EMBERS

HORIZON 2020
Information and Communication Technologies
Integrating experiments and facilities in FIRE+

Deliverable D2.2
First Report on Experiment Design and Description

Grant Agreement number: 687992
Project acronym: EMBERS
Project title: Enabling a Mobility Back-End as a Robust Service
Type of action: Innovation Action (IA)
Project website address: www.embers-project.eu
Due date of deliverable: 2016-09-30

Dissemination Level		
PU	Public	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document properties

Leader partner	**Technische Universität Berlin (TUB)**
Author(s)/editor(s)	Alexander Willner (TUB), André Duarte (UW), Thomas Günther (FHG), Christian Klopp (FHG), Daniel Nehls (TUB), Saint-Marcel Frederic (INRIA)
Version	0.4

Abstract

The primary objective of Work Package 2, entitled "Experimentation and Testing", is to utilize the FIRE+ infrastructures as well as the Federation and Experimentation tools set up during WP1, with the purpose of accurately defining, implementing and conducting the multiple experiments (and tests) to mature the MBaaS platform. Following this objective, Section 2 of this deliverable briefly depicts the first experiments of EMBERS designed to turn the smart city solution to a market-ready product and Section 3 describes the services provided by the FIRE+ infrastructures FIT IoT Lab by INRIA and FUSECO Playground by Fraunhofer FOKUS.

Finally, Section 4 concludes the deliverable.

Table of Contents

1 Introduction

With the purpose of achieving two of the project's primary goals (making the MBaaS platform ready to market and contributing back to FIRE+ infrastructure), experimentation with the platform requires careful planning and execution. Therefore, and following the outcomes of the WP1 deliverables, the initial phase is planned to involve the adaptation of the Mobility platform to allow its integration with the experimentation facilities, available for this project.

Since Ubiwhere's Mobility Backend as a Service platform (MBaaS) will work on a distinct cloud instance, remotely from the testing laboratories, it will need to support several communication protocols and open standards, not only for the dissemination events with third-party entities but also for the experimentation and testing phases. The final goal is to allow the communication through the most extensive set of protocols and to use the most diverse set of open standards possible, which will turn the platform into a solution ready for testing with an enormous number of devices from the experimentation facilities.

The emerging "Internet of Things" covers a vast range of industries and scales of devices/applications. Some different standardisation bodies and groups are actively working on creating more interoperable protocol stacks and open standards for the Internet of Things (IoT). As we move from the HTTP, TCP/IP stack to the IoT-specific protocol stack we are suddenly confronted with an acronym soup of protocols. Below the requirements for the communication protocols that MBaaS should provide are described.

