Performance Testing of an Internet of Things Platform

John Esquiagola, Laisa Costa, Pablo Calcina, Geovane Fedrecheski and Marcelo Zuffo
Integrated Systems Laboratory, Sao Paulo University, Sao Paulo, Brazil

Keywords: Internet of Things, Testing, Performance.

Abstract: The Internet of Things (IoT) is a network of physical objects, or things, with embedded electronics, software, sensors, and connectivity. The connection of all these things leverages value generation, by offering new services and strategic information. In order to make the Internet of Things possible, the integration of many technologies is necessary, such as machine-to-machine and cyber-physical systems. The process of testing IoT applications introduces new challenges because it does not only includes typical test strategies and methodologies. Testing an IoT system depends on its the specific configuration, and it also needs to consider the hardware platform and the network environment. Currently, industry and academy efforts are focusing on usability and connectivity tests, such as: simulating the environment where the device is to be used, and ensuring information is exchanged in a secure manner. In this paper, we use the current version of our IoT platform to perform stress testing of our IoT platform under different conditions. Our test methodology for IoT applications is also presented. Three different hardware platforms have been used for performing the stress testing of our platform.

1 INTRODUCTION

In future IoT networks, the increasing number of devices and services will be reflected in a diversity and heterogeneity of hardware, software platforms, network protocols, and service providers. The realization of the IoT paradigm implies many challenges that need to be addressed, including availability, reliability, mobility, performance, scalability, interoperability, security, management, and trust (Al-Fuqaha et al., 2015).

Recent advances in networking technology, such as the IP protocol used in embedded devices has changed the Internet landscape. One of the advantages of the IP technology within embedded devices is the possibility of using a web services architectures, where ”things” are wrapped into web services. This concept is reflected in the Web of Things (WoT) (W3C-Group, 2016) initiative. Web services can be used on top of IP stack in order to reduce the complexity of applications and to improve the software re-usability (Colitti et al., 2011).

Lately, there has been an increasing number of IoT middleware proposals. (Mineraud et al., 2015) present a survey of several efforts from academy and industry; a gap analysis is done too, focusing in the weakness of current solutions, and aiming to improve the development of future IoT platforms.

Testing IoT systems have complications not present in traditional system deployments, such as enterprise web services, due to the heterogeneous and massively distributed nature of its components (Reetz et al., 2013). In order to guarantee the correct functioning of such complex systems, functional tests and performance evaluation must be done before deploying an IoT system in a production environment. In such a configuration, the real interaction with the physical world needs to be observed, differently from common software testing methods (Reetz et al., 2013). The heterogeneous nature of IoT components demands strong testing capabilities to ensure service performance meets the user requirements as well as service level agreements between service providers and consumers. Cognizant et al (Cognizant, 2016) defined the following types of testing that needs to be performed within an IoT ecosystem:

- *Functional testing*, which validates the correct functionality of the IoT application.
- *Connectivity testing*, it is responsible for testing the wireless signal in order to determine what happens in case of weak connection, or when there are many devices trying to communicate.
- *Performance testing*, validates the communication

Esquiagola, J., Costa, L., Calcina, P., Fedrecheski, G. and Zuffo, M.
Performance Testing of an Internet of Things Platform.
DOI: 10.5220/0006304503090314
In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS 2017)*, pages 309-314
ISBN: 978-989-758-245-5

and computation capabilities. Stress testing can be used in order to find how many simultaneous connections can be supported by a specific device.

- *Security testing*, focus in privacy, authorization and authentication features.
- *Compatibility testing*, verifies the correct functionality under different protocols and configurations.
- *Exploratory testing*, also called user experience tests.

There are some industrial reports about IoT testing procedures (Cognizant, 2016)(Bloem, 2016)(RCR-Wireless, 2016), however, little academic work has been found about testing IoT systems, and most of them concentrate in performance evaluation(Lunardi et al., 2015)(Thangavel et al., 2014)(Vandikas and Tsiatsis, 2014), IoT resource emulation, and IoT testbeds deployments (Sanchez et al., 2014)(Adjih et al., 2016).

