

Monitoring Informed Testing for IoT

Ahmed Abdullah

School of Science

RMIT University

Melbourne, Australia

ahmed.abdullah@rmit.edu.au

Heinz W. Schmidt

School of Science

RMIT University

Melbourne, Australia

heinrich.schmidt@rmit.edu.au

Maria Spichkova

School of Science

RMIT University

Melbourne, Australia

maria.spichkova@rmit.edu.au

Huai Liu

Engineering & Science

Victoria University

Melbourne, Australia

huai.liu@vu.edu.au

Abstract—Internet of Things (IoT) systems continuously collect a large amount of data from heterogeneous “smart objects” through standardised service interfaces. A key challenge is how to use these data and relevant event logs to construct continuously adapted usage profiles and apply them to enhance testing methods, i.e., prioritization of tests for the testing of continuous integration of an IoT system. In addition, these usage profiles provide relevance weightings to analyse architecture and behaviour of the system. Based on the analysis, testing methods can predict specific system locations that are susceptible to error, and therefore suggest where expanded runtime monitoring is necessary. Furthermore, IoT aims to connect billions of “smart devices” over the network. Testing even a small IoT system connecting a few dozens of smart devices would require a network of test Virtual Machines (VMs) possibly spreading across the Fog and the Cloud. In this paper we propose a framework for testing of each IoT layer in a separate VM environment, and discuss potential difficulties with optimal VM allocation.

Index Terms—Software Engineering, Testing, Internet of Things

I. INTRODUCTION

Internet of Things (IoT) aims to connect billions of “smart devices” over the internet. Usefulness of IoT systems is being realised in various application domains such as transportation, healthcare, smart cities, smart cars, smart factories, etc. Depending on the complexity of an IoT system, its architecture may involve several layers of networking, where appropriate IoT middleware support an IoT application. One of common middleware design approaches is Service Oriented Architecture (SOA), cf. [1]. SOA-based IoT middleware is a promising design perspective to overcome heterogeneity of physical devices and their underlying communication technologies with high level standardised service interfaces.

IoT network and middleware design follow a common layered approach. Both from networking and middleware perspective, Fog and Cloud computing are envisioned as natural fit for layered design of IoT systems, cf. [2], [3]. Layered architecture model of IoT middleware allow designing IoT application components, which reside into separate VMs that may spread across Cloud and Fog infrastructure. Cloud infrastructure provides centralized resources and processing at a global scale [4]–[7]. As Cloud computing and storage services are highly scalable and efficient, many applications utilise Cloud for analytics, Big Data processing, etc. It also

enables long running and data-intensive scientific experiments [8]. In the Fog infrastructure, certain services are managed at the edge of the network, at so-called *Fog nodes*. Both Fog and Cloud nodes share a high level of virtualisation. Whereas fewer cloud nodes represent fewer and much larger global resources at very long distances away from IoT devices, Fog nodes are numerous, smaller and more compact and importantly accessible from from IoT devices through local area or regional networks. IoT applications can be handled taking into account their specific requirements, such that low latency response, mobility support, location awareness, etc.

Contributions: We propose a framework for testing each IoT layer in a separate Virtual Machine (VM) environment. As per Bourque and Dupuis [9], software testing consists of the dynamic verification of the software behaviour against the expected behaviour, using a finite set of test cases. The set of test cases has to be selected from the (usually infinite) execution domain, with the goal to fulfil the specified coverage criteria. However, dynamic verification of program behaviour requires observation of its runtime behaviour (i.e., monitoring) thereby allowing measurement of compliance against expected behaviour of the system. Our study is focused on deriving operational profiles from monitoring data that is collected during dynamic verification of behaviour. We aim (1) to use operational profiles for prioritising tests during test selection in regression testing, (2) to predict fault location in IoT services by combining operational profiles and Markov chain usage models, derived from interface behaviours of IoT services. If it is predicted that an IoT might have faults, it would undergo extended runtime verification procedure, accomplished with monitoring. The corresponding specifications for monitors are going to be derived from interface specifications of the standardised IoT services. We call this methodology “monitoring informed testing”.