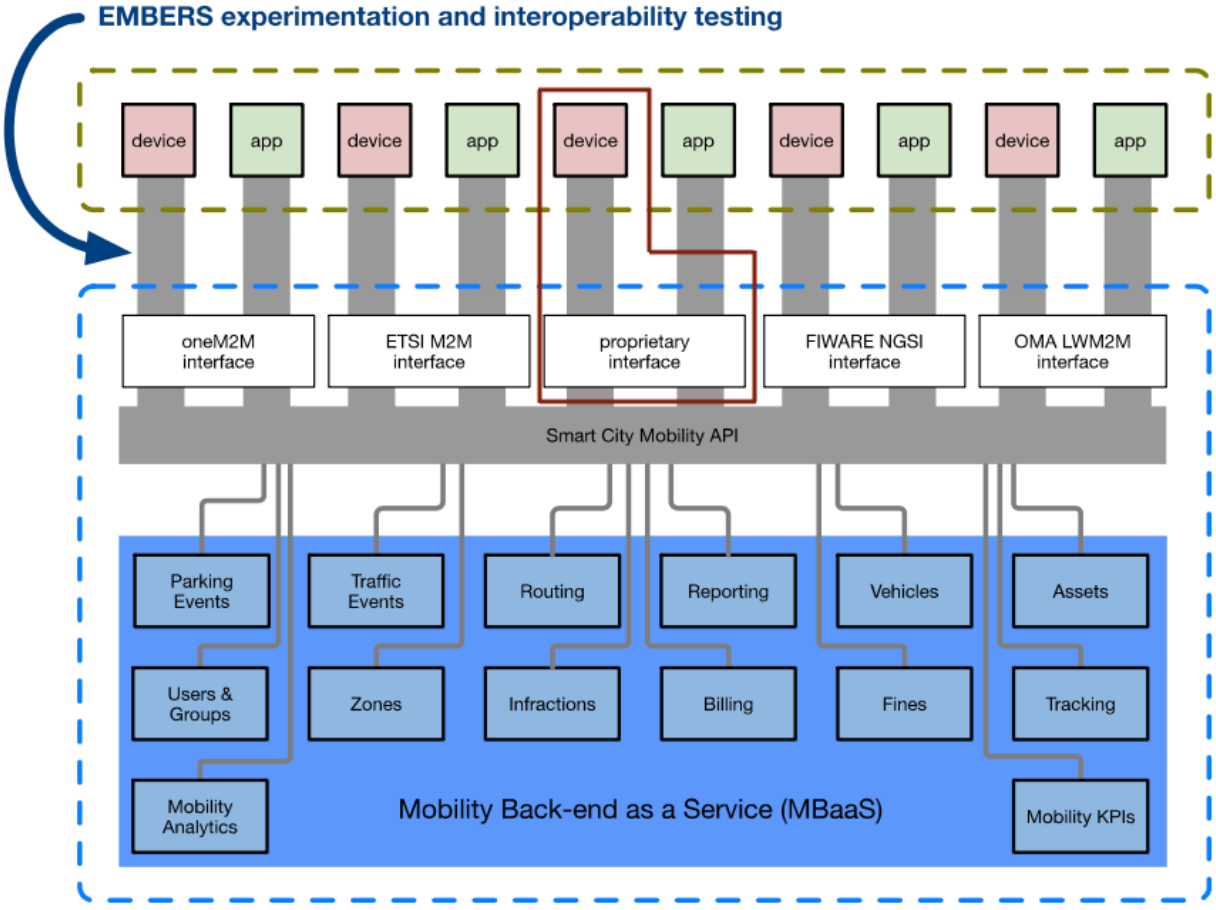

Figure 1 - MBaaS after EMBERS (with the red box representing the involved parts in the experimentation tests)

2 Experiment Description

The experiments found in the next sections were done in order to test the communication between devices available at the FIRE+ laboratories and the mobility backend. In order to achieve it, three different scenarios were tested. These communication tests used three types of data: messages with random values, defined data pre-provided for devices by the FIT-IoT Lab and pre-defined data provided by the Load Generator component that emulates the devices behaviour. The Load Generator can also include historical datasets with real events so that the test can include real data from real devices (e.g. parking sensors). The detailed requirements for the experiments are available in deliverable D1.1, where all the requirements and the metrics used to evaluate the performance limits are specified. In this experimentation tests, we have accomplished to complete some of them, but others are still to be made. Figure 1 represents the MBaaS after EMBERS, where we can clearly observe the main layers of the platform. The parts involved in the tests are surrounded by the red mark, which are:

- Communication broker - Meshblu;
- Communication standard - Citibrain proprietary standard;
- Communication protocol - HTTP.

This scenario was chosen due to its simplicity, while it is important to clarify that these types of integrations take time and need to be done in an iterative way. Therefore, we started to experiment with this simple scenario and in the future, these will be held in more complex scenarios with different communication brokers, standards and protocols as is specified on the deliverable D1.1. Alongside this, tools for the FIRE+ testbeds have been provided, which aim to simplify the interaction process. In the FIT-IoT LAB case, there is a tool to register sensors on Meshblu and send messages and, in the FUSECO Playground case, a Load Generator has been developed, with the capability to mimic the behaviour of sensors.

About each of the experiments, the first (Simple Broker Communication) creates a strong foundation to further enhance the complexity of experiments. Experiments two and three, FIT-IoT Lab Tool and Load Generator, were developed based on the first experiment and their main goal was to test the communication between real and simulated devices and the mobility backend. They were also used as a way to test the system against an increased load, where the number of devices simultaneously communicating is highly increased.

In a simple way, Figure 2 represents the high-level overview of the experiments. Using the HTTP protocol, each one of the devices is registered on Meshblu. After that, each one of them can send messages about the indicators they measure.

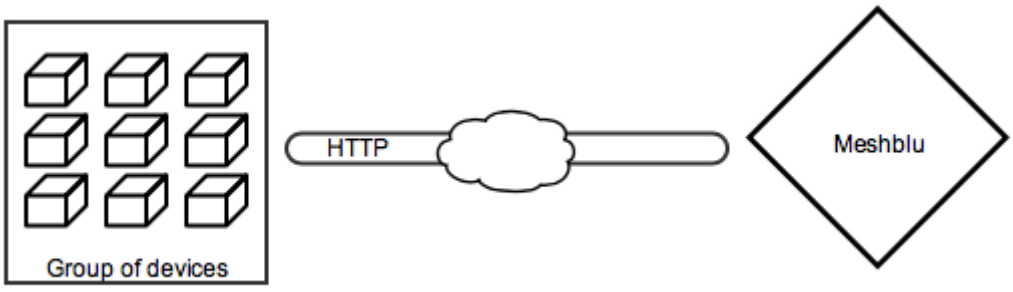

Figure 2 - High-level overview of the experiments

In order to test the performance limits, the requirements and metrics defined in the deliverable D1.1 were used. The requirements consist of the following:

- Test the MBaaS with all the device brokers;
- Test the platform with all the protocols available per device broker;
- The reports must provide a successful / failed messages ratio;
- The platform must provide tools to run tests automatically;

The metrics consist of the following:

- Number of sensors;
- Frequency of measurements;
- Latency of measurements;
- Successful/failed messages ratio;

Therefore, each of the metrics is supposed to test each of the requirements. However, these experimentation tests only considered the number of sensors, with the Meshblu device broker and the HTTP protocol with a tool to automatically run all the tests. Each of the three experiments is described below.

2.1 Experiment 1 – Simple Broker Connection

This experiment consists of testing Meshblu via HTTP. The primary purpose is to validate the communication between a new device and the MBaaS, regardless of the data provided by the devices. This experiment covers some of the requirements gathered in D1.1, specifically, the ones that intended to test the MBaaS with all the device brokers and the one that aims to test the platform with all the available protocols. This experiment involved four main steps:

- Provide a Meshblu gateway to register devices
- Register a new device at the Meshblu gateway
- Send messages with random data
- Verify that device was correctly registered and the message received

Although being a very simple use case, this experiment represents the starting point for next two, because ensuring that this simple use case works correctly, it is possible to test the next two experiments that represent more complex. Once more, the process of doing these experiments needs to be iterative, to understand all possible failure points and mitigate them in the future.

2.2 Experiment 2 – FIT-IoT Lab Tool

This experiment consists of Meshblu tests via HTTP using pre-defined data sent by real devices provided by FIT-IoT-Lab. The primary purpose of this test is to validate the communication between these devices and the MBaaS, with the particularity that the number of devices was increased.

This operation is characterised by the increasing number of sensors. This occurs with the purpose of experimenting the MBaaS with a growing number of devices communicating with it at the same time, to understand its scalability and determine its limits regarding the maximum amount of sensors sending events simultaneously.

This experiment involved four main steps:

- Provide a Meshblu gateway to register devices;
- Register new devices at the Meshblu gateway using FIT-IoT Lab tool;
- Send messages with pre-defined data provided from real sensors (devices) by using FIT-IoT Lab tool to the Meshblu gateway;
- Verify that devices and messages were correctly registered and received.

To do this experiment and test the MBaaS against an increasing number of devices communicating with it at the same time, it was used a tool[1] provided by FIT-IoT Lab. Also, this instrument was used as a way to automate all the test process.

The automated process done by this tool can be divided into three main parts:

1. Register a defined amount of devices at the Meshblu gateway. Figure 3 represents the devices registration process using this tool.

```
Register device m3-1.lille.iot-lab.info : abf8ecfd-63e6-484c-9a3f-dd86502378f3 - 95e1ec324b51d509ffac09f4c629aa2ec2cec8a3
Register device m3-3.lille.iot-lab.info : 355c0747-bc3e-4a76-acd3-82e3d46c5c70 - 75076282c7c4b844d6fcc7889aefa08a25dbc947
Register device m3-6.lille.iot-lab.info : 6c6ca41f-7c4b-428d-8f96-0d7d349727fd - 151c856cf7b43c87cbb22f2f8565951b9ad17e09
Register device m3-9.lille.iot-lab.info : 2311444f-9510-4477-99d8-5e818cef861e - 7a01669f986b6d73f92a648e5379f207d2d0e373
Register device m3-10.lille.iot-lab.info : ac218b5b-c865-415a-b5bb-c3ea0ac116a3 - c5e5fa6797d990f761ccf9ad81165b1164119afc
Register device m3-11.lille.iot-lab.info : 33fb98f2-8c81-4634-8c6b-b22423ef6847 - b3d76f21f97e88658186f190c9fe04c8a6cc15ff
Register device m3-14.lille.iot-lab.info : 9bed6e70-4787-48c1-a2e5-73b58401405c - 0cb62da4af49a7152fcecbb8c247c0d86197b4f0
Register device m3-15.lille.iot-lab.info : 850daae8-2717-4837-ac17-52de88277b35 - 22edfa2d418efbf761ee5f261f1753c5054462aa
Register device m3-18.lille.iot-lab.info : f9a7caf5-8f4b-4b01-b60e-aa4bd2dfe1fd - d8417b83247a25bfacc1565787873fef379bbc98
Register device m3-19.lille.iot-lab.info : 8fa2c6cb-b453-4d03-baf8-8bc8e23f529f - 09f80f7368ce64bc14006d5a2b8889ce82d39b8c
```

Figure 3 – Registering devices with Meshblu

2. Send messages to each one of the devices previously listed at the Meshblu gateway. Figure 4 represents the messages that are sent to the Meshblu gateway using this tool.