In this paper, we use the current stage of our IoT platform implementation based on SwarmOS concept(Costa, 2015) in order to present our test methodology that was followed to cover most of the fields in the IoT testing area. Finally we show some testing results of our platform after deploying within single board computing devices. Response time and the network throughput were measured during the tests and some conclusions were obtained according to the computing capabilities of the devices.

The remainder of this paper is organized as follows: Section II presents the related work in the IoT field. Section III highlights the test methodology including the architecture overview of our IoT application. The stress testing results and analysis are presented in Section IV. Section V concludes the paper with final considerations and future work.

2 RELATED WORK

In IoT systems, functional and performance tests must be done before deploying the system in a production environment. In contrast with typical test methods, the deployment of an IoT system need to guarantee a successful interaction with the physical environment (Reetz et al., 2013). The heterogeneity of the components of an IoT system implies the development of new test methods and architectures in order to ensure the performance of the system and to meet user requirements.

COBASEN(Lunardi et al., 2015) is a framework that allows scalable search and transparent usage of computational devices in IoT environments; this framework proposes a search engine which allows the use context information to discover, select and use of device according to user and application requirements. Framework validation was done by implementing a platform and executing functional performance tests. A Java application prototype was developed in order to stimulate the search engine by capturing user needs through a query. Requirements of the test were: (1) query processing time and (2) indexing time duration.

Thangavel et al. (Thangavel et al., 2014) present the design and implementation of a common middleware that supports MQTT and CoAP. Performance evaluation was done experimentally by using the middleware under different network conditions. End-to-end delay and bandwidth consumption were the metrics that were studied for both protocols. Tests are focused in the transportation of the sensor data at the gateway node to the back-end server or broker. Bandwidth usage was defined as a the total data transferred per message. Delay was defined as a difference between the time when a data file was received and the time when the file was published. A small program was implemented to generate the sensor data in order to emulate the reception from sensor nodes. Besides that, open source implementation of MQTT and CoAP were used and integrated to the middleware. Authors conclude that performance of different protocols are dependent on different network conditions, and it can be exploited to improve the network performance by deciding what protocol to use according to the current network conditions.

IoT-Framework is a computational middleware thas was implemented by using open source components (Vandikas and Tsiatsis, 2014). This proposal uses the RabbitMQ publish-suscribe, elasticSearch database, WebMachine REST Api, R statistical software among others. Authors performed the evaluation of the system by using two set of tests, first one creates a total number of 10^5 HTTP POST requests that simulates a different number of users (producers). Second tests were done by using a simple Java client to generate a varying number of consumers in order to see the impact in the system.

Nowadays, IoT testbeds are available in order to evaluate IoT applications in real environments with real-world conditions. That testbeds are already deployed in several sites in Europe and has been used to perform IoT testing and collect results such as consumption or packet ratio. For example, IoT-Lab (Adjih et al., 2016), SmartSantander (Sanchez et al., 2014) and Web of Things TestBed (WoTT) (Belli et al., 2015). Most of academic research about IoT testing are focused on functional tests. There are a

few academic reports covering specific IoT testing fields like connectivity, compatibility and exploratory.

3 IoT TEST METHODOLOGY

3.1 IoT Platform

In this section, we describe our IoT platform, based on the Swarm vision(Rabaey, 2011). The swarm concept expands the IoT vision to a more organic networks where the peers communication, characteristics and topologies can be adapted dynamically. The implementation of the platform follows a framework called SwarmOS(Costa, 2015) which explores the dynamic cooperation and peer-to-peer communication of devices. The architecture of the system is illustrated in Figure 2. A central element in the proposed architecture is the IoT Broker, which acts as a facilitator for the communication and interaction among the heterogeneous devices in the Swarm/IoT. The following modules are defined in current version of our broker:

- *Registry*, a central directory for the services in the device,
- *Discovery*, a service that lets the broker communicate with other brokers to search a requested service;
- *Contracting*, a service that helps in match the parameter for service consumption among requester and provider services;
- *Authorization*, a service that manages the access to services;
- *Semantic mediation*, a service which performs an automatic equivalence between concepts in different service description domains, in order to achieve interoperability;
- *Monitoring and optimizing*, a service based on a continuous monitoring of the network, creates a quantitative model that suggests optimization plans;
- *Binding*, a service that performs an syntactic adaptation of the communication among services,
- *Policy management*, a service that manages the access policies of the authorization module.