II. IOT ARCHITECTURE

IoT middleware could be divided into three layers (perception, middleware and application layers), where each of them has specific functionalities, cf. [1].

Perception Layer: Sensor data i.e. context information is gathered from smart physical objects. IoT applications with various application-specific requirements (such as low latency response, mobility support, location awareness, etc.) are handled at this layer. Data is collected at collector nodes from

sensors, then filtered, processed, analysed, and after that the decided actions are performed through actuators. Functionalities like data acquisition, context annotation, device configuration, device management, etc., are provided as services through SOA-based IoT middleware.

Middleware Layer: Fog computing is considered as widely accepted approach that could be set up at this layer to prevent large burst of sensor data from flooding into the Internet. Fog nodes [10] are highly virtualised platforms (e.g., IOx and Cloudlet), which may be a computer like “cloud in a box” providing computation, storage, and networking services. Middleware support aims to provide higher-level services. For example, SOA IoT middleware provide activity reasoning, processing, machine-to-machine-interoperability services [11].

Application Layer: Global coverage for an IoT application can be provided through a Cloud. The Cloud is particularly useful to serve as data storage and also providing processing power needed for IoT big data analytics. Service interfaces, APIs, etc. provided through IoT middleware are utilised to implement data analytics, storage management, application and health monitoring, profiling, etc. Here, we explain the testing set-up of an industrial IoT application and also discuss SOA functionalities of industrial IoT middleware *oneM2M* [12], which is an industrial IoT middleware for machine-to-machine (M2M) interoperability, compliant to the European Telecommunications Standards Institute (ETSI) standards¹.

Device Nodes: We envision of a global industrial automation systems development company (e.g., ABB and Siemens) who has several *Test Fields* in different countries across the world. In each *Test Field* they would have a collection of IoT enabled smart devices for testing purposes. Such a *Test Field* would have a variety of sensors and actuators attached smart machines, embedded devices and computational nodes. Figure 1 illustrates perception layer setup for an industrial IoT application, where each smart machine is connected with a dedicated Device Service Capability Layer (DSCL), i.e., a middleware component provided by oneM2M that collects data from a smart device.

Fog Nodes: We anticipate that *Devices Nodes* that are located at a *Test Field* will directly connect to the *Fog Nodes*, cf. Figure 1. OneM2M’s Network Service Capability Layer (NSCL) is a middleware component that provides higher-level services such as resource access, provisioning, and self-configuration, etc.

Cloud Nodes: The industrial IoT application would implement a dashboard providing data analytics, industrial process monitoring, machine learning, etc. at the Cloud. In oneM2M, nodes consist of at least one common services entity (CSE) or one application entity (AE). An AE defines application logic for an end-to-end M2M solution [13].

III. MONITORING APPROACHES

The monitoring requirement for testing lies in the amount and type of information required for (1) detecting failures

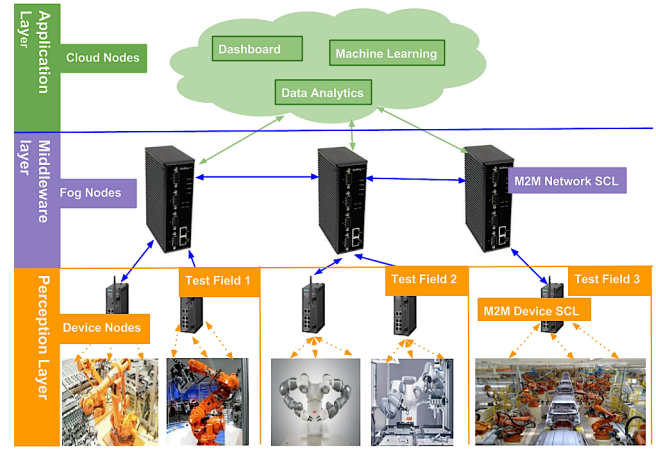


Fig. 1. Industrial IoT Networking Architecture

due to bugs (functional or extra-functional violations) or slugs (performance bottlenecks), (2) locating their root causes or errors, and (3) analysing the error context so as to prepare fixing or preventing them. For testing sequential processes [14], it is usually sufficient to monitor input and output behaviour through predefined service interfaces. The actual information to be monitored can be categorized into three groups:

- *Data flow* to observe the input data flow of a service as well as the output data produced and passed through the service interfaces;
- *Control flow* to observe in what order inputs are received and outputs are produced; and
- *Resource flow* including usage and performance of memory, compute, network and file system resources.