1 https://github.com/iot-lab/embers

Figure 4 – Sending messages to the registered devices in Meshblu

3. Unregister all the previously registered devices. Figure 5 represents the Meshblu process to unregister devices.

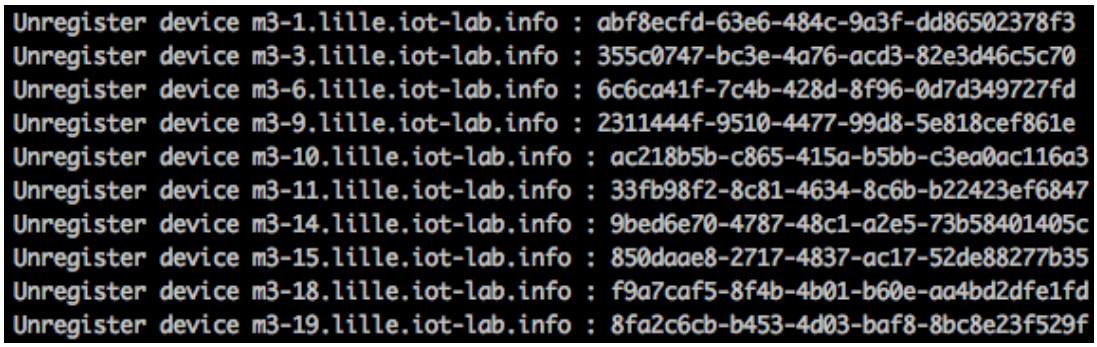

Figure 5 – Unregistering devices from Meshblu

With this experiment, it was possible to ensure that the communication between FIT-IoT Lab devices and the MBaaS works as expected, regardless of the number of devices doing it. This experiment was also used as a way to test the system in an environment similar to the real one. Likewise, the expected real-life scenario, with this operation, the registration of an increased amount of devices communicating simultaneously with Meshblu has been made possible.

2.3 Experiment 3 – Load Generator

This experiment consists of Meshblu tests via HTTP using simulated data provided from a Load Generator. The primary purpose of this experiment, similarly to the previous ones, was to validate the communication between devices and the MBaaS, but instead of using the using FIT-IoT Lab tool, the devices and the data sent by them were simulated using the Load Generator provided by Fraunhofer FOKUS. The Load Generator could also include historical datasets so that the simulated devices could send messages with real data created previously by real devices.

This experiment involves four main steps:

- Provide a Meshblu gateway to register devices;
- Register new devices simulated by the Load Generator;
- Send messages to the simulated devices by the Load Generator;
- Verify that devices and messages were correctly registered and received.

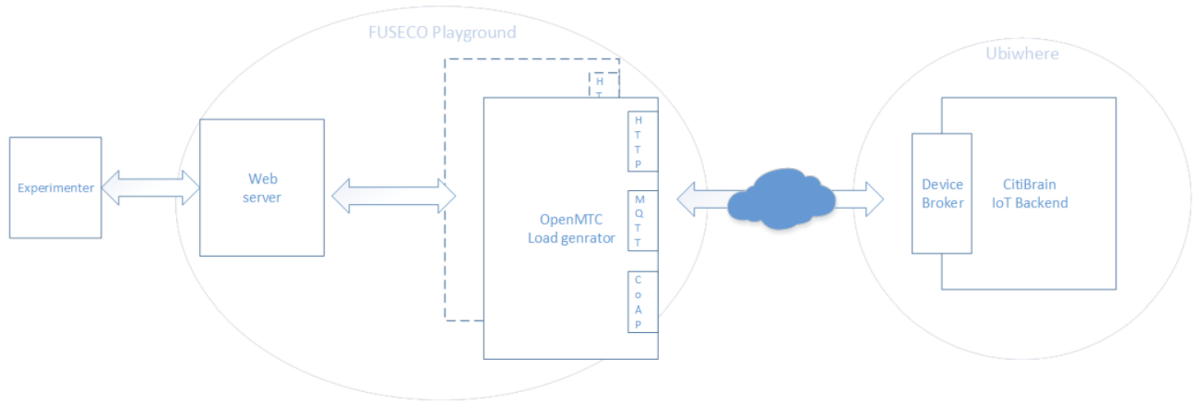

Figure 6 - Load Generator high-level architecture

Figure 6 represents the high-level design of load generator scenario and workflow. In order to start an experiment with the Load Generator we need to do the follow steps:

1. Send configuration file (JSON syntax) via RESTful API to the web server that is forwarded to the Load Generator;
2. The Load Generator parses the configuration file and sets up the number of virtual sensors provided in the configuration file;
3. All devices are registered at the communication broker - Meshblu;
4. If all sensors are registered successfully, each one will start sending data to Meshblu;
5. After that, it is possible to acquire the experiment log from the web server via RESTful API.

On the configuration file it is possible to set the follow parameters:

- Amount of devices;
- Frequency of information send;
- The size of the payload.

In order to make the performance test more realistic, were integrated historical data sets of parking events of municipalities as an input source for the load generator.

With this experiment, it was possible to test the communication between simulated devices and the mobility backend. With the load generator tool, developed by Fraunhofer FOKUS, there is a way to test simulated devices with different configurations, which is important given that it allows automating tests on different scenarios.

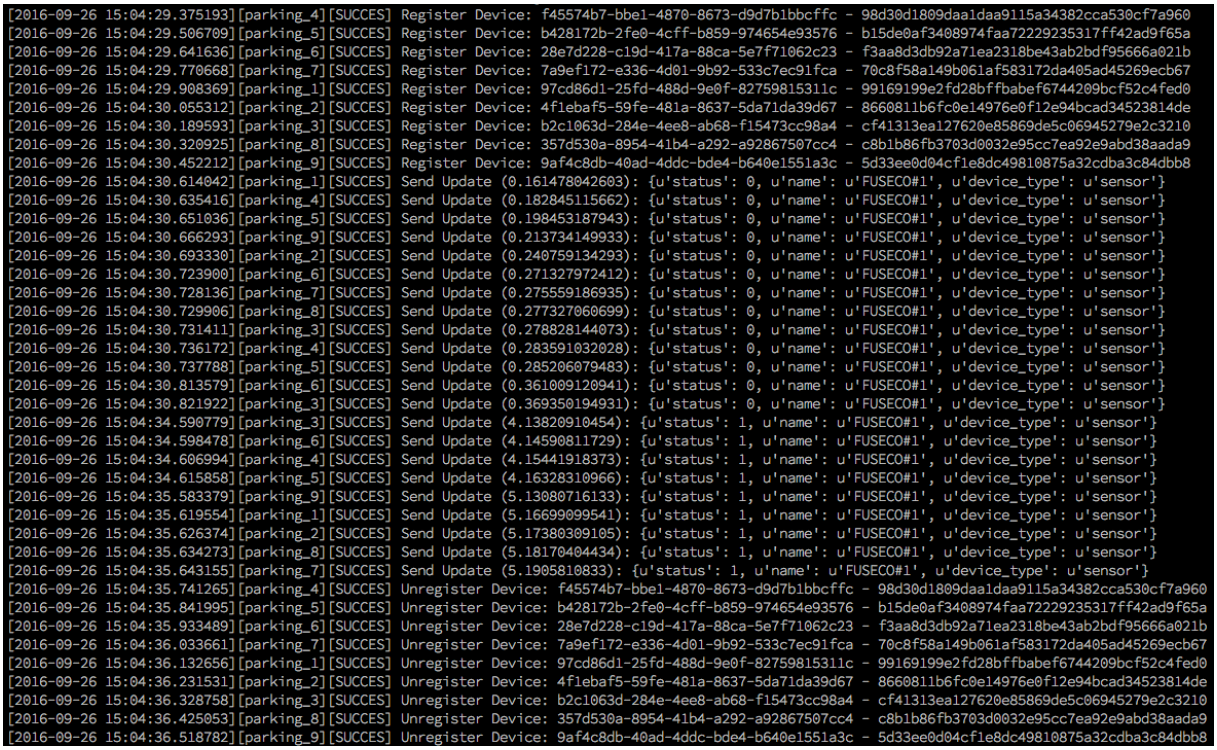

Figure 7 – Load Generator Experiment Log

Figure 7 shows a full log of an experiment ran with 9 parking sensors. At first all sensors are registered and getting their UUID/Token pairs from Meshblu. After that each sensor sends two updates. Finally, all sensors are unregistered

Although the three experiments are very similar, each one of them represents a crucial step of the experiments plan, because thanks to them it is possible to test:

- A communication broker, which was Meshblu;
- A communication protocol, which was HTTP;
- A communication standard, which is Citibrain property standard.

