2016 IEEE First International Conference on Internet-of-Things Design and Implementation

Performance Evaluation of an M2M Platform in Different Deployment Setups

Chayma Berrhouma[1], Asma Elmangoush[2], Adel Al-Hazmi[1], Ronald Steinke[1], Thomas Magedanz[2]
[1]Next Generation Network Infrastructures
Fraunhofer FOKUS Institute - Berlin, Germany
[2]Next Generation Networks Department
Technical University Berlin - Berlin, Germany
{chayma.berrhouma, adel.alhazmi, ronald.steinke}@fokus.fraunhofer.de, asma.a.elmangoush@campus.tu-berlin.de,
thomas.magedanz@tu-berlin.de

***Abstract*— Machine-to-Machine (M2M) communication technology is emerging as one of the major trends shaping the development of services in Smart Cities. However, there are grand challenges related to complexity in integrating existing vertical silos limiting services interoperability. Providing standard M2M platforms allows the interaction of billions of devices and services in a network independent way. In this paper, we present the results of performance evaluating of a common platform for M2M using the open Machine Type Communication platform (OpenMTC), which is aligned with various M2M standards. The evaluation process considered different deployment setups (hardware, transport protocols, interaction models, etc.) and traffic conditions likely common in Smart City, EHealth and Smart Energy services.**

Keywords: Machine-to-Machine; Internet of Things (IoT); Evaluation; Standard Platform

I. INTRODUCTION

The communication world is heading to a new era, where a huge number of things (e.g., smartphones, air-conditioners, meters, cars, etc.) will be interconnected within the Internet of Things (IoT). The vision of the IoT is to enable objects to be connected any-time, any-place, with any-thing and any-one ideally using any-path/network and any-service. From a technical point of view, this vision could not be accomplished by implementing one novel technology; instead, several complementary technical developments shall provide functionalities and capabilities to assist in bridging the gap between the virtual and physical world. To this end, the successful development of such systems requires embedded sensing devices, efficient communication infrastructure over Machine-to-Machine (M2M) platforms, and intelligent data processing capability.

Standardization activities in the M2M communication are looking indeed into the direction of horizontal solutions to support the interoperability of vertical application domains. In this regard, various standard organizations are working on gathering requirements from different vertical domains and specifying common reference architectures accordingly. A standardized architecture with a common set of service layer capabilities and open Application Programmable Interfaces (APIs) should help M2M service providers to reduce investments, time-to-market, development and on-boarding costs and facilitate management of devices and applications.

M2M platforms support multiple types of interactions between connected devices, such as pushing data, retrieving data, and receiving notifications of dedicated events. These interactions shall take place through open standard interfaces to support interoperability between different services and vendors. Nevertheless, the smooth running of such operations could be affected from one side by the transmission network that can cause delays and data loss, in addition to the efficiency of the services provided by the M2M platform itself. Furthermore, the role of the M2M platform in ensuring a reliable M2M services is to proses the incoming requests in a very short time in order to minimize as maximum as possible the end-to-end delay with reliability and without message lost.

The remainder of the paper is structured as follows. In Section II background information related to M2M communication and relevant work by others are presented. Section III discuss the design and architecture of the OpenMTC platform. Section IV presents the performance evaluation of our system. Section V discuss the usability of our platform in different M2M solutions. Finally, the paper is concluded in Section VII.

II. BACKGROUND AND RELATED WORK

The European Telecommunications Standards Institute (ETSI) M2M technical committee has defined a middleware Service Capability Layer (SCL) that interact with M2M nodes over open interfaces named: mIa, dIa and mId [1][2]. These interfaces offer generic and extendable mechanism for interactions with the SCLs at both device and gateway domain and network domain.

