MONROE

Measuring Mobile Broadband Networks in Europe

H2020-ICT-11-2014
Project number: 644399

Deliverable D1.2
System Design and Prototype implementation

Editor(s):	Vincenzo Mancuso
Contributor(s):	Rafael Garcia Leiva, Özgü Alay, Andra Lutu, Audun Fosselie Hansen, Thomas Hirsch, Stefan Alfredsson, Roberto Monno, Marco Mellia, Ali Safari Khatouni
Work Package:	1 / System Design and Software Implementation
Revision:	0.1
Date:	August 3, 2016
Deliverable type:	DEM (Prototype)
Dissemination level:	Confidential, only for members of the consortium (including the Commission Services)

European Commission
Horizon 2020
European Union funding
for Research & Innovation

Abstract

This document presents an overview of the MONROE system design and prototype implementation of the platform ready at M12. Specifically, the document describes different components of the platform, i.e.: (*i*) the user access portal, (*ii*) the scheduling system, (*iii*) the management and maintenance system, (*iv*) the software running on the MONROE node (core components and experiments), (*v*) the repositories and data importer, (*vi*) the central database, and (*vii*) and the online visualization tool.

Keywords

Prototype, Experiments, Metadata, Fed4FIRE, Scheduling, DB, Visualization.

Participant organisation name	**Short name**
SIMULA RESEARCH LABORATORY AS *(Coordinator)*	SRL
CELERWAY COMMUNICATION AS	CWY
TELENOR ASA	Telenor
NEXTWORKS	NXW
FUNDACION IMDEA NETWORKS	IMDEA
KARLSTADS UNIVERSITET	KaU
POLITECNICO DI TORINO	POLITO

Contents

1 Introduction

The main goal of the MONROE project is to build and operate a unique platform in the world for measurements and experiments in operational mobile networks. MONROE aims to design and operate the first European transnational open platform for independent, multi-homed, large-scale monitoring and assessment of performance of mobile broadband networks in heterogeneous environments. In this document, we will describe the main software components of the MONROE platform and the status of the prototype implementation in the first year of the project.

We summarize the system design for the MONROE in Figure 1. The system is composed of seven main parts: (*i*) user access system, (*ii*) scheduling system, (*iii*) management and maintenance system with inventory, (*iv*) MONROE node modules, (*v*) remote repositories and data importer, (*vi*) database, and (*vii*) visualization.

All the blocks in the figure are color-coded as follows:

- Blue: These are the components that are provided as background to MONROE (from Simula's Nornet testbed and Celerway routers) and will not be made open source.
- Red: These are the components developed during the project and will be made open source.
- Green: These are the components that the external users are responsible for developing. Templates will be provided as open source.

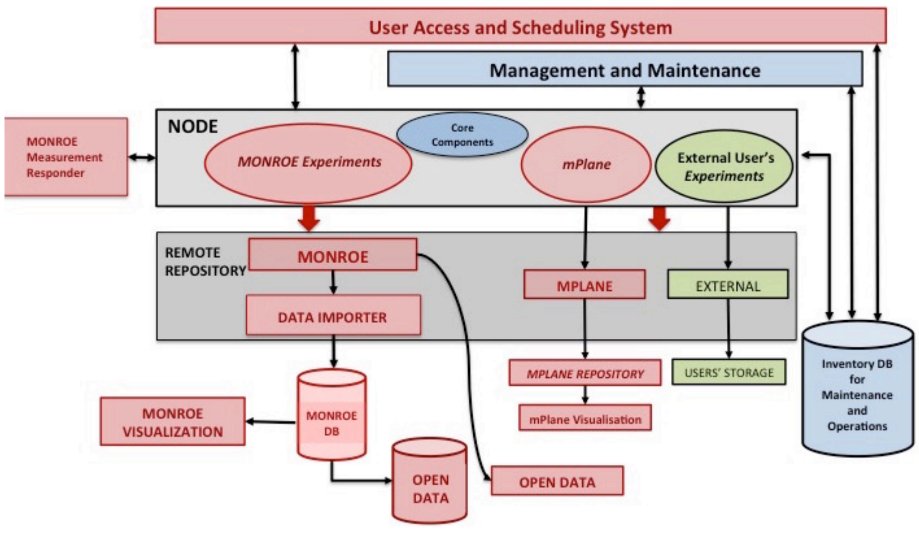


Figure 1: Building Blocks of MONROE System.

The following sections provide an overview of each component focusing on the details of the design and the implementation. Also the communication among the modules is briefly discussed.

2 User Access and Scheduling System

User Access to the experimental platform will be provided via a web-based MONROE Experimenters Portal that enables users to schedule and run new experiments. The portal allows to access the MONROE scheduler, which is in charge of setting up the experiments without requiring the users to access (log in) the nodes.

Since MONROE is federated with Fed4FIRE in order to build a large-scale, distributed and heterogeneous platform, authentication and provisioning of resources follows the Fed4FIRE specifications.

In the following subsections, we will provide details on MONROE's federation with FED4FIRE, user authentication, experimenters portal and scheduler.

2.1 MONROE as a Fed4FIRE federated project

Fed4FIRE Portal is a common and well-known tool where registered users can select and access an available testbed, in our case the MONROE platform. Fed4FIRE Portal is powered by MySlice software[1] and offers a directory of all FIRE testbeds, tools and links to project websites. Basically, it acts as an experimentation tool and bridge to experiment control tools.

2.1.1 User Workflow

The workflow that the experimenters must follow when they want to join MONROE and run their experiments consists of the steps described in what follows:

- *Read Documentation*: The user must be familiar with the terminology and tools of the Fed4FIRE federation and, in particular, with the MONROE testbed.
- *Get a valid account*: The user has to apply for a valid Fed4FIRE account and download the corresponding certificate. All account must be associated to an existing MONROE project.
- *Perform the MONROE tutorial*: The user should follow and attempt the MONROE tutorial, which describes those elements that are specific to the MONROE testbed, including the AngularJS client developed in the project to schedule the experiments.
- *Provision Resources*: The user must ask for resources to run the experiment to the MONROE scheduler and provide the experiment scripts. This step is done through the user access client presented in this document in the Section 2.3.
- *Test phase*: The scheduled experiments are run by the nodes without the manual intervention of the experimenters. Results are collected on the nodes involved in the test. Experimental logs and data specifically generated by the scripts of the experiments are transferred to a MONROE repository and made accessible to the experimenters.

The next sections describe more in detail each of these steps.