These tests were iteratively performed from a small use case to ensure the proper functioning using only one device with random data for the messages (Experiment 1), until a larger use case where it was possible to test the communication in the way meant to be used (Experiment 2 and 3).

Although, these experiments were only used to verify some of the requirements referred in the deliverable D1.1, in the future is intended to check the remaining requirements by improving these tests or creating new ones.

3 Infrastructure Description

After making Ubiwhere's MBaaS capable of integrating with the testing facilities, here we present the requirements for enabling and seamlessly running the experiments and tests to mature Ubiwhere's product and make it ready for the market. For these phases, the authors of this document are assuming that all the requirements above are taken into consideration for the experimentation and testing phases since the MBaaS relies on the communication protocols and open standards from the previous chapter for the automatic execution of the tests described here.

3.1 FIT IoT-LAB

3.1.1 IoT-LAB User experience

IoT-LAB provides a comprehensive set of tools to deploy, run and manage experiments on the testbed. Web-based and command-line tools are available, along with a public RESTful API. The command-line tools (e.g. CLI tools) can be used interactively or by script, on the platform's ssh front-ends or on the user's computer as describe in deliverable D1.2. In this way it's really easy to develop an experiment workflow and execute repeatable experiments on the testbed.

Moreover, launching an experiment is fast and simple. Firstly, you must build your application with one or several firmwares. IoT-LAB offers full open support for embedded software development, ranging from direct access to node hardware and to embedded operating system (OS) level features. To submit an experiment, the user reserves one or more nodes, in one or more sites, for some duration. To each node are attached some features: mobile/fixed, location and radio chip. After reserving the nodes, the user can make additional configurations for the experiments, including whether to use any monitoring tools (power consumption, wireless sniffer), how the nodes should be powered (battery/mains), and specify the firmware image(s) to load on the nodes. The scheduler starts the experiment as soon as the nodes are available, or at a precise date, if specified.

Once an experiment is running, the user has full control over the nodes reserved and dedicated to him/her. Nodes can be reset, reconfigured, or reprogramed individually or globally. A user can control the experiment directly using CLI tools, quickly edit source code and build/deploy firmware and access a node's serial port on the ssh front-end. A wealth of documentation is available IoT-LAB Website, including tutorials and wiki.

3.1.2 IoT-LAB Embers tools

For the Embers project, we have developed a SDK tool (e.g. Python scripts) which encapsulate all steps needed for running Experiment 2 test phase. This SDK tool is based on IoT-LAB software python libraries available for a user and hosting on the ssh front-end. Firstly, you should reserve nodes and submit experiments on the testbed with IoT-LAB CLI tools. You can choose a specific topology on IoT-LAB sites, a duration and the number of nodes. Of course it's possible to automatize the experiment submission and launch many experiments at the same time on the testbed. In this manner a user should easily benchmark and setup different experiment's configuration like increasing the number of nodes. Each experiment submitted on the testbed can be considered as a unit test case.

Once reserved, the user can deploy firmware onto the nodes. We provide a default firmware example in charge of reading sensors data like environmental temperature, luminosity and pressure. It's also possible to generate and simulate traffic events as mentioned in deliverable D1.1. This firmware has been written with an IoT-LAB open-source port of embedded operating system FreeRTOS. This firmware is configurable and wait by the node's serial port access a configuration where you can specify the sensors measure frequency. It's expected during the project that we will provide and test different firmware implementations with popular IoT embedded operating systems (e.g. Contiki, RIOT, OpenWSN) maintained by the IoT-LAB team. The next step is to run a main script of the SDK tools on the ssh front-end which acting as an IoT-LAB M2M gateway. Thanks to the IoT-LAB aggregator python library which aggregates all experiment node's serial port you could send the measurement configuration to the nodes and received all measurement data by the same way. With the consumer-producer paradigm the IoT-LAB gateway should send the measurement data to the Meshblu device broker device. At the moment we have implemented only a HTTP client for the Meshblu gateway but it's expected that we will test other protocols like CoAP or MQTT to support several communication protocols. In this scenario the IoT-LAB gateway should be seen as a proxy between the IoT-LAB experiment nodes and the Meshblu device broker.

Moreover, we will conduct different M2M gateway and nodes deployment. In addition of the "Point to Point" network scenario described above we will deploy meshed network with wireless radio communication between the IoT-LAB M2M gateway and the experiment nodes. For this purpose, we will write new firmwares with

6LoWPAN and RPL routing protocol support and improve the IoT-LAB Embers SDK tools to take account this new case.