Recently, the oneM2M consortium was established with the aim to become the world-wide accepted standard for M2M/IoT communication by consolidating the work from ETSI M2M and other international standardization bodies. OneM2M specifies a high-level architecture at both the field and infrastructure domain, to support end-to-end M2M services. The oneM2M functional architecture comprises of three entities: Application Entity (AE), Common Services Entity (CSE), and Underlying Network Services Entity (NSE). Each M2M node (i.e., Infrastructure Node (IN), Middle Node (MN) or Application Service Node (ASN)) could deploy one or more entities. Both ETSI M2M and oneM2M have aimed to develop an application-agnostic M2M framework for the vertical market solutions, with

978-1-4673-9948-7/16 $31.00 © 2016 IEEE
DOI 10.1109/IoTDI.2015.23

emphasis on specific technologies. Considering the connectivity control aspect, each SCL in the ETSI M2M architecture shall implement the Generic Communication (GC) capability, which is responsible for the established transport session including encryption and reporting errors features. Similarly, the Communication Management and Delivery Handling (CMDH) function, defined by oneM2M, handles the communication functionalities with other entities i.e. CSEs, AEs and NSEs. The CMDH uses the Underlying Network equivalent delivery handling functionality based on provisioned policies to decide when to use the communication channel to transmit the data. The M2M server deployed in the Infrastructure Node (IN) is expected to handle a wide variant of M2M traffic patterns, while most of existing communication platforms in the operator infrastructure have been developed to support regular Human-to-Human traffic. A conceptual overview of different M2M traffic classes is presented in [3].

There are several research projects working towards define a reference architecture for IoT system deployments. The IoT-A project presented the direction of defining a general Reference Model Architecture for IoT, to be used as a basis of platforms design. The project has created an "Architectural Reference Model" (IoT ARM) as the common ground for the field of IoT. The model defines entities and describes their basic interactions and relationships with each other [4]. The FI-WARE service infrastructure developed upon reusable Generic Enablers is making the development of value-added services easier. A number of research projects have been established as a phase-two use case projects that aim to validate the FI-WARE Generic Enabler in specific use cases, such as FI-STAR [5], XIFI [6] and FRACTALS [7]. The compatibility of FI-WARE IoT Generic Enablers with ETSI M2M service layer is quite notable, which facilitates the interoperability of the developed applications with other platforms. However, global interoperability with international standards is still missing.

III. OpenMTC Design and Architecture

The OpenMTC platform [8] is a prototype implementation of recent standard specification for M2M service middleware. It has been designed to act as a horizontal convergence layer supporting multiple vertical application domains, such as transport, automotive, eHealth, etc. The platform may be deployed independently or as part of a common platform. OpenMTC features are aligned with ETSI M2M Rel. 1 specifications [1][2] and oneM2M Rel. 1 specifications [9].

Keeping in mind the diversity of computing capabilities of connected devices, the OpenMTC front-end components are implemented with support of various hardware platforms, such as Android platform for mobile devices and Arduino platform for constrained devices. On the one hand, Android smart phones could be utilized as M2M gateways for sensors and devices connected to the Personal Area Network (PAN), such as eHealth sensors. On the other hand, Arduino provide a light platform for power constrained devices installed for long range monitoring and controlling usage, such as home automation to remote control measurement. OpenMTC can be deployed in devices equipped with sensors or/and actuators and supporting their special requirements and capabilities.

OpenMTC Release 4 features include several features that can be deployed in the field domain gateway/device and in the infrastructure nodes. The architecture, as depicted in Fig 1, is based on horizontal functional layers covering the following:

A. Application Enablement

OpenMTC supports a client/server based RESTful architecture with the hierarchical resource tree defined by ETSI M2M and oneM2M. The data exchange communication over open interfaces is independent of the transport protocol in use. OpenMTC supports both of MIa interface specified by ETSI as well as Mca interface specified by oneM2M. Furthermore, the application enablement layer defines a set of high-level abstraction API, which are categorized under three groups: Device, Data and Network APIs. In addition, OMA NGSI-9 and NGSI-10 interfaces for context management [10] are supported on the backend server, to allow seamless integration of M2M platforms.