The services need to be instrumented with *probes* to be able to observe their behaviours. Furthermore, these probes need to be left within the system. However, effect of probes in the system need to be measured and compensated by allocating additional resources to the system. However, *in-line-probes*, a kind of probe instrumented at task level (e.g. atomic service level) to yield additional outputs to the task, would be sufficient for our purposes. Our goal is to create operational profiles through analysing the outputs that are produced through the service level in-line-probes instrumented within the system. This approach has to be applied within the development phase. One of the potential issues related with the in-line-probes, is over instrumenting that decreases the system performance as this may impact on Resource Flow monitoring.

Another approach to monitoring is *runtime verification* (RV), where a formal specification of behaviour is instrumented in the system during runtime. RV aims to combine testing with formal methods such that systems behaviour is thoroughly checked against requirement specifications at runtime. In this approach, formal models (e.g., state machines) for service interfaces are typically developed during designing test models. The formal model is derived from requirement

¹<http://www.etsi.org/about>

specification and then integrated with source code. Afterwards, monitors are generated automatically based on the formal models and weaved into the runtime code. An advantage of RV is that formal models can be encoded with varying levels of abstraction thus allowing varying levels of monitoring data to be generated.

IV. STATISTICAL TESTING

In statistical testing, statistical methods are used to determine the reliability of a system to demonstrate a system's fitness against its intended use. In our future work, we are going to focus on a fundamental approach that applies Markov Chain Usage Model (MCUM) in conjunction with this method. The statistical testing process may be started at any point: all test artefacts become valuable assets and may be reused throughout the software system life cycle. Statistical testing involves several steps, such as usage model development, model analysis and validation, tool chain development, test case generation, etc. We are going to focus on the issues on usage model construction as it relates to one of the fundamental concepts of our study.

A. Markov Chain Usage Model

A fundamental statistical artefact for statistical testing is study of population, which requires the characterization and representation of the system. This should include common and typical as well as infrequent and exceptional scenarios of the system at a suitable level of abstraction. One such method of characterisation and representation of system is done through developing operational usage model. An operational usage model characterises the population of usage scenarios described in terms of how it is going to be used in an operational environment. Markov Chain Usage Model (MCUM) characterises usually infinite population, which contains all possible scenarios of a system. However, when a population is too large for exhaustive study, a statistically correct sample must be drawn as a basis for inferences about the population.

MCUM is usually derived from requirements specification, and can be developed in two stages, structural and statistical. The structural stage involves possible use; the statistical stage deals with expected use. Developing the structural model involves identifying sets of states within a system and associated state transitions, which are defined by directed arcs. The statistical stage involves determining transition probabilities in the structure. There are two basic ways of assigning transition probability - one based on direct assignment of probabilities and the other is determining values through analytical method.

Direct assignment of transition probabilities among states in a usage model may involve collecting data from historical or projected usage of an application. Transition probabilities among states yield various usage information of a system for example usage environments, user demographics, classes, or other special usage situations. Moreover, sets of transition probability may change several times as systems mature, based on availability of information, or experience of use. A

probability value for each arc of the model may be determined when extensive field data for systems is collected over long periods of time. However, to determine transition probabilities for new systems, we may follow a manual process, i.e., analyse software requirements documents, review user guides, take customer reviews, etc. In case essential information is not available, all transition states are assigned with uniform probability in the beginning.

B. Operational profile

An operational profile characterizes how a system will be used in production [15]. Therefore, it allows estimating reliability of a software product, helps prioritization of product feature to be developed and tested according to their usage frequency. In the context of IoT systems and software in production, deployed across millions of devices with thousands of different variants of code and personal user preference settings, it is particularly important to reflect the frequency of software deployment in specific device, location and personal contexts. This does not only require monitoring the unconditional probability of executing a piece of code globally, but its conditional probability, given the likelihood of executing in a specific context defined by variants of devices, OS, application software, location and personal preferences etc.