3.2 FUSECO Playground

3.2.1 Load Generator

The Load Generator is a tool to emulate virtual sensors sending data in a configurable session. The two main ideas behind this tool are to allow benchmarking of the EMBERS-Platform and provide a simple interface for third party developers to test their applications. Therefore, it is designed as a modular tool, which can be integrated into different environments. For instance, it is possible to run the Load Generator as a standalone tool communicating with a Device Broker (e.g. Meshblu) or integrated to a M2M Platform like OpenMTC. Additionally, the Load Generator can be deployed as a Docker image or by using OpenStack. This way it scales well especially considering benchmarks. In terms of simplicity the Load Generator provides a RESTful API via HTTP and is using the JSON data format to configure benchmarks and experiments. Such a JSON object enables a user to configure the following parameters:

- Device Broker (e.g. the EMBERS Meshblu instance)
- Protocol (e.g. HTTP)
- Duration of the experiment
- Virtual sensor types
    - Amount of emulated sensors
    - Interval of status updates
    - Random Offset to the interval
    - Data
        - Device (e.g. the Meshblu Gateway)
        - Payload

With the payload parameter it is possible to define multiple data fields like temperature, humidity or anything else. In regard to simplicity it is not necessary to provide the concrete data of every data field. The concrete data can be generated in different configurable fashions. Of course it is possible to define manual data, this is done by simply define a value for every timestamp you want a specific status update of a configured virtual sensor. But furthermore it is possible to define the probabilities of specific sensor statuses. The Load Generator will afterwards calculate the concrete data in terms of the configured probabilities. Additionally, it is possible to provide just a few timestamp/value pairs. The Load Generator is able to interpolate the remaining values by using linear interpolation.

Consequently, it is possible to define arbitrarily different sensor types (e.g. parking sensors, traffic sensors, temperature sensors, multipurpose sensors) avoiding high efforts for configuration.

The discussed configuration in the JSON data format can be sent to the Load Generator by using a RESTful HTTP interface. There will be a path for every experimenter containing all his experiments. The API of the Load Generator provides different data on an experiment. At first there is the configuration provided by the user. Next the user is able to see, which data was generated if the user configured the generators discussed above. If the experimenter is unpleased with the generated values, it is possible to regenerate them. Additionally, the experimenter is able to see the experiment sequence. This will show at which time the Load Generator will send which data for every virtual sensor. The experimenter is able to adjust every value of the experiment sequence if necessary. It is even possible for an experimenter to provide just such an experiment sequence skipping the prior configuration steps. This becomes especially meaningful if an experimenter wants to replay historical data sets of his own. After converting his own data to the JSON experiment sequence format of the Load Generator, he is able to do so.

The experiment can be started at any time via the RESTful API. After the experiment is finished the experimenter is able to see results like successfully sent status updates or the average delay of those status updates.

3.2.1.1 Load Generator API

Following you can find some examples for typical API calls assuming that the Load Generator is running on localhost and port 5000. The examples using an admin user with the credentials: “admin:secret”. Examples of a configuration.json file is provided in the Annex.

User API:

- Show all configured users:
  GET http://admin:secret@127.0.0.1:5000/
- Create a new user:
  POST http://admin:secret@127.0.0.1:5000/<USERID>
- Delete a user:
  DELETE http://admin:secret@127.0.0.1:5000/<USERID>

User Experiments API:

- Show all experiments of a user:
  GET http://admin:secret@127.0.0.1:5000/<USERID>
- Create a new experiment:
  POST http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>
- Delete an experiment:
  DELETE http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>

User Experiments Configuration API:

- Upload an experiment configuration:
  POST http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/config
  Headers: Content-Type: application/json
  Body: <configuration.json>
- Show an experiment configuration:
  GET http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/config

User Experiment control API:

- Regenerate random values for experiment:
  POST http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/regenerate
- Upload an experiment sequence:
  POST http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/sequence
  Headers: Content-Type: application/json
  Body: <sequence.json>
- Show an experiment sequence:
  GET http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/sequence
- Start an experiment:
  POST http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/start

- Check status of an experiment:
  GET http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/status
- Show results of an experiment:
  GET http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/results
- Show full experiment log:
  GET http://admin:secret@127.0.0.1:5000/<USERID>/<experimentID>/log

4 Conclusions

This deliverable summarizes the first experiments designed to test the MBaaS in order to fulfil the projects two primary goals. Three experiments have been briefly described, which focus on different aspects of the platform and accordingly the tools and hardware of the FIRE+ testbeds.