B. Core M2M features

The GEvent is used as underlying platforms to manage events across OpenMTC components. It composes of event based generic libraries providing asynchronous I/O API that can scale its number of execution units according to the processing load. While it is often used for its speed and locality as it is targeted for constrained devices or single-instance setups, it can be deployed in could-based infrastructure. To persistently store data related to entities in the M2M system, OpenMTC internally uses a database abstraction layer. By using different database adapters, OpenMTC can be configured to data in different data back-ends with different characteristics depending on the usage

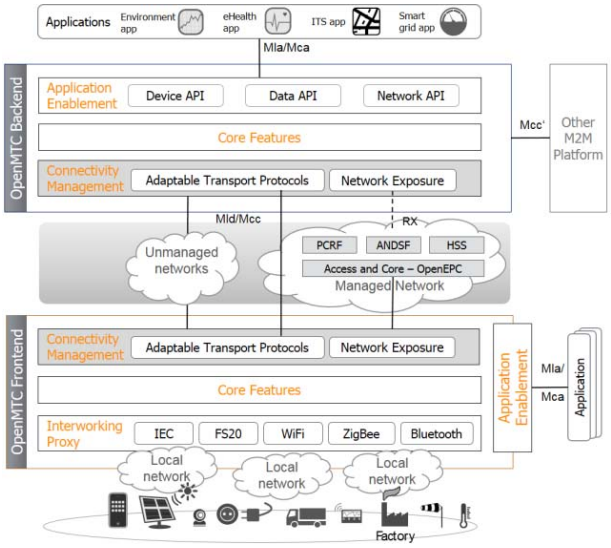

Figure 1. OpenMTC Functional Architecture

scenario at hand. Additionally, OpenMTC features an extended subscriptions management system in which the applications as well as the remote M2M common feature are notified when specific data is created, modified or deleted.

C. Connectivity and Network Exposore

A connectivity management layer enables the interaction between the frontend and backend over managed access, and unmanaged Access Networks. This allows the integration with 3GPP networks based on the Diameter protocol over the Rx interface. The used bearer can be selected based on defined policies. Both ETSI mId and oneM2M Mcc reference point implementation specified support for synchronous, semi-synchronous and asynchronous communication between the network service platform and M2M gateways/devices. Various transport protocols are supported such as HTTP and CoAP. Allowing to use transport protocols dynamically depending on the application. A Store and Forward (SAF) feature is supported by OpenMTC, which enables the handling of different traffic streams based on their priority.

D. Interworking proxy

Interworking Proxies allow interaction with non-ETSI/oneM2M compliant devices or systems. In order to enable interaction with off-the-shelf sensors and actuators and to offer the OpenMTC functionality to devices not compliant to ETSI and oneM2M standards, Gateway Interworking Proxy (GIP) are defined to translate specific control and data requests to the ETSI and oneM2M conformant model. Various protocol adapters are implanted based on different access technologies like Bluetooth, Zigbee and FS20. Similarly, Network Interworking Proxy (NIP) are implemented to enable the interworking with other service platforms or M2M standards by translating messages from other platforms to the ETSI/oneM2M standard [11].

E. Device Management

OpenMTC platform integrates a Device Management (DM) implementation based on the OMA LWM2M protocol [12]. A library for LWM2M message parsing and creating, and managing communication back to the registered clients is implemented at both front-end and back-end servers. The LWM2M library creates a tree dictionaries of the supported Management Objects (MOs) and is easy to extend with another one by adding new entries in the dictionary. The processing is then uniform for each of the management objects when it comes to parsing, storing, updating information related to the management objects.

IV. PERFROMANCE EVALUATION SCENARIOS

M2M devices support a large variety of applications in several domains (e.g. Smart metering, e-Health, Home automation, etc.). Therefore, we face various traffic patterns and volumes. As the OpenMTC platform is generic and abstracted from any M2M domain, it should be able to support any type of traffic as well as message size. The evaluation of the key functionalities of the OpenMTC has been carried out within the Future Seamless Communication (FUSECO) Playground at Fraunhofer institute FOKUS.

Parameters of M2M traffic pattern alter depending on the sensor type and application features, and it might be periodic with constant time intervals or randomly generated. In order to avoid non-meaningful tests, a set of sampling rate values has been selected according to the requirements of the actual existing M2M applications specified in multiple research studies [13][14]. The same criteria have been used to choose the values set of the payload size parameter.