When software is developed, in testing phase or already in deployment, operational profiles may be generated either manually as described earlier or automatically through analysing event logs. The operational profile then may be used to generate random test cases such that random walk through MCUM [16].

C. Continuous Integration and Regression Testing

An IoT application may be layered into the three distinguishable components: *Device Component* in perception layer, *Middleware Component* in networking layer, and *Application Component* in presentation layer. Each of these functional components would undergo automated regression testing as soon as new changes to the code are submitted. Regression testing nowadays is integrated with continuous integration software development practice that aims to ensure program's correctness as soon as new changes to the code is submitted to a mainstream code repository that triggers automated build, and testing. One of the core functionalities of regression testing is prioritizing test cases to achieve some performance goal [17], for example selecting frequently used features for testing early. Executing tests ordered by priority of frequently used feature would allow to achieve higher reliability by discovering and fixing high frequency failures sooner. Hence, we aim to prioritize tests according to the order of frequency of the features derived within an operational profile.

V. FRAMEWORK FOR MONITORING INFORMED TESTING

The aim of the framework is to monitor IoT services and generate operational profile from monitoring information. IoT systems provide various types of services at different layers. Sensor data collected through service interfaces travel

from perception layer, through middleware layer to application layer. Furthermore, the data is formatted and processed at different layers differently. Our goal is to analyse the monitoring information in services as well as in sensor data with a view to design operational profile for an IoT services. We aim to design continuously adapted operational profiles. Thus, we need to investigate in what time intervals the operational profiles are adapted and also that the same event is not recorded multiple times in the contentiously adapted profile.

We are going to prioritize tests based on service usage frequency in operational profile. The monitors will be derived from specifications of IoT services and integrated into relevant IoT services at IoT runtime. Our intention is to apply runtime verification only for the services identified through fault prediction mechanism. Thus, MCUM will be derived the interface specifications that are provided with IoT services for identifying states and state-transitions that are needed for a MCUM. The judgements from operational profiles have to be evaluated against the prediction derived through MCUM. Therefore, we plan to design a high confidence fault prediction mechanism where prediction measured through MCUM would be combined with that gathered from the *operational profile*.

All three IoT architectural layers (i.e., perception, middleware and application layers) will be tested in separate VMs, as illustrated in Figure 2. We define a *Test Server VM* required for Continuous Integration (CI) setup, involving loading source code and tests from repository, doing the builds, deploying the build to appropriate test machines and running regression tests.

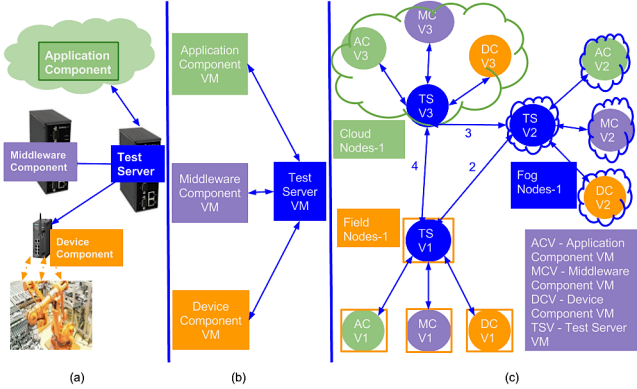


Fig. 2. Proposed Framework: Infrastructure

A global company with multiple test fields would require a large test network setup with test VMs deployed across Fog and Cloud. Figure 2 shows a network of test VMs that forms a weighted graph where weight of an edge is measured according QoS constraints like bandwidth, latency, etc. Thus, we intend design an optimal VM allocation strategy that would allocate VMs in test network considering all these QoS constraints. Such an VM allocation method is needed for various reasons such as lowering the amount of data transfer through high cost edge, thereby enabling faster test execution, lowering networks usage, etc. However, the VM allocation method needs to consider following issues:

- *Data exchange*: A huge amount of data would be exchanged across the test network. For example source code and tests need to be transferred to test servers, and tests have to be sent on testing VMs for execution. An optimal VM allocation method would be required to minimise the amount of data transferred, especially across the high cost edge.
- *Allocation of VMs*: Device Components (DCs) have to be tested at the field network at perception layer. Virtual sensors may be suitable for testing purposes to simulate data into DCs. This would allow us to allocate them either to the Fog or the Cloud. In some cases where testing of virtual sensors or actuators is not feasible, the VM allocation method should find an optimal route keeping DC nodes allocated to a dedicated layer.
- *Quality of Service*: Network Quality of Service (QoS) constraints such as bandwidth, latency and network failure have to be considered. For example, the network connection between *Field Node* and *Fog Node* should have high bandwidth, low latency, and very low network failures. Therefore, the QoS constraints need to be measured as part of our Resource Flow monitoring and as demanded by the optimization method.