2.1.2 Documentation

The very first step for each new user (somebody that wants to run an experiment in the MONROE testbed) is to read the introductory documentation. In fact, the user must be familiar with the terminology and tools used in the Fed4FIRE federation and, in particular, with the MONROE testbed. That would require to read:

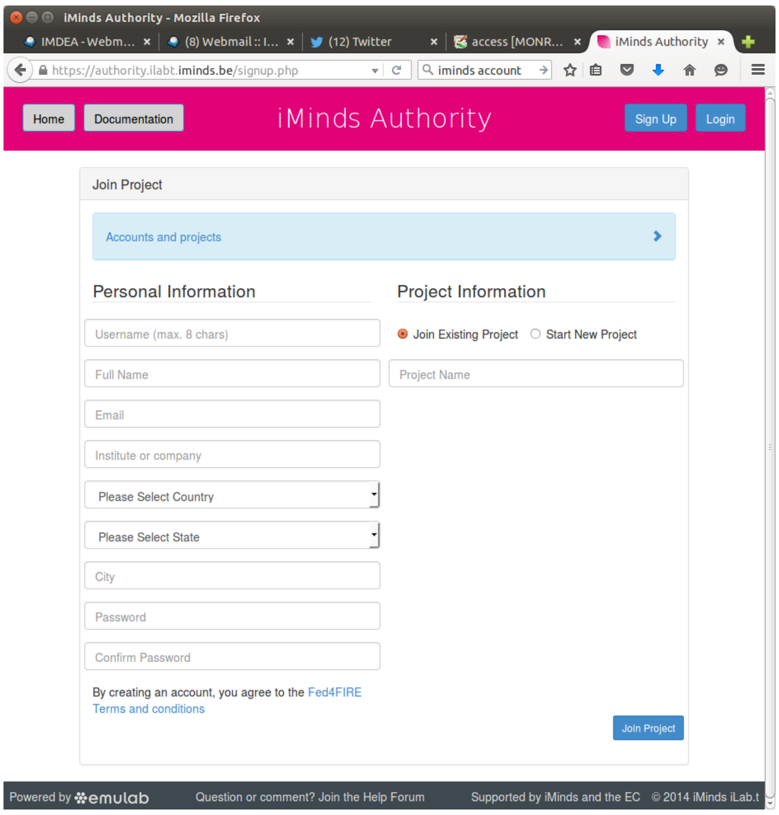
- The available documentation of Fed4FIRE, that describes the federation of testbeds as a generic environment.
- The specific documentation of MONROE contains the specific details of the MONROE testbed that every user should know, in terms of project goals, hardware description, experiment requirements, and so on.

[1] MySlice: http://myslice.info

2.1.3 Accounts

Once the user is familiar with the concepts and tools used in Fed4FIRE, and with the particular idiosyncrasies of the MONROE testbed, he/she must open a valid Fed4FIRE account and (optionally) download the authentication and authorization certificate. How to create an account and how to download the certificate is explained at the Fed4FIRE introductory documentation. Please note that the user must specify an already existing MONROE project, or alternatively, create a new one. For more information about how Fed4FIRE accounts work, and how MONROE projects are created, please refer to the corresponding section below.

As an example, Figure 2 shows how to create an account using the iMinds certification authority:



Join Project

Accounts and projects

Personal Information

Username (max. 8 chars)

Full Name

Email

Institute or company

Please Select Country

Please Select State

City

Password

Confirm Password

By creating an account, you agree to the Fed4FIRE Terms and conditions

Project Information

[✓ Join Existing Project] Start New Project

Project Name

Join Project

Powered by emulab   Question or comment? Join the Help Forum   Supported by iMinds and the EC   © 2014 iMinds iLab.t

Figure 2: iMinds Authority.

2.1.4 MONROE Tutorial

The MONROE tutorial provides an introduction to those elements that are specific to the MONROE testbed. The tutorial also provides links to the official MONROE documentation. Those users that plan to run experiments in MONROE testbed should be familiar with the contents of the MONROE tutorial. The tutorial provides basic information about MONROE specifics, among which:

- Specifications on the node hardware
- Specifications on the node software
- Experiments allowed in the testbed
- How to create a new MONROE project

- How to join an existing project
- How to provision resources
- How to use the MONROE client to schedule experiments
- How to access to wireless network interfaces
- How to design a new experiment
- How to run an experiment
- How to collect experiment datasets

2.1.5 Provision of Resources

Once the user has a valid MONROE account, he/she is familiar with the MONROE testbed (from a technical point of view and its policies), the next step consists in asking for resources (nodes, disk, bandwidth, etc.) to the MONROE scheduler. The resources assigned to the user are the maximum number of resources that the user can use simultaneously for all of his/her experiments at a given time. To guarantee a minimum amount of resources for a specific experiment, the experimenter has to use the MONROE scheduler (see Section 2.4), which can be accessed through an AngularJS client developed in the project (see Section 2.3). With the above, the experimenter can reserve the resources up to the limit granted to him/her by the MONROE consortium. Moreover, since the MONROE scheduler supports GENI/FED4FIRE APIs, the users will be able to request the set up of their experiments connecting directly to the scheduler, without using the MONROE user client.

2.2 User Authentication

MONROE will adhere to the Fed4FIRE Authentication and Authorization schemas. In this section, we describe the Fed4FIRE AAA policies and procedures, and how they will be adapted to the MONROE project.

2.2.1 Fed4FIRE AAA

A federation is a collection of testbeds (or "islands") that share and trust the same certification authorities and user certificates. Fed4FIRE realizes a federation of a large number of wired, wireless and openflow-based testbeds principally located in Europe. Each island manages its resources using dedicated tools and can decide which kind of certificates (and from which authorities) it wants to accept. In this context, Fed4FIRE works with X.509 certificates to authenticate and authorize experimenters (users) on its testbeds. The authority which provides valid certificates in the Fed4FIRE federation is located at the iMinds infrastructure.

In order to add a new testbed to the Fed4FIRE federation, all the PIs (Principal Investigators) of the already existing testbeds must agree on the new membership.

The certification authority has the concept of Projects which bundle multiple users. Any user can requests for the creation of a new project, but it must be authorized by the Fed4FIRE administrators. In a Project, the PIs can approve new experimenters for that particular project, without Fed4FIRE administrators needing to approve this. User can get invitations to join to a particular project. Testbed resources assigned to each project are decided and managed by the testbed administrators.

In the end, the user will have a file on his local PC which contains the certificate information and that can be used to access to testbed facilities.

2.2.2 MONROE AAA

MONROE shares and trusts the certificates generated by the iMinds authority, and therefore, is a member of the Fed4FIRE federation. Other certification authorities, and other federations, like GENI, are not supported by MONROE.

Each partner of MONROE will have its own private project inside Fed4FIRE, where the PI of the partner will be the PI of the project (a responsibility that could be delegated if needed). All those entities granted with projects trough the open calls will have their own private projects. Other external institutions, non-funded, could have their own private projects as well, upon request and approval by the MONROE Project Board. On the other hand, individual researchers cannot join the MONROE testbed, as all the users must belong to, at least, one project. The PIs can easily invite new users and grant access to their respective projects offering the available resources which are managed in a project basis by the MONROE administrators.

It is worth noting that all the project functions and operations in MONROE are based on the user certificates. For example, if a user wants to make a reservation for a resource, he/she needs his/her certificate; if the user wants to run an experiment, he/she needs his/her certificate; if a user wants to download an experiment data, he/she needs his/her certificate. Some information about MONROE testbed could be made public trough the use of a Visualization Web page (access to this page does not requires a certificate).

2.3 The Experimenters Portal (MONROE User Access Client)

The very first screen that appears when the application is loaded is the Login screen (Figure 3), where the user provides his credentials to authenticate the application. The application allows to upload a certificate from file, and authenticate password-protected certificates. If the user clicks the "remember me button", the credential will be installed in the browser. If the user has the credential already installed in the browser and is not password protected, or password has not expired, this window will be not shown.