A. Tesbed Setup

The OpenMTC platform runs on different types of hardware platforms with different capabilities. In the performance evaluation presented here, we will cover two types of systems for an OpenMTC gateway: an embedded system operated with Raspberry Pi (model B) BCM2708 processor, uses the ARMv6 instruction set with 400 MB RAM, and a PC with 2.4 GHz quad-core Intel processor, 8 GB RAM. Our testbed setup, as shown in Fig 2, is composed of three main parts: the OpenMTC gateway, M2M application and M2M device. The sensor traffic generator is running on a PC, with 3.70 GHz Xeon Intel processor, 16 GB RAM and running Ubuntu OS. For the M2M application emulator, a PC with 2.40GHz Duo core Intel processor, 3GB RAM is used. In order to define a meaningful evaluation scenario, it is important to choose well the test parameters and the performance metrics.

During the evaluation, the test cases consider two types of M2M interactions to test, these are pushing data and retrieving data. Therefore, we will study the case of having different devices/sensors pushing/retrieving different types of data to/from the platform. All the interactions and the M2M traffic exchange will be established through both HTTP and CoAP protocols.

Apache JMeter was used to carry out all evaluation scenarios. Apache JMeter is a pure Java application designed for load testing and performance measurement of different servers/protocols such as HTTP, HTTPS, REST, etc. [15]. It could be also extended to support other protocols, such as CoAP and MQTT, through multiple pluggable samplers[15][15]. A sampler plugin for CoAP was deployed to perform the presented performance evaluation. In order to emulate the use case of having multi-devices connected to the OpenMTC gateway, we have used the parallel JMeter Ant task. In fact, Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent on each other. For the high sample rates, we have used the JMeter distributed testing, which consists in running the same test

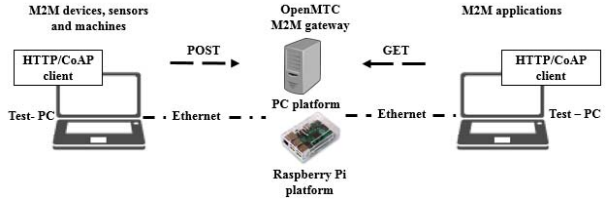

Figure 2. Experimental testbed setup

scenario on different virtual machines on the cloud to overcome the one machine limitation. One machine (master) can control any number of other machines (JMeterEngines/slaves) and collect all the data from them. All test machine's clock have been synchronized using the Network Time Protocol (NTP), belong to the same LAN and use a wired Ethernet connection. By choosing a wired connection, we aim to eliminate any problems related to the network transmission such as packet loss, congestion, etc. Each test scenario lasts more than 5 minutes and has been repeated at least 5 times in order to get accurate results.

On each test scenario, an application registered to the M2M gateway prior to start the data interaction of pushing or retrieving data. According to the oneM2M specifications, data storage in M2M gateways is associated with container related operations. In our tests, each registered application has one container that enables maximum 10 content instances to be stored with a maximum size 1Kb each. These parameters have been set to limit the effect of database processing time.

B. Results, Scientific and Technical discussion

Fig 3 and Fig 4 plot the response time/delay when pushing data with POST request, as well as the 95% confidence interval for different sampling rates. The requests were sent over HTTP and CoAP protocols respectively to the OpenMTC gateway (PC and Raspberry Pi platforms). The payload size was fixed to 200bytes. The 95% confidence interval (CI) aims to describe how reliable and relevant is the average value by showing us the range of the 95 % of the captured results in each test scenario.