The IoT architecture was designed to following the REST principles, so it exposes device functionalities through light web services by HTTP or CoAP protocol. Broker behaves as a facilitator to atend requests from any service in the network. It has a local

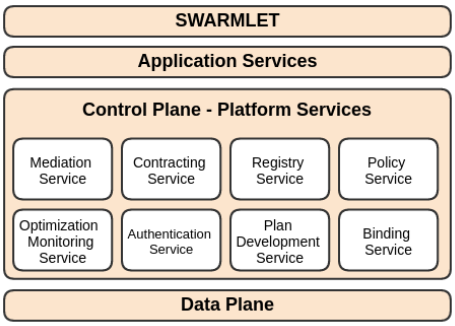

Figure 2: IoT Broker Architecture.

database where stores the descriptions of the local services which it can offer to other brokers or services. Broker functionalities can be exposed through three entries: *registry* and *locate* and *status*. Other capabilities presented in the proposed architecture are under active development.

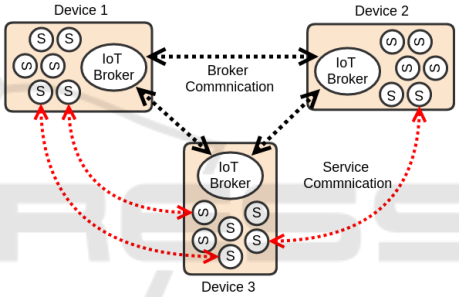

Figure 3: Device communication.

In Figure 3 we depict the deployment of a simple device configuration in order to better visualize the communication using the proposed IoT Broker. Services can be deployed in the same device and broker provides the interface to communicate with services deployed in other devices. A device hosts several services along with one broker. When a service is executed, it first *register*s in its local *IoT Broker*, which can be found for other brokers in the network. Next, when a device needs to use the functionalities of other device, it looks for a service in the network, by using the broker *locate* service.

3.2 Test Methodology

Testing the IoT involves the validation process of various aspects related to network connectivity, such as bandwidth, dropped connections, etc. The challenges of IoT testing are beyond software implementation and hardware devices because IoT adds new complexity parameters to the classic test models (Al-Fuqaha et al., 2015). In our specific IoT platform we have defined several kinds

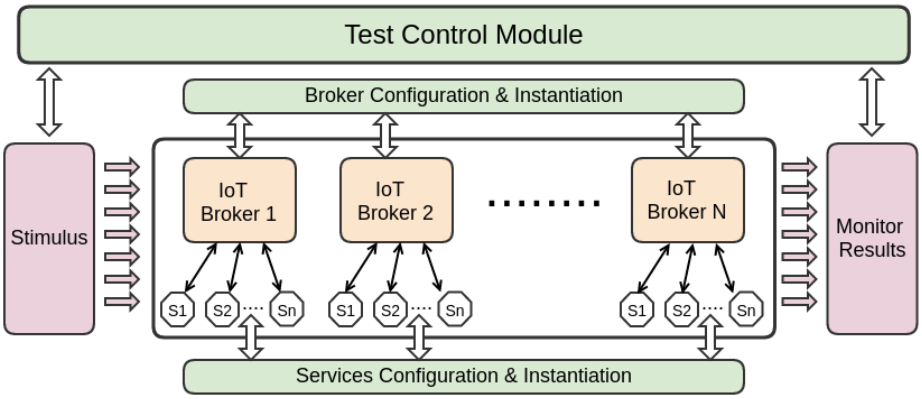

Figure 1: Modular Test Architecture.

of testing phases in order to validate the whole functionality of the system. Then, there are some important layers that are part of an IoT system and needs to be test. The following layers and corresponding test phases were defined for our IoT system:

Software Interaction Layer:

- Unitary tests
- Integration tests
- System tests
- Acceptance tests

Hardware Interaction Layer (Device comm and Network):

- Performance & Conectivity tests
- Security tests
- Interoperability tests

User Interaction Layer:

- Usability tests
- Conformance tests
- Reliability tests
- Scalability tests

4 PERFORMANCE TESTING RESULTS

The implementation of our prototype is based on the Java programming language, version 8. Jetty web server is used to host the RESTful services in the IoT network. For the implementation of the tests in the different layers defined in the previous section we have used different tools and frameworks.