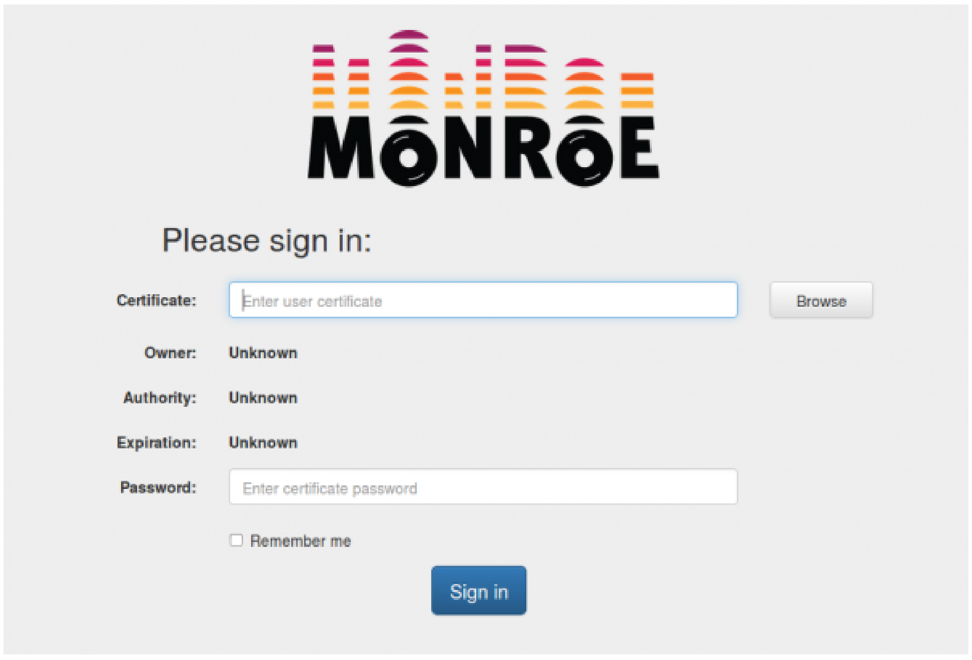
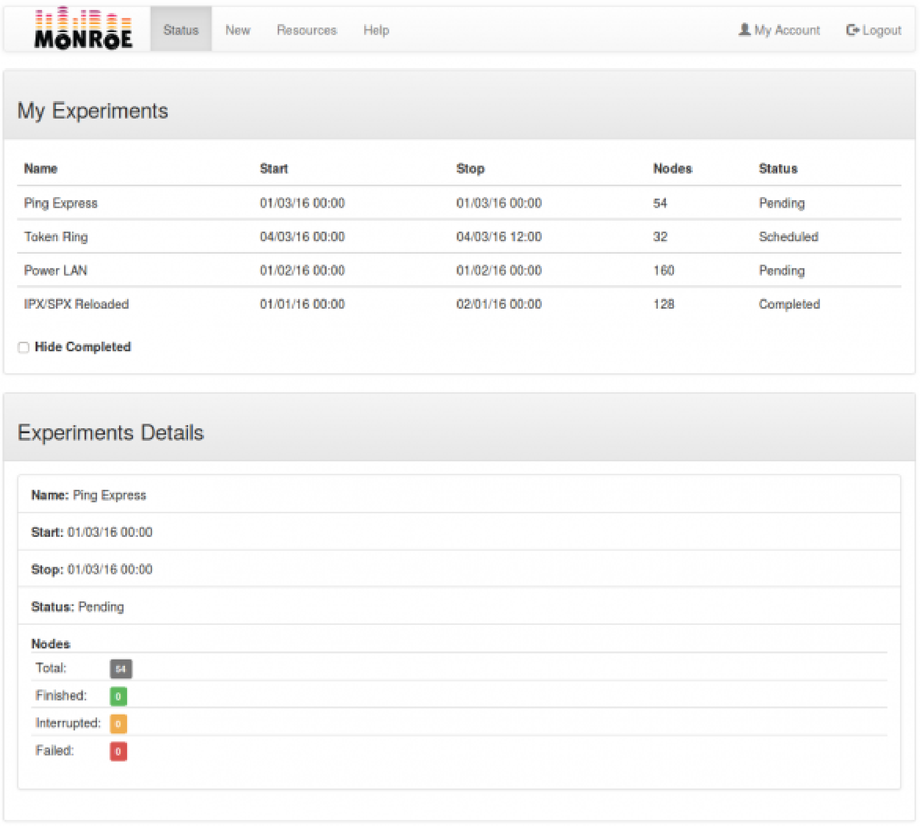


Figure 3: Portal login.

After successfully authenticated, it is shown a screen with the current status of all the experiments of the user (Figure 4). By clicking in any row of the table, the details of the experiment are shown.



My Experiments

Name	Start	Stop	Nodes	Status
Ping Express	01/03/16 00:00	01/03/16 00:00	54	Pending
Token Ring	04/03/16 00:00	04/03/16 12:00	32	Scheduled
Power LAN	01/02/16 00:00	01/02/16 00:00	160	Pending
IPX/SPX Reloaded	01/01/16 00:00	02/01/16 00:00	128	Completed

☐ Hide Completed

Experiments Details

Name: Ping Express

Start: 01/03/16 00:00

Stop: 01/03/16 00:00

Status: Pending

Nodes

Total:	54
Finished:	0
Interrupted:	0
Failed:	0

Figure 4: Experiment status.

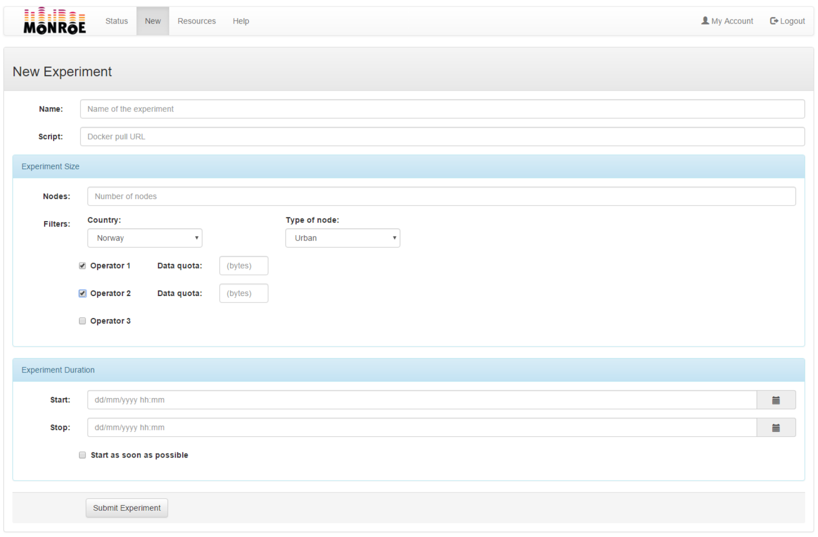
In the New experiment tab (Figure 5), the user can create a new experiment, by providing the experiment details. The basic experiment details is the name of the experiment and the docker script to run the experiment. In the Experiment Size group, the user specifies the number of nodes required to run the experiment, and the desired characteristics of those nodes using filters that allows to select, e.g., the location of the nodes to use in the experiments, their hardware/software version, static or mobile nodes, testing nodes for preliminary/debugging tests, etc. Furthermore, the user can select the operator of interest and then define the maximum amount of data to be transferred per experiment over that interface/operator. This data limit is enforced during the experiments in order to avoid exceeding the mobile data quotas. In the Experiment Duration the user specifies the duration of the experiment. The duration can be specified by providing a starting and stopping date-time (either manually or using a calendar tool), or by clicking the "as-soon-as-possible" check box.