The first experiment tests the basic ability of the platform to register new devices and the communication between them and the MBaaS.

The second experiment makes use of the tools provided by IoT-Lab and tests the ability of the platform to interconnect with real devices and emulated events.

The third experiment describes the use of the Load Generator at the FUSECO Playground.

The tools implemented by the two infrastructures are described in more detail in Section 3.

.

5 References

[1] Citibrain, “Citibrain,” [Online]. Available: http://www.citibrain.com.

[2] Telefonica, “FI-WARE NGSI Open RESTful API Specification,” [Online]. Available: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Open_RESTful_API_Specification.

[3] ETSI, “ETSI - Internet of Things,” [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/internet-of-things.

[4] oneM2M, “oneM2M,” [Online]. Available: http://www.onem2m.org/.

[5] OMA, “OMA LwM2M,” [Online]. Available: http://openmobilealliance.hs-sites.com/lightweight-m2m-specification-from-oma.

[6] European Commission, “FIRE+,” [Online]. Available: http://www.ict-fire.eu/home.html.

[7] INRIA, “FIT IoT-Lab,” [Online]. Available: https://www.iot-lab.info.

[8] FRAUNHOFER FOKUS, “FUSECO Playground,” [Online]. Available: https://www.fokus.fraunhofer.de/go/en/fokus_testbeds/fuseco_playground.

[9] Octoblu, “Meshblu,” [Online]. Available: https://Meshblu.readme.io/.

[10] TZI, “CoAP,” [Online]. Available: http://coap.technology/.

[11] MQTT, “MQTT,” [Online]. Available: http://mqtt.org/.

[12] OMA, “LwM2M,” [Online]. Available: http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0.

[13] FIWARE, “IDAS - Backend Device Management,” [Online]. Available: http://catalogue.fiware.org/enablers/backend-device-management-idas.

[14] FRAUNHOFER FOKUS, “openMTC,” [Online]. Available: http://www.openmtc.org/.

Annex

Configuration example


```
{
  "device_broker": "Meshblu",
  "device_broker_provider": "embers",
  "duration": 20,
  "offset": true,
  "protocol": "http",
  "sensors": {
    "environment": {
      "amount": 5,
      "data": {
        "devices": "b4518108-57c9-412a-a664-793414025275",
        "payload": {
          "carbon_dioxide": {
            "data": {
              "0.0": 300,
              "20": 400
            },
            "generator": "interpol_1d"
          },
          "carbon_monoxide": {
            "data": 2000,
            "generator": "static"
          },
          "device_type": {
            "data": "sensor",
            "generator": "static"
          },
          "humidity": {
            "data": {
              "0.0": 40,
              "10": 70,
              "20": 50
            },
            "generator": "interpol_1d"
          },
          "light": {
            "data": {
              "0.0": 10,
              "10": 500,
              "20": 10
            },
            "generator": "interpol_1d"
          },
          "name": {
            "data": "FUSECO#33",
            "generator": "static"
          },
          "nitrogen_dioxide": {
            "data": 0.5,
            "generator": "static"
          },
          "noise_level": {
            "data": {
              "0.0": 30,
```

```
          "20": 10,
          "3.0": 100,
          "8.0": 20
        },
        "generator": "interpol_1d"
      },
      "ozone": {
        "data": 100,
        "generator": "static"
      },
      "particles": {
        "data": 0.3,
        "generator": "static"
      },
      "precipitation": {
        "data": {
          "0.0": 40,
          "20": 60
        },
        "generator": "interpol_1d"
      },
      "solar_radiation": {
        "data": 2000,
        "generator": "static"
      },
      "temperature": {
        "data": {
          "0.0": 18,
          "20": 5,
          "3.0": 20,
          "8.0": 30
        },
        "generator": "interpol_1d"
      },
      "vocs": {
        "data": 100,
        "generator": "static"
      },
      "wind_speed": {
        "data": {
          "0.0": 23,
          "20": 40
        },
        "generator": "interpol_1d"
      }
    }
  },
  "interval": 2
},
"parking": {
  "amount": 20,
  "data": {
    "devices": "b4518108-57c9-412a-a664-793414025275",
    "payload": {
      "device_type": {
        "data": "sensor",
        "generator": "static"
      },
      "name": {
```

```
            "data": "FUSECO#1",
            "generator": "static"
          },
          "status": {
            "data": [
              {
                "0.0": 0,
                "5.0": 1
              },
              {
                "0.1": 0,
                "4.0": 1
              }
            ],
            "generator": "manual"
          }
        }
      },
      "interval": 1
    }
  }
}
```