A. Operational profile based online test prioritization

Software testing based on users' perspective is known as usage-based testing. Both Markov chains and operational profiles are used to characterise usage models statistically. Many usage based testing methods have been proposed in classical software engineering and also a few in the service oriented architectures. Sammodi et al. [18] applied operational profile for test case prioritization where service based monitoring data is used to derive operational profile. From an IoT perspective, data is generated from heterogeneous smart devices and collected through IoT services. Hence, we believe it is essential to analyse monitoring data with reference to appropriate inline probing mechanism that we discussed previously. Bai et al. [19] proposed an ontology-based method for producing operational profile through analysing SOAP messages. However, they use the operational profile for test generation purposes. On the other hand, we aim to create operational profiles for IoT oriented CoAP or REST services for the purposes of test prioritization.

There are also many approaches on operational profile generation. In the industry, various tools are used for event correlation, e.g., SEC [20]. Nagappan et al. [21] proposed an algorithm for creating operational profiles that calculates frequency of both used and unused features. The procedure begins with identifying the list of functions from the source code that prints out logs. Then a log abstraction method is used, where an integer equivalent of the execution log is created. Afterwards, using that integer equivalent of the execution log an operational profile is generated for each identified function by implementing a suffix array based algorithm that has $\log(N)$ complexity. However, our goal is to produce continuously

adapted operational profile from monitoring data and event logs generated from IoT devices.

B. Usage model based fault prediction

The main objective of statistical testing methods is reliability prediction. Usage models such as Markov chains and operational profiles are built to achieve this objective. Test cases generated from usage modes carry statistical significance and failure of test case (test sequence) is referenced back to usage model for predicting the location of fault. Sammodi et al. [18] cross checked test sequence from Markov chain with input sequence in monitoring information and based on some precision measurement in fault prediction and propose runtime adaptation of services. Metzger et al. [22] applied similar fault prediction mechanism, also suggesting runtime testing of services for confidence measurement. In contrast to these approaches, our goal is to enable runtime verification of service that is predicted to have fault.

C. Virtualized IoT layers and Optimal VM Allocation

Integrating heterogeneous smart objects (e.g. sensors and actuators) to an automated test system would be difficult task. Also setting up a test system and maintaining it with connected smart devices would be challenging. We aim to use virtual sensors [23] to simulate the interactions of smart objects with the physical layer of IoT middleware, thereby enabling the Physical Component of IoT application be tested in a VM. A test network for a global industrial IoT company could be depicted as a weighted graph, cf. Figure 2. A similar scenario where optimal VM allocation problem over distributed Cloud been presented as a weighted graph in [24]. The authors explained this to be a graph slicing (a NP-hard) problem and proposed two approximation algorithms, one for VM allocation within same cloud and another for inter cloud VM allocation. However, we are considering a weighted graph which has multiple dynamic weights, i.e., bandwidth, latency, network failure, etc. and also the weight of an edge would be calculated dynamically.

VI. CONCLUSIONS

This paper introduces the core ideas of an adaptive framework for monitoring informed testing, with an aim to monitor IoT services and generate operational profile from monitoring information. The proposed framework will enhance testing activities by utilizing monitoring data gathered from IoT smart devices and relevant event logs. We suggest

- (1) to use monitoring probes for Data, Control and Resource flow combined;
- (2) to model operational profiles in a context dependent way using conditional probabilities with reflecting the high variation in hardware, software and consumer contexts, and
- (3) to predict fault locations in IoT services by combining operational profiles and Markov chain usage models, derived from interface behaviours of IoT services.