The Resources tab (Figure 6) allows the users to query all the existing resources in the Monroe testbed and their time availability for experimenters, using multiple filters if required.

Finally, the help page contains a short description of how to use the application and provides a pointer to the MONROE user guide and to the project webpage.

2.4 MONROE Scheduler

The scheduler ensures that there are no conflicts between users when running their experiments and, assigns a time-slot and node resources to each user. MONROE provides the necessary interface and tools to control this process.



New Experiment

Name: Name of the experiment

Script: Docker pull URL

Experiment Size

Nodes: Number of nodes

Filters: Country: Norway — Type of node: Urban

[✓] Operator 1 — Data quota: (bytes)

[✓] Operator 2 — Data quota: (bytes)

[ ] Operator 3

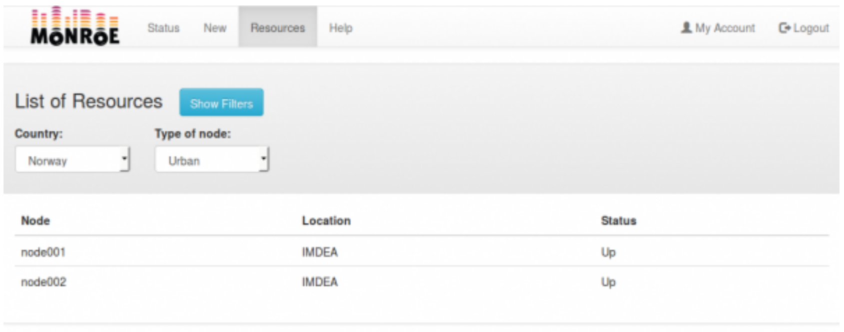
Experiment Duration

Start: dd/mm/yyyy hh:mm

Stop: dd/mm/yyyy hh:mm

[ ] Start as soon as possible

Figure 5: New experiment.



List of Resources

Country: Norway — Type of node: Urban

Node	Location	Status
node001	IMDEA	Up
node002	IMDEA	Up

Figure 6: Resources availability.

2.4.1 Policies and Procedures

We have defined the scheduling procedures and policies to guide the MONROE experimentation:

- The scheduler allows booking of fixed time slots for each measurement experiment.
- Priority is defined by the first-come first-serve principle, while the consortium will monitor fairness.
- If an experiment is marked as *exclusive*, only one experiment may run at a given time on a node.
- If an experiment is marked as *active*, one such experiment may run at a given time on a node, while allowing passive experiments.
- If an experiment is marked as *passive*, a given number of such experiments may run at a time. No traffic may be generated by the experiment.
- User experiments may be scheduled as periodic, continuous, or one-time.
- Only experiments for which a time slot has been booked in advance may be run.
- Nodes may be of different types (static, mobile, urban, rural, certain country, etc...) defined by the MONROE project. Booking requests can select to use or reject these filters.

- A booking over several nodes or several time periods is treated as atomic. I.e. if one of the booking periods or nodes is unavailable, the entire booking is rejected. Several bookings over different nodes or time periods may be linked to an atomic unit.

In order to determine the resource requirements, each experiment first has to be scheduled and run on the testing nodes. For this, the identical system image and parameters of the experiment as provided by the user will be loaded onto a testing node. The experiment will be run, and its resource usage will be monitored. If the usage is within defined constraints, the system image will be approved by means of a cryptographic signature. Only then, the experiment image is cleared to be scheduled on regular nodes. This is both a safety measure and a service to the experimenter, helping to stay within the limits of the MBB data quotas.

The scheduling process on the node defines three actions: (*i*) deployment, (*ii*) start and (*iii*) stop of the experiment. The deployment step may take place at any time before the scheduled start time, and should finish before the experiment is started. In this step, the experiment image is loaded onto the node and requested resources are reserved. During the start process, resource quotas are set and a container system where experiments will be run (see Section 4.2) is started with the experiment image. The stop action will notify the experiment of its impeding shutdown, then remove the container after a short grace period. Measurement results may be stored on disk, and will be transferred during and after the termination of the experiment as connectivity allows.

2.4.2 Implementation

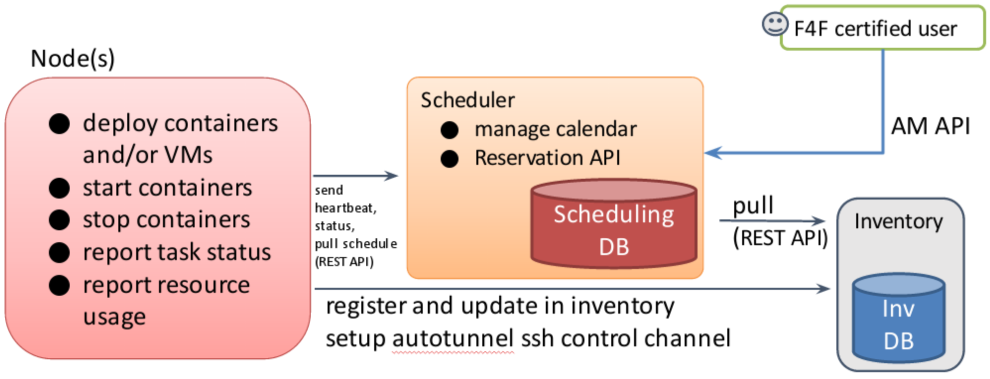


Figure 7: Scheduling system.

In a first phase, we investigated existing task schedulers, both from the Fed4FIRE initiative and from the area of cluster computing. The most promising candidates were OMF on the one side, and SLURM[2] and FLEET[3] on the other. However, all of these schedulers assume a high degree of connectivity between the nodes and the scheduling system, and often an internal addressing (VPN) within the testbed. In our use case, we assume nodes to be available independent of short-time loss of connectivity, while they may contact the scheduler from different addresses in their multi-homing setup, possibly with provider-dependent modifications and filters.

We decided therefore to implement a low-connectivity scheduling system which takes these assumptions into account. It consists of two components - the *scheduling server* running in a central, well-known location

[2]http://slurm.schedmd.com/
[3]https://coreos.com/using-coreos/clustering/

and the *scheduling client* running on the nodes (Figure 7).

The scheduling server

- takes care of the experiment schedule and resolves conflicts
- assigns roles to authenticated users
- provides a REST API to users and nodes to query and edit scheduling status
- provides an XML-RPC API compatible with the Fed4FIRE AM API definition

The scheduling client

- sends a regular heartbeat and status to the scheduling server
- fetches the experiment schedule for the current node
- downloads, deploys, starts and stops scheduled experiments

Authentication to the server is based on X.509 client certificates. Users, administrators and nodes all authenticate using this mechanism and use the same scheduling API. By importing the Fed4FIRE certification authority certificate, users may authenticate using their Fed4FIRE credentials.