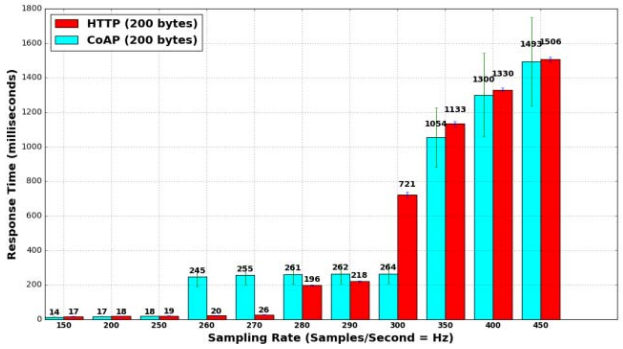

Figure 3. Impact of a periodic traffic on the response time (POST & PC platform)

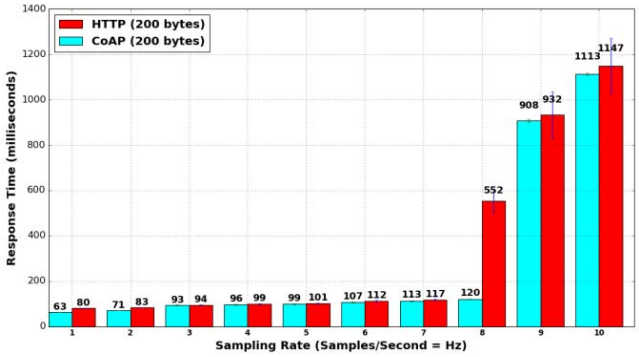

Figure 4. Impact of a periodic traffic on the response time (POST & Raspberry Pi platform)

From Fig 3 a slight increase in the response time values for the sampling rates lower than 250 Hz (for CoAP) and 270 Hz (for HTTP) is observed, which is followed by a strong surge that exceeds one second for the rates higher than 350 Hz with both protocols. A deeper look in the response time behavior over a specific period when using HTTP, allows us to distinguish between two states (for the rates lower than 270 Hz). At the beginning of each connection, the delay is extremely high and unstable, and then it maintains lower values during the remaining time.

These steep values at the starting of each connection could be explained by the fact that HTTP is based on TCP connection which causes higher latency due to connection setup (three-way handshake & slow start). However, for the rates higher than 280Hz, we have only one state as the response time values remain high during the period. The processing time within the OpenMTC platform becomes longer. Beyond 250 samples/second (CoAP), we notice a strong rise in the delay values. The same surge also has occurred with lower payload size messages (around 10 bytes). This could be related to the CoAP protocol implementation according to [16] which mentions that a 16-bit size message can enable up to about 250 messages per second from one endpoint to another with default protocol parameters similar to this case. As illustrated in Fig.5 the upper bound of short delays is approximately between [7-9 Hz] for both protocols. Beyond this limit, the system is overloaded as the response time exceeds 1 second for most requests while it reaches more than 1 minute for other ones.

Table 1 summarizes the resources usage during the tests. The memory usage is relatively the same for both protocols and for different sample rates. However, the CPU utilization appears slightly higher in case of using HTTP. The resource consumption differences between the fact of using HTTP and CoAP is clear in the case of using the Raspberry Pi. The load average (process in queue) is relative to the number of processor cores, e.g., for the single core processor 1.00 means 100% CPU utilization while for the quad core 4.00 signifies 100% CPU utilization. For the higher sample rates, one of the CPU cores (PC platform) usage reaches up 98 % (the OpenMTC gateway is one core based). Generally, CoAP performs slightly better than HTTP for the low, medium and high sample rates [1-250 Hz, 300-500 Hz] for both platforms. More tests were carried on with variable traffic rates using HTTP and CoAP, it was noticed that the OpenMTC gateway could handle the fast variability in the number of incoming requests (between 1 and 250 Hz) efficiently.

Fig 5 and Fig 6 show the response time of POST requests at fixed sampling rate with different payload sizes running respectively on the PC and Raspberry Pi platforms. It is clear that the impact of the payload size (sizes lower than 1Kb) is negligible for both HTTP and CoAP protocol and both platforms. As the size of the CoAP one block is between 16 and 1024 bytes in case of using block-wise transfer option, which means that only one block has been sent in each test scenario (payload size<1Kb). In addition, the Ethernet MTU is equal to 1500 bytes at the network layer, then no IP

fragmentation for CoAP messages (UDP supports larger payloads through IP fragmentation), and no TCP fragmentation for HTTP messages have occurred.