Software Interaction Layer: Java code was developed for the unitary and integration tests. JUnit Framework integrated with Maven and Eclipse tools were used. For the system tests, a functional test architecture was written in Python language and allows us to emulate the broker module and the services. This model serves to achieve all functional tests before deploying the system in specific hardware devices. Functional tests include status, locate and registry tests under different conditions. Figure 1 shows our modular test architecture that has been utilized to verify the functionality of our application.

Hardware Interaction Layer (Device comm and Network): In order to implement these layer tests, we have defined some specific test cases for the connectivity tests. Table1 shows bandwith test case where criteria of test is to change the bandwidth value emulating a typical connectivity problem. Table2 present the case when the device is suddenly switched-off. Table3 focuses in the droppped connections among devices. Every test case were tested by using the modular test architecture presented previously.

Table 1: Bandwitdh tests.

Purpose	Test Bandwidth during register process
Pre-condition	Broker and Service running
Criteria	Change the bandwidth
Test Procedure	Service registering in Broker Increase/decrease bandwidth
Expected	Delay in the register process. Timeout.

In order to achieve performance test of our platform, we have successfully deployed our implementation on three different hardware platforms that are presented in Table 1. To test our application in a real environment and under stress conditions, we use a load generation tool called Tsung, which can be configured to determine the number of request per second

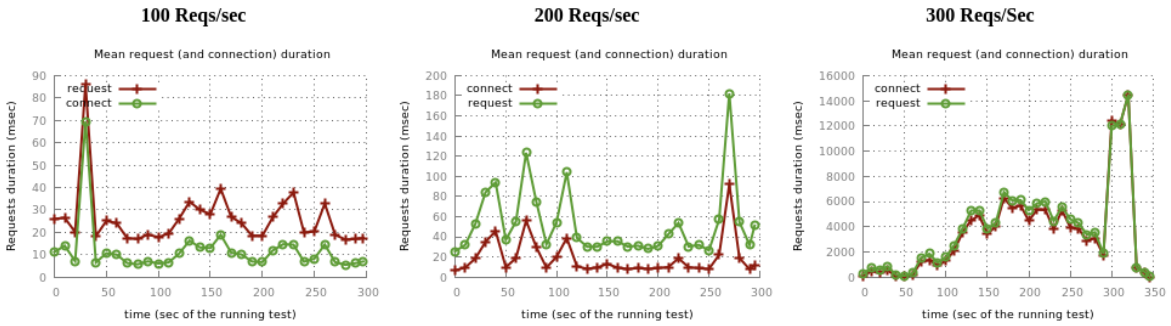

Figure 4: Results for 100, 200 and 300 requests/sec in Intel Edison device.

Table 2: Connectivity tests - Switched-off.

Purpose	Broker switched-off suddenly
Pre-condition	Broker and Service running
Criteria	Switch-off broker
Test Procedure	Service registering in Broker Switch-off the broker suddenly
Expected	Broker maintain the registered services

Table 3: Connectivity tests - Dropped Connection.

Purpose	Brokers has connection dropped
Pre-condition	Broker and Service running
Criteria	Force dropped connection
Test Procedure	Start broker & services Perform registry or locate Drop connection
Expected	Broker maintain the registered services No services found

as well as the overall test duration. Our configuration was defined as a sequence of values from [10-100] requests/sec and [100-1000] requests/sec. We have used a desktop core i5 computer to run the client side of Tsung, and started the IoT application on the server side.

Tsung generates a high load of HTTP requests in the client side towards server side. Tsung tool generates an HTML reports for each simulation and there is a Perl script that is used in order to generate the performance graphics from the log file. So for example in the case of Intel Edison device we simulate for 10, 20, 30, 40, 50, 100, 200, 300, 400, and 500 requests per second. Figure 4 only presents the response time results for 100, 200 and 300 requests per second. We choose these specific simulations because at 300 requests/sec the response time of our application starts to increase in the order of seconds that is not suitable for a simple HTTP request. Besides that we focus on the performance of Intel Edison because it is the only device that was using wifi interface which is important for our research.