Due to the connectivity constraints especially of mobile nodes, deployment of experiments on the node is not immediate. Download and deployment of experiments will take place as early as possible within the constraints of available space on the node. The node will report a successful deployment to the scheduler and schedule the start and stop times for the experiment container internally. Changes in the schedule are propagated to the node whenever possible.

3 Management and Maintenance

Management and Maintenance System will be used mostly by the operations team to manage and maintain the MONROE testbed and has mainly 4 different components:

- **Inventory** keeps all the information required for operations and maintenance. This involves the status of each node, status of different connections, location of the nodes, etc. This data is kept in a relational database and an API is provided to retrieve data from the inventory DB.
- **Software Configuration Agent** talks to inventory to retrieve node specific configuration and grabs configuration files and job instructions from Software Configuration Server.
- **Monitoring Agent** monitors and reports the health of the system including logging, performance monitoring, self checks for services etc. that extends beyond basic watchdog functionality.
- **Status Visualization and Management Interface** is the main point for editing node details and tracking availability for operations teams. Log and monitoring systems will be available through this portal.

Since these modules mostly consider the operations and maintenance, we have inherited them from the current Nornet Management and Maintenance System and provided an interface to MONROE.

4 MONROE Node

The Node represents all the software running on the MONROE node as illustrated in Figure 8, and has two main parts: Core Components and Experiments.

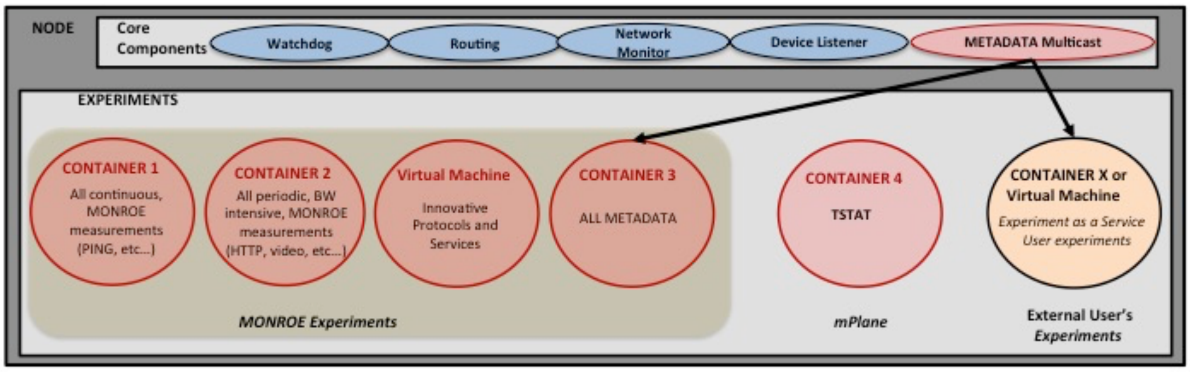


Figure 8: SW running on the node.

4.1 Core Components

The **Core System Components** consist of the main software running on the node and make sure that the node is up and running. The MONROE node includes the following software modules:

- *System-wide Watchdog* is responsible for ensuring all core services are running. It supports restart of the devices in the case of a problem.
- *Routing Deamon* is responsible for routing which is required when connected to multiple interfaces simultaneously.
- *Device Listener* is responsible for detecting and connecting to the 3G/4G modems and WiFi dongles. The current version supports all tested modems and dongles.
- *Network monitor* monitors connectivity of the different interfaces. In order to do this, it performs some of the core measurements (ICMP, DNS and time to first byte). In addition, the tool supports IPv4/v6 and makes sure that the route for unbound traffic goes through a *good* interface.
- *Metadata Multicast* is responsible to multicast the metadata such as node status, connection technology and GPS. Different experiments can subscribe to specific relevant information, for example to correlate GPS/operator with a throughput measurement. This metadata is using a multicast publishing system (Zero MQ) to distribute this information to subscribers. All metadata is also collected and stored itself to enable post-processing and -correlation with experiments.

4.2 Experiments

The **Experiments** constitute all the measurements and experiments running on the node. These include the experiments run by consortium as well as the external users. The node runs applications inside light-weight virtualization containers to provide isolation and resource management. For experiments that require kernel changes, the node will also support full virutalization through Virtual Machines (VMs).

Containers and VMs will be used to initialize, execute and finalize the experiments. In Figure 8, Containers 1-4 are run and controlled by the consortium to determine the key MBB parameters. Container 1 will collect all the continous MONROE measurements such as ping test. Container 2 is allocated to run periodic bandwidth intensive applications such as web bulk upload and download, and video streaming. Since these experiments are expected to consume a large portion of the data quota, they cannot run continuously like the experiments in Container 1, otherwise the monthly data quota would be consumed in short time. The periods must also be scheduled such that no other experiment is running simultaneously, which would be affected by the high bandwidth utilization. Container 3 is responsible for collecting all the metadata. Container 4 is dedicated to `Tstat`. The experiments that will be run by the consortium on innovative protocols and services will be run in a VM. Container X and external user's VM will be used by the external users to run all other experiments.

Experiment data is produced and collected in different ways depending on the tools in use. This data (including the metadata) is then reformatted into JavaScript object notation (JSON) files. For example ping RTT experiments use text output logging from a ping command, which is reformatted into JSON file format. Using the same data format enables efficient reuse of software tools and processing. The JSON files are stored at a specific location in the file system and at periodic intervals (on the scale of 30-60 seconds), a data transfer container copies the collected JSON-files to a remote repository. If no Internet connection is available, files will be cached on the node until the next transmission attempt. At the repository only completely received files are processed.

The experiments will be run towards MONROE measurement Responders. For example, for the simple ping test or for the traceroute experiments, this responder will become the *echo server*. For different type of tests, the responder can be configured.

As an example, here we report the description of a typical traceroute experiment, for which we assume that channel quality and network information is constantly and automatically stored by other experiments, so we do not include such information in the log of this specific experiment.

Regarding the specific data fields of the traceroute experiment, we need to record the following:

- data ID: monroe.exp.traceroute
- Fields:
  - InterfaceName: local name of interface
  - TimeStampStart: time when the experiment started
  - TimeStampEnd: time when the experiment ended
  - HopCount: the total number of hops
  - Hop#Ip: the IP address of hop number #
  - Hop#Name: the name of hop number # (optional)
  - Hop#min: minimum delay of hop number #
  - Hop#average: average delay of hop number #
  - Hop#Max: maximum delay of hop number #
  - Hop#LossRate: the loss rate of hop number #

Note that right before the traceroute, one should generate a minimum amount of network traffic to make sure that the actual experiment is not affected by promotion delays. In the case of 3G the generated traffic should be enough to ensure that the interface is in Dedicated Channel (DCH) state at the beginning of the real experiment.

Prototype. In the prototype we have implemented Containers 1, 3, 4, while Container 2 with periodic measurements to be scheduled without overlap with external experimenters' tests, will be added later, before the external experimenters will have access to the platform. We have decided to run the ping test in container 1 and collect the connection type (e.g. 3G, 4G) and the signal strength that is represented by bars (e.g. 1

bar shows very poor quality where 5 bars show very good quality, similar to the ones we see on current smartphones) in Container 3. In Container 4, a light version of `Tstat` collects data.

Integration with mPlane. `Tstat` represents a special experiment case, since it is a tool developed in the frame of the mPlane project that has been integrated in the MONROE platform. Therefore, in the following subsection we quickly present mPlane's architecture and its integration with MONROE.

