/doc/1799626/internet-things-coming>, 2011.
- [116] Andras Markus, Gabor Kecskemeti, and Attila Kertesz. Flexible representation of iot sensors for cloud simulators. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 199–203. IEEE, 2017.
- [117] Dejan Milojicic, Ignacio M. Llorente, and Ruben S. Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):11–14, March 2011.
- [118] Ioannis A Moschakis and Helen D Karatza. Towards scheduling for internet-of-things applications on clouds: a simulated annealing approach. *Concurrency and Computation: Practice and Experience*, 27(8):1886–1899, 2015.
- [119] Mithun Mukherjee, Rakesh Matam, Lei Shu, Leandros Maglaras, Mohamed Amine Ferrag, Nikumani Choudhury, and Vikas Kumar. Security and privacy in fog computing: Challenges. *IEEE Access*, 5:19293–19304, 2017.
- [120] N. Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, 2017.

- [121] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE international systems engineering symposium (ISSE)*, pages 1–7. IEEE, 2017.
- [122] Stefan Nastic, Sanjin Sehic, Duc-Hung Le, Hong-Linh Truong, and Schahram Dustdar. Provisioning software-defined iot cloud systems. In *2014 international conference on future internet of things and cloud*, pages 288–295. IEEE, 2014.
- [123] Matthew Reynolds and Ulrich Schurr. The 4th international plant phenotyping symposium. *Plant science: an international journal of experimental plant biology*, 282:1, 2019.
- [124] B Rossi. Gartner’s internet of things predictions. *Information Age, Vitesse Media, January*, 2015.
- [125] Ivanovitch Silva, Rafael Leandro, Daniel Macedo, and Luiz Affonso Guedes. A dependability evaluation tool for the internet of things. *Computers & Electrical Engineering*, 39(7):2005–2018, 2013.
- [126] SimpleSoft SimpleIoTSimulator. Online: <http://www.smplsft.com/simpleiot/simulator.html>. Accessed Dec, 2017.
- [127] Boson NetSim Network Simulator. Online: <http://www.boson.com/netsim-cisco-network-simulator>. Accessed in March, 2017.
- [128] Stelios Sotiriadis, Nik Bessis, Nick Antonopoulos, and Ashiq Anjum. Simic: Designing a new inter-cloud simulation platform for integrating large-scale resource management. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 90–97. IEEE, 2013.
- [129] Stelios Sotiriadis, Nik Bessis, Eleana Asimakopoulou, and Navonil Mustafee. Towards simulating the internet of things. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 444–448. IEEE, 2014.
- [130] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36, 2010.
- [131] Simon J.E. Taylor, Tamas Kiss, Anastasia Anagnostou, Gabor Terstyanszky, Peter Kacsuk, Joris Costes, and Nicola Fantini. The cloudsme simulation platform and its applications: A generic multi-cloud platform for developing and executing commercial cloud-based simulations. *Future Generation Computer Systems*, 88:524 – 539, 2018.
- [132] Stig Telfer. The crossroads of cloud and hpc: Openstack for scientific research. *OpenStack consortium*, 2016.
- [133] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. Institute for Computer Sciences, Social Informatics, 2008.

Publications

- [P1] Attila Kertész and Tamas Pflanzner. Towards enabling scientific workflows for the future internet of things. In *Internet of Things. IoT Infrastructures. Second International Summit, IoT 360° 2015, Rome, Italy, October 27-29, 2015. Revised Selected Papers, Part I*, 1 2017.
- [P2] Attila Kertész, Tamas Pflanzner, and Tibor Gyimothy. A mobile iot device simulator for iot-fog-cloud systems. *Journal of Grid Computing*, 17(3):529–551, 2019.
- [P3] Tamas Pflanzner, Zoltan Feher, and Attila Kertész. A crawling approach to facilitate open iot data archiving and reuse. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 235–242. IEEE, 2019.
- [P4] Tamas Pflanzner, Marta Fidrich, and Attila Kertész. Simulating sensor devices for experimenting with iot cloud systems. In *Connected Environments for the Internet of Things*, pages 105–126. Springer, 2017.
- [P5] Tamas Pflanzner, Miklos Hovari, Imre Vass, and Attila Kertész. Designing an iot-cloud gateway for the internet of living things. In *International Conference on Cloud Computing and Services Science*, pages 23–41. Springer, 2019.
- [P6] Tamas Pflanzner and Attila Kertész. A survey of iot cloud providers. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 730–735. IEEE, 2016.
- [P7] Tamas Pflanzner and Attila Kertész. A taxonomy and survey of iot cloud applications. *EAI Endorsed Transactions on Internet of Things*, 3(12), 10 2017.
- [P8] Tamas Pflanzner and Attila Kertész. Analyzing iot, fog and cloud environments using real sensor data. In *Fog Computing*, pages 83–105. Springer, Cham, 2018.
- [P9] Tamas Pflanzner and Attila Kertész. A private gateway for investigating iot data management. In Víctor Méndez Muñoz, Donald Ferguson, Markus Helfert, and Claus Pahl, editors, *Proceedings of the 8th International Conference on Cloud Computing and Services Science, CLOSER 2018, Funchal, Madeira, Portugal, March 19-21, 2018*, pages 526–532. SciTePress, 2018.
- [P10] Tamas Pflanzner, Attila Kertész, Bart Spinnewyn, and Steven Latre. Mobiot-sim: towards a mobile iot device simulator. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 21–27. IEEE, 2016.

- [P11] Tamas Pflanzner, K Zs Leszko, and Attila Kertesz. Summon: Gathering smart city data to support iot-fog-cloud simulations. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 71–78. IEEE, 2018.
- [P12] Tamas Pflanzner, Roland Tornyai, Akos Zoltan Goracz, and Attila Kertesz. Characterizing paas solutions enabling cloud federations. In *Application Development and Design: Concepts, Methodologies, Tools, and Applications*, pages 1095–1120. IGI Global, 2018.
- [P13] Tamas Pflanzner, Roland Tornyai, and Attila Kertesz. Towards enabling clouds for iot: Interoperable data management approaches by multi-clouds. In *Connectivity Frameworks for Smart Devices*, pages 187–207. Springer, 2016.
- [P14] Roland Tornyai, Tamas Pflanzner, Attila Kertesz, Anita Schmidt, and Balazs Gibizer. Performance analysis of an openstack private cloud. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2, CLOSER 2016*, page 282–289, Setubal, PRT, 2016. SCITEPRESS - Science and Technology Publications, Lda.