After analyzing the reports generated by the Tsung tool, we can determine the maximum number of HTTP requests per second that can be supported

Table 4: Hardware platforms.

Type	Frequency	RAM	Interface
NUC	Core i5 @1.3GHZ	4GB	Ethernet
Edison	Atom @500MHZ	1GB	Wifi
Galileo	400MHz	256MB	Ethernet

by each device. We present this result in the Figure 5, the worst device was Galileo1 board, it works fine until 70 requests per second, after that the response time increases exponentially. For Edison board, this number is 200 request per second. And for Intel NUC, the number is more than 1500 requests per second.

Figure 5: Requests per second vs Response Time.

For the case of interoperability tests, we have performed tests about HTTP and CoAP communication through a proxy in order to guarantee the compatibility between these both protocols.

User Interaction Layer: this kind of tests is currently in development.

5 CONCLUSION AND FUTURE WORK

This work presents the initial testing results of our IoT application. We have defined three different layers for testing: software, hardware and user. Software layer tests were done by using standard software frameworks like JUnit. In the case of hardware layer, we

have deployed our system on three different hardware platforms. We have used the Tsung tool to perform the tests by using several configurations of requests per second. Initial results show the maximum number of requests per second that can be supported for each hardware device. Best performance device was the Intel NUC, followed by Intel Edison and the weakest device was Intel Galileo. According with the results, we can determine some connectivity problems when constrained devices are used for the specific Java implementation. Then it is important to define our hardware requirements before start developing IoT applications.

For our future research, we plan to continue the stress testing but under different conditions, for example increasing and decreasing the power of the wifi signal of the devices. Security tests will need to be implemented because currently we are implementing the authentication modules of our platform.

REFERENCES

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., and Watteyne, T. (2016). FIT IoT-LAB: A large scale open experimental IoT testbed. In *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pages 459–464.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, 17(4):2347–2376.

Belli, L., Cirani, S., Davoli, L., Gorrieri, A., Mancin, M., and Picone, M. (2015). Design and Deployment Oriented Testbed. *IEEE Computer*, 48(9):32–40.

Bloem, J. (2016). *IoTMap - Testing in an IoT environment*. Sogeti Publisher.

Cognizant (2016). The internet of things: Qa unleashed. https://www.cognizant.com/InsightsWhitepapers/the-internet-of-things-qa-unleashed-codex1233.pdf. Last accessed: November 2016.

Colitti, W., Steenhaut, K., and De Caro, N. (2011). Integrating wireless sensor networks with the web. *Extending the Internet to Low Power and Lossy Networks (IP+SN 2011)*.

Costa, L. (2015). Swarm os control plane: An architecture proposal for heterogeneous and organic networks. *2015 IEEE International Conference on Consumer Electronics (ICCE)*, pages 245–246.

Lunardi, W. T., de Matos, E., Tiburski, R., Amaral, L. A., Marczak, S., and Hessel, F. (2015). Context-based search engine for industrial iot: Discovery, search, selection, and usage of devices. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8.

Mineraud, J., Mazhelis, O., Su, X., and Tarkoma, S. (2015). A gap analysis of internet-of-things platforms. *CoRR*, abs/1502.01181.

Rabaey, J. M. (2011). The swarm at the edge of the cloud-a new perspective on wireless. In *VLSI Circuits (VLSIC), 2011 Symposium on*, pages 6–8. IEEE.

RCR-Wireless (2016). Testing the internet of things: Making the iot work.

Reetz, E. S., Kuemper, D., Moessner, K., and Toenjes, R. (2013). How to test iot-based services before deploying them into real world. In *Wireless Conference (EW), Proceedings of the 2013 19th European*, pages 1–6.

Sanchez, L., Mu??oz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E., and Pfisterer, D. (2014). SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks*, 61:217–238.

Thangavel, D., Ma, X., Valera, A., Tan, H. X., and Tan, C. K. Y. (2014). Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6.

Vandikas, K. and Tsiatsis, V. (2014). Performance evaluation of an iot platform. *Proceedings - 2014 8th International Conference on Next Generation Mobile Applications, Services and Technologies, NGMAST 2014*, pages 141–146.

W3C-Group (2016). Direct to device connectivity in the internet of things. https://www.w3.org/WoT/. Last checked: January 2016.