4.3 Tstat and mPlane in MONROE

4.3.1 mPlane architecture

One of the main challenges of mPlane is the integration of programmable probes with legacy probes all sharing a common standardized interface in order to realize a large-scale, distributed measurement layer.

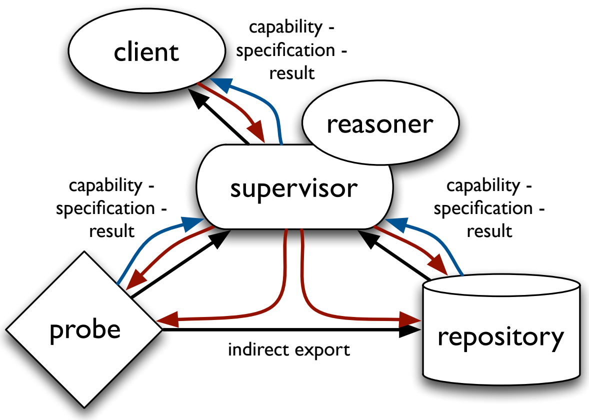

client
capability - specification - result
reasoner
supervisor
capability - specification - result
capability - specification - result
probe
indirect export
repository

Figure 9: The mPlane Architecture.

Figure 9 summarizes the mPlane architecture as defined in mPlane deliverable D1.3.[4] An mPlane infrastructure consists of a set of components meant to perform passive or active measurements, or to store and analyze the resulting data. Components can be roughly divided into probes and repositories. Probes perform raw measurements, and may pre-process them. Repositories store, correlate and analyze the measurements provided by multiple probes. These components are coordinated by a supervisor, which handles the mechanics of distributed measurement (e.g., component discovery, capabilities management, access control, etc.) as well as final analysis, correlation, and compilation of results from multiple repositories and probes.

A key feature of most troubleshooting workflows is iterative measurement. Interaction in mPlane begins from a set of component capabilities: on one hand, a supervisor collects the capabilities of the components it supervises, and presents capabilities to clients (e.g., representing measurements it can perform or queries it can answer with its components); on the other hand, a client selects some set of capabilities and sends a specification to the supervisor, i.e., a description of which measurement to perform, how, where, and when. The supervisor authenticates the client, checks its authorization to perform the measurements called for in the specification, and sends corresponding specifications to the appropriate components. Results returned contain all the parameters of the specification used to generate it, so that they are self-contained. This simplifies management in large-scale deployments, while reducing the amount of state that each component has to store while waiting for one of its specification to return. This iterative analysis, supported and auto-

[4] https://www.ict-mplane.eu/sites/default/files//public/public-page/public-deliverables/ /697mplane-d13.pdf

mated by an intelligent reasoner, is sorely missing in present measurement systems, and is one key element of the mPlane.

4.3.2 mPlane integration in the MONROE platform

To offer a service that may be useful for MONROE users in general, the consortium agreed to instrument each MONROE node with mPlane complaint passive probe, which, by simply analyzing packets sent by the node on any interface at any time, is able to extract measurements and to present them to any interested user. To this extent, each MONROE node becomes a mPlane probe, i.e., it runs `Tstat`, that is the mPlane enabled passive probe. Measurements produced by each node are then collected to a mPlane repository. As any mPlane platform, the mPlane Supervisor is in place to orchestrate all measurements, and a mPlane Reasoner configures and manages the system to automatically start, stop, restart each component.

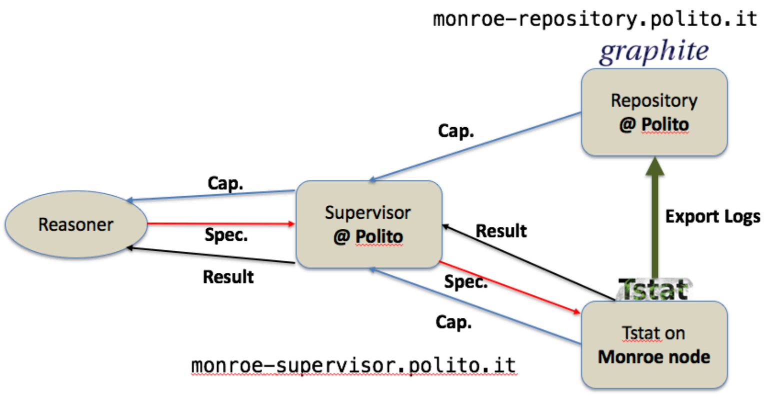


Figure 10: mPlane deployment in the MONROE context.

Figure 10 shows the workflow. The supervisor runs in POLITO premises, and it is directly reachable as `Monroe-supervisor.polito.it`. The repository, based on Graphite, runs in a separate server, reachable as `Monroe-repositorory.polito.it`. It offers its capabilities to the Supervisor (blue arrow). Each MONROE node runs `Tstat`, and offers its capabilities to the supervisor too. The Reasoner is a simple application which periodically queries the supervisor, collects capabilities, and then sends specification to start the `Tstat` probes (red arrows). Moreover, it can setup the exporting of results to the repository via a direct export. In the following we detail each component deployment and show some results.

4.3.3 Integration of `Tstat` probe

`Tstat` is a tool which, starting from standard software libraries, offers to network managers and researchers audit information about classic and novel performance indexes and statistical data about Internet traffic by simply observing packets exchanged by the node. Started as evolution of TCPtrace, `Tstat` analyzes either real-time captured packet traces, using either common PC hardware or more sophisticated ad-hoc cards such as the DAG cards. Beside live capture, `Tstat` can analyze previously recorded packet-level traces, supporting various dump formats, such as the one supported by the libpcap library, and many more.

`Tstat` rebuilds each TCP connection status looking at the packet header in the forward and backward packet flows. The bi-directionality of the TCP flow analysis allows the derivation of novel statistics (such as, for example, the congestion window size, out-of-sequence segments, duplicated segments, etc.) which are collected distinguishing both between clients and servers, (i.e., hosts that actively open a connection and hosts that reply to the connection request) and also identifying internal and external hosts (i.e., hosts located inside or outside the measurement point). To install `Tstat` on each MONROE node, we have configured a `Tstat` container following the docker platform supported by MONROE. Since `Tstat` needs to run on the node, and to observe all packets being exchanged via any interface, and being generated by any experimenters, we engineered a docker installation (we have also tested it with an lxc-container) in which `Tstat` is configured to capture traffic on all available interfaces (3G, 4G, or WiFi) in the MONROE node by listening to the "any" interface. When running, `Tstat` generates two sets of logs: a Round-Robin Database (RRD), and a detailed per-flow level log. The RRD logs collect and aggregate time series metrics at different levels, which are computed every 5 minutes. For instance, they account for the amount of received packets, or the amount of initiated TCP connections. The flow level logs are a set of Comma Separated Values files where each row corresponds to a different flow and each column is associated to a specific measure. For instance, for each TCP connection, a row details several metrics, like client IP address, port, server IP address, port, number of transmitted/received packets/bytes, average Round Trip Time, etc. A total of more than 120 metrics are extracted for each flow. When it is useful, the columns are grouped according to C2S - Client-to-Server - and S2C - Server-to-Client - traffic directions. The detailed description is available at `http://tstat.polito.it/measure.shtml#LOG`. Time series collected in RRD files are exported to the Graphite repository automatically, while for log files we are planning to use the metadata exporter developed in MONROE and integrate them in the MONROE repository directly. Some modification to the mPlane original code for `Tstat` and proxies were needed to support the specificity of the MONROE deployment.