TABLE I. RESOURCES USAGE

Transport Protocol	Rate (Hz)		CPU (%)		Process in queue	
	PC	Ras	PC	Ras	PC	Ras
HTTP	270	1	75	2	1.62	0.25
	280	7-8	83	65 – 85	2.04	1.0 - 2.425
	290	9	86	99	2.41	2.95
CoAP	250	1	47	1	0.57	0.12
	260	8	69	72	0.62	1.0
	270	9	73	98	0.74	1.5

The results obtained with payload sizes bigger than 1Kb prove the ability of the OpenMTC gateway to handle and process high request message sizes in a very short period. Additionally, the impact of the packet fragmentation on the response time for CoAP packets was more noticeable. Generally, HTTP performs much better than CoAP with high payload sizes. The system performance was tested also for data retrieving interaction. However, the results are not included due to space constraints.

Testing the system performance with multiple connected nodes proves the scalability of the OpenMTC. The measurements showed a strong increase in responses time

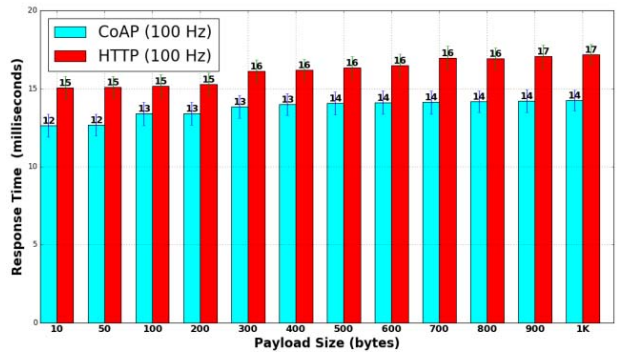

Figure 5. Impact of the payload size on the response time (POST & PC platform)

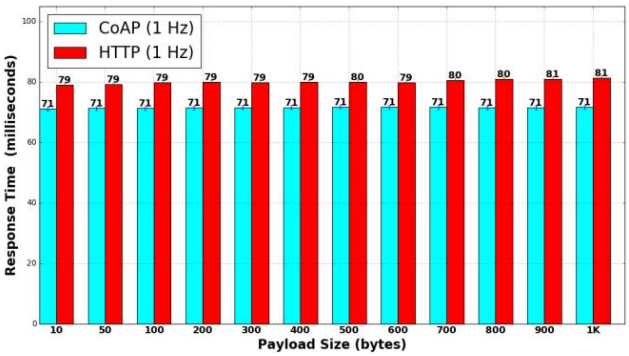

Figure 6. Impact of the payload size on the response time (POST & Raspberry Pi platform)

when the number of total transactions is between 240Hz and 320Hz for HTTP, while there isn't any range for CoAP, as it's very variable depending on the number of devices and sample rates. The Raspberry Pi platform causes very long delays in case of increasing slightly the number of devices even for the very low rates. As mentioned earlier, the system is overloaded, when the sample rate exceeds 9 Hz, in regard to one connected device and for both protocols. Therefore, the OpenMTC Raspberry Pi platform is more suitable for very low rates.

V. CONCLUSION AND DISCUSSION

For the deployment of an M2M system in any domain, such as health care, Smart Home or Industry, it's important to start by specifying the environment conditions and requirements. The understating of operating specifications including the integrated devices, traffic pattern, samples rate, and delay tolerance level, have a great impact on designing the final solution. Many options are available when selecting the components, platform capabilities and access technologies to be used in the system.

Based on the evaluation results, presented in this paper, the deployment of the OpenMTC platform on constrained resources gateway showed a fine performance for delay tolerant application with low sample rate traffic. It would be suitable to deploy Smart Home use cases to provide the monitoring of the home environment and a number of appliances, e.g., temperature, light level, relative humidity and presence. Usually, the sample rates and message payload size of such sensors/applications are very small. In such a way, the platform can reliably serve a specific number of connected devices with a minimal cost in term of hardware capabilities. Furthermore, when using CoAP protocol, the response time of data pushing and retrieving requests is less affected than with HTTP, slimier observation is obtained for the usage of hardware resources. However, a gateway with higher resource-capability is required for the more exigent M2M domains such E-health, and Smart Grid. We have already proved through the usage of two different platforms the strong impact of the hardware features on defining the limits of the system.