This would enable, e.g., runtime verification of service that is predicted to have fault.

REFERENCES

- [1] M. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: a survey," *IoT Journal*, vol. 3, no. 1, pp. 70–95, 2016.
- [2] M. Yannuzzi, R. Milito, R. Serral-Gracià, D. Montero, and M. Nemirovsky, "Key ingredients in an IoT recipe: Fog computing, cloud computing, and more fog computing," in *CAMAD*. IEEE, 2014, pp. 325–329.
- [3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Mobile cloud computing*. ACM, 2012, pp. 13–16.
- [4] I. Yusuf, I. Thomas, M. Spichkova, S. Androulakis, G. Meyer, D. Drumm, G. Opletal, S. Russo, A. Buckle, and H. Schmidt, "Chimney: Reliable computing and data management platform in the cloud," in *37th International Conference on Software Engineering, Volume 2*, 2015, pp. 677–680.
- [5] M. Spichkova, H. Schmidt, I. Thomas, I. Yusuf, S. Androulakis, and G. Meyer, "Managing usability and reliability aspects in cloud computing," in *ENASE*, 2016, pp. 288–295.
- [6] M. Spichkova, I. Thomas, H. Schmidt, I. Yusuf, D. Drumm, S. Androulakis, G. Opletal, and S. Russo, "Scalable and fault-tolerant cloud computations: Modelling and implementation," in *ICPADS*. IEEE, 2015, pp. 396–404.
- [7] M. Spichkova, H. Schmidt, I. Yusuf, I. Thomas, S. Androulakis, and G. Meyer, "Towards modelling and implementation of reliability and usability features for research-oriented cloud computing platforms," in *Series on Communications in Computer and Information Science*. Springer, 2016, pp. 158–178.
- [8] I. Yusuf, I. Thomas, M. Spichkova, and H. Schmidt, "Chimney: Connecting Scientists to HPC, Cloud and Big Data," *Big Data Research*, vol. 8, pp. 39–49, 2017.
- [9] A. Abran, J. Moore, P. Bourque, R. Dupuis, and L. Tripp, "Software engineering body of knowledge," *IEEE Computer Society*, 2004.
- [10] S. Li, L. Da Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.
- [11] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *Communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2014.
- [12] M. Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira, "Om2m: Extensible etsi-compliant m2m service platform with self-configuration capability," *Procedia Computer Science*, vol. 32, pp. 1079–1086, 2014.
- [13] M. Palattella, M. Dohler, A. Grieco, G. Rizzo, J. Torsner, T. Engel, and L. Ladid, "Internet of things in the 5g era: Enablers, architecture, and business models," *SAC*, vol. 34, no. 3, pp. 510–527, 2016.
- [14] H. Thane, "Monitoring, testing and debugging of distributed real-time systems," Ph.D. dissertation, 2000.
- [15] J. Musa, "Operational profiles in software-reliability engineering," *IEEE Software*, vol. 10, no. 2, pp. 14–32, 1993.
- [16] J. Poore, R. Lin, L. and Eschbach, and T. Bauer, "Automated statistical testing for embedded systems," in *MBTES*. CRC Press, 2011, pp. 111–146.
- [17] X. Zhang, T. Chen, and H. Liu, "An application of adaptive random sequence in test case prioritization," in *SEKE*, 2014, pp. 126–131.
- [18] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl, "Usage-based online testing for proactive adaptation of service-based applications," in *CSAC*. IEEE, 2011, pp. 582–587.
- [19] X. Bai, S. Lee, W. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," in *Web Services*. IEEE, 2008, pp. 465–472.
- [20] R. Vaarandi, "Sec-a lightweight event correlation tool," in *IP Operations and Management*. IEEE, 2002, pp. 111–115.
- [21] M. Nagappan, K. Wu, and M. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *ISSRE*. IEEE, 2009, pp. 41–50.
- [22] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards pro-active adaptation with confidence: augmenting service monitoring with online testing," in *SEAMS*. ACM, 2010, pp. 20–28.
- [23] E. Reetz, "Service testing for the internet of things." Ph.D. dissertation, 2016.
- [24] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *Infocom*. IEEE, 2012, pp. 963–971.