4.3.4 Deployment of mPlane repository

The mPlane repository for RRD logs and GUI is deployed in POLITO. It runs on a dedicated server in POLITO premises with public IP address which possibly can support up to 250 nodes. It is currently dedicated for demo and for the final deployment. The Graphite GUI provides fine grained presentation of imported data. Each MONROE node will automatically export RRD locally generated to the centralized repository, thus offering an easy mean to browse through collected data.

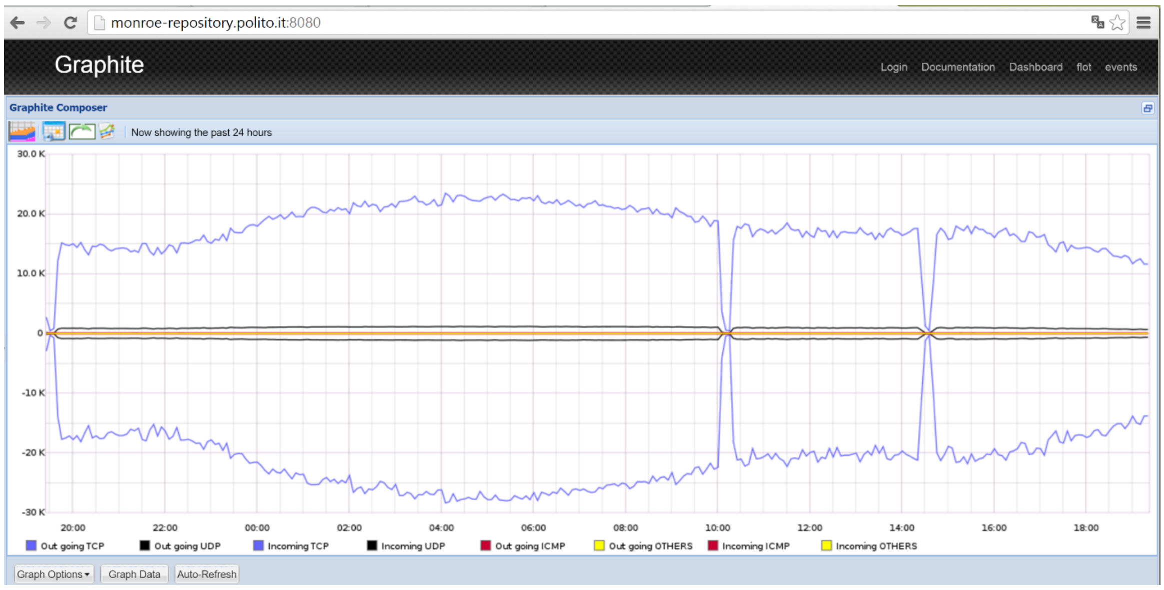


Figure 11: Graphite GUi for RRD logs.

Via a simple graphical interface, the authorized user can thus browse through the data, and easily produce time plots. Some pre-defined graphs are also made available. For instance, Figure 11 shows the plot for the number of different connections in both incoming and outgoing direction in a MONROE node where some periodic tests have been running. The GUI allows to check the correlation of RRD data for different node and provides dashboards for use internal and external users. The repository is already available at `http://monroe-repository.polito.it:8080`.

4.3.5 Deployment of mPlane supervisor and client

The mPlane supervisor is the main component of the mPlane architecture. All components have to register their capabilities to the supervisor. The supervisor is installed in one of the server of POLITO and has a public IP address (`Monroe-supervisor.polito.it`). Each MONROE node is configured to automatically advertise capabilities to this supervisor, which in turn exposes them to the reasoner. The latter orchestrates all measurements, by configuring and starting the `Tstat` passive probes at each MONROE node, and setting them up to export RRD data to the Graphite repository. The reasoner is also installed in the same machine where the supervisor is running.

5 Remote Repositories and Data/Metadata Importer

5.1 Remote Repositories

The Remote Repository is composed of 3 (logical) repositories as already anticipated at the beginning of this document in the Figure 1: (*i*) MONROE repository, (*ii*) mPlane repository and (*iii*) external user repository.

- The MONROE repository collects all the continuous/periodic measurements including the metadata from Containers 1-3. Some of this data will be visualised in near real-time, therefore, we use periodic synchronization between the nodes and the MONROE repository. `rsync` has been used in Nornet so far without much problems, so it is used also in MONROE.
- The mPlane repository collects all the `Tstat` output. This data will further be fed into the mPlane project repository and be visualised through the mPlane GUI.
- The external repository collects all the other measurement and experiment results (ContainerX). MONROE containers further provide experimenters with an OML client for Fed4FIRE testbeds, so that OML can be freely used by experimenters owning an OML server. In general, experimenters can use the tools they prefer to collect the results of their measurements.

5.2 Data/Metadata Importer

The data and metadata are collected in the node and transferred to the repositories, to be stored (imported) in a central database. Data and metadata are pre-processed on the node into files with JSON formatting. The files are periodically copied by `rsync` to a specific incoming directory in the repository. At periodic intervals (which may be much shorter than the transfer interval time) the incoming directory is scanned for new JSON files. These are parsed and inserted into the MONROE database according to database schema. The periodicity of the copy and import operations are configurable and are typically less than a minute.

Tables 1, 2 and 3 report some examples of device metadata and system events and sensors that cause the generation of metadata in the MONROE node. Table 1 reports two examples of metadata sets generated by the modem and by the GPS devices embedded in the MONROE node, respectively. Such metadata sets include basic connectivity status information and basic location info that can be used to correlate the results of the experiments with factors such as the position and the quality of signal. Table 2 shows a list of system events that need to be monitored (reboots, failures, etc.) and so they generate metadata messages to be exported as a consequence of each observed event. Finally, Table 3 reports a list of sensors present in the MONROE node that are used to monitor the mother board utilization and its working status. The events revealed by such sensors are key to monitor the health of the node and the proper behavior of the processes running on the node. Thus, each variation of the quantities read by the sensors is exported as metadata and stored in the database.