Nowadays, the E-health domain is gaining more momentum. Several E-health solutions have been developed for remote monitoring of patient health and fitness information, remote control of certain treatment and alarms triggering in case of detecting critical situations based on M2M communication. Several wearable sensors are available in the market to measure vital signs. Based on our evaluation scenarios and the analysis of the e-health communication requirements and traffic loads, presented in [14], we can deduce that the OpenMTC gateway is suitable for such systems and it's used by some use cases in the FI-STAR project [17]. The Smart Grid is also an important M2M domain, covering different applications such as Automatic Meter Reading (AMR) and Substation Automation [13]. The Smart Energy is one of the domains under study within the TRESCIMO project [18].

ACKNOWLEDGMENT

This research leading to these results has received funding from the SmartOrchestra project.

REFERENCES

[1] ETSI TS 102 690 v1.1.1, "Machine-to-Machine communications (M2M); Functional architecture," 2011.

[2] ETSI TS 102 921 v1.1.1, "Machine-to-Machine communications (M2M); mIa, dIa and mId interfaces," 2012.

[3] A. Elmangoush, A. A. Corici, R. Steinke, M. Corici, and T. Magedanz, "A Framework for Handling Heterogeneous M2M Traffic," *Procedia Comput. Sci.*, vol. 63, pp. 112–119, 2015.

[4] A. Bassi, S. Meissner, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, and S. Lange, Eds., *Enabling Things to Talk - Designing IoT solutions with the IoT Architectural Reference Model*. Springer, 2013.

[5] Fi-Ware Project, "Fi-Ware (Core Platform of the Future Internet)." [Online]. Available: http://www.fi-ware.eu/.

[6] "FI-XIFI Project." [Online]. Available: https://fi-xifi.eu/home.html. [Accessed: 19-Aug-2015].

[7] "FRACTALS." [Online]. Available: http://fractals-fp7.com/. [Accessed: 02-Sep-2015].

[8] "OpenMTC platform." [Online]. Available: http://www.open-mtc.org/index.html.

[9] oneM2M-TS-0001, "OneM2M Functional Architecture," vol. 1. 2015.

[10] "OMA Next Gereration Servicres Interface V1.0." [Online]. Available: http://technical.openmobilealliance.org/Technical/release_program/ngsi_v1_0.aspx.

[11] T. Klinpratum, C. Saivichit, A. Elmangoush, and T. Magedanz, "Performance of Interworking Proxy for Interconnecting IEEE1888 Standard at ETSI M2M Platforms," *Appl. Mech. Mater.*, vol. 781, pp. 141–144, 2015.

[12] Alliance Open Mobile, "Lightweight Machine to Machine Architecture V1.0." 2013.

[13] R. H. Khan and J. Y. Khan, "A comprehensive review of the application characteristics and traffic requirements of a smart grid communications network," *Comput. Networks*, vol. 57, no. 3, pp. 825–845, Feb. 2013.

[14] L. Skorin-Kapov and M. Matijasevic, "Analysis of QoS requirements for e-Health services and mapping to evolved packet system QoS classes," *Int. J. Telemed. Appl.*, vol. 2010, pp. 1–18, Jan. 2010.

[15] "Apache JMeter." [Online]. Available: http://jmeter.apache.org/. [Accessed: 12-Oct-2015].

[16] Z. Shelby, K. Hartke, and C. Bormann, "RFC 7252: The Constrained Application Protocol (CoAP)." 2014.

[17] Fi-Star Project, "Fi-Star (Future Internet Social and Technological Alignment Research in Healthcare)." [Online]. Available: https://www.fi-star.eu/home.html.

[18] A. A. Corici, A. Elmangoush, T. Magedanz, R. Steinke, J. Mwangama, and N. Ventura, "An OpenMTC platform-based interconnected European – South African M2M Testbed for Smart City Services," in *the first International Conference on the use of Mobile Informations and Communication Technology (ICT) in Africa - UMICTA 2014*, 2014, pp. 35–39.