Table 1: Device metadata

Device info	**Values**
modem	"InterfaceName": local name of interface, "InterfaceId": ICCID, "Mode": number indicating connection type (LTE, GMS etc), "Operator": number indicating operator, "SequenceNumber": sequence number from core component, "SignalStrength": RSSI, "Timestamp": timestamp when values where logged by core component
GPS	"Longitude": longitude position of the node, "Latitude": latitude position of the node, "Altitude": Altitude of the node, "NumberOfSatellites": Number of Satellites used for reading the position, "SequenceNumber": sequence number from core component, "NMEA": raw NMEA string as received form the GPS unit, "Timestamp": timestamp when values where logged by core component

Table 2: Events causing metadata generation

EventType	**Meaning**	**Extra keys**
Watchdog.Failed	The system watchdog detected an error symptom	message
Watchdog.Repaired	The system watchdog resolved the issue	message
Watchdog.Status	Periodic status messages from the watchdog	message
System.Boot	All core services are running after boot	
System.Halt	System halt is requested	
Scheduling.Started	The node starts to query the scheduling server	
Scheduling.Stopped	The node stops to query the scheduling server, no new experiments are scheduled	
Scheduling.Aborted	All running experiments are about to be stopped, the node is in maintenance mode	
Maintenance.Start	A manual login on the node is registered	
Maintenance.Stop	The manual login is closed	

Table 3: Sensors on the MONROE node causing metadata generation

Sensor	**Field**	**Description**
temp	cpu	Temperature (^{o}C)
rest-services	modems	Modem info service (HTTP Status code)
	dlb	Load balancer info service (HTTP status code)
	usb-monitor	USB hub info service (HTTP status code)
session	id	Session number (Boot counter)
	start	Start time (Unix timestamp)
	current	Uptime (seconds since start of the session)
	total	Uptime (Cumulative uptime of the node over all sessions)
	percent	Uptime (Percent of uptime vs total lifetime of the node)
cpu	system	CPU time spent by the kernel in system activities
	steal	The time that a virtual CPU had runnable tasks, but the virtual CPU itself was not running
	guest	The time spent running a virtual CPU for guest operating systems under the control of the Linux kernel
	iowait	CPU time spent waiting for I/O operations to finish when there is nothing else to do
	irq	CPU time spent handling interrupts
	nice	CPU time spent by nice(1)d programs
	idle	Idle CPU time
	user	CPU time spent by normal programs and daemons
	softirq	CPU time spent handling “batched” interrupts
memory	apps	Memory used by user-space applications
	free	Unused memory
	swap	Swap space used

6 Database

MONROE database (DB) is based on a NoSQL database. Alternatives considered were Apache Cassandra and Scylla, and the prototype includes an implementation using Cassandra. There are some important differences between relational and non-relational databases. The most relevant ones to the MONROE DB are:

- The design is guided by queries, since databases are designed to answer queries quickly. There is no schema and there is no need to model data and relations. In fact, there is not a way to define a relation in Cassandra.
- Redundancy is very good. Cassandra encourages the redundancy of data, exactly the opposite that relational databases do.
- Cassandra has been designed for environments where requirements change often, so changes to database are easy to implement.
- It is easy to add more computer power to the database, and performance grows linearly (as long as database design comes after queries).

The database follows an experiment-oriented design, where the queries performed by users and applications play a central role. There will be two collections of tables, one for experiment measurements, and another for metadata. Each experiment will have its own experiment measurements table (or set of tables). The code snippet reported in Table 4 shows how to create a sample ping experiment table.

Table 4: Cassandre code used to create an experiment table

```
CREATE TABLE monroe.monroe_exp_ping (
    dataversion int,
    nodeid text,
    interfacename text,
    timestamp decimal,
    bytes int,
    dataid text,
    host text,
    rtt double,
    sequencenumber bigint,
    PRIMARY KEY ((dataversion, nodeid, interfacename), timestamp)
) WITH CLUSTERING ORDER BY (timestamp ASC);
```

Figure 12 shows how the Cassandra database is populated and updated by the MONROE system. Experiment data is generated in the nodes, and collected to a central repository by a Data Importer. The Data Importer clean-ups experimental data and then inserts proper rows in the database. MONROE nodes export data and metadata in compressed JSON files,[5] and the Data Importer uses a python script to parse JSON files and to produce *write* requests to the DB.

Writing on the DB is performed very frequently (every 20 seconds in the initial implementation of the system) to allow for near-real-time availability of MONROE data/metadata at the DB. As a consequence, the visualization tool described in Section 7 can show MONROE data/metadata in near-real-time as well. Note that the information of the database can be extracted either through the Visualization tool or an external

[5]We decided to use JSON for it makes it easier to debug the operation of the data/metadata export process. However, we consider more compact representation of data and metadata for future releases of the platform.

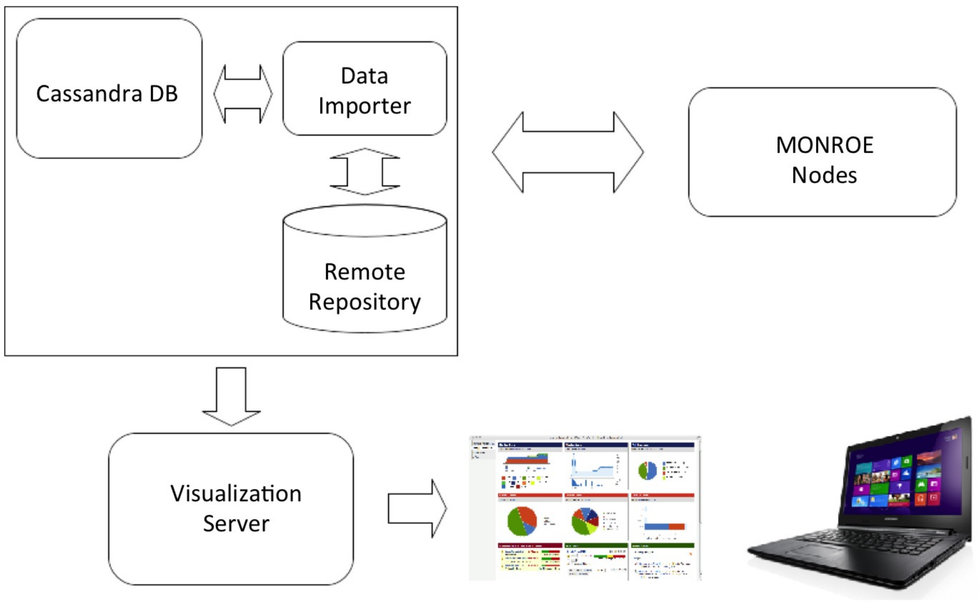


Figure 12: Cassandra architecture.

facility, like the ones offered by Pentaho data warehouse solutions,[6] already integrated with non-relational databases like Cassandra.

More specifically, the JSON files produced by a node include the following input:

- About one entry every 5 seconds, per interface, generated by the modems to comunicate their state and the state of connectivity (therefore, with 5 wireless interfaces, we have about 1 line per second written by each node in a JSON file); after extracting the information contained in the JSON files, this corresponds to about 400 bytes/s to be written in the DB for each node.
- 1 line per second is written to report GPS data (about 250 bytes/s per node at the DB);
- Container 1 produces 1 report per second per each interface used for measurements, therefore we have 3 lines per second per node (about 160 bytes/s per node at the DB).
- One report on connectivity is added about every 5 seconds per each interface, so 1 line per second per node in total (about 50 bytes/s per node at the DB).
- Sensors produce about 4 different messages every 5 seconds, therefore less than 1 line per second per node (less than 1200 bytes/s per node at the DB).

With the above, each node produces less than 7 lines per second, each of which requires a write operation to the DB. In addition, Container 2 will generate a few lines per hour, so we can neglect the load generated by periodic experiments implemented by means of Container 2. The resulting load is about 140 write operations per node every 20 seconds (i.e., each time the python script parses JSON files). With less than 500 nodes, this results in less than 70,000 writing operations every 20 seconds. As we can see from Figure 13, such load is handled quite easily by a Cassandra DB even when using a low-end server.

Indeed, in order to test the performance of the Cassandra database on a variety of hardware configurations, a series of stress tests have been performed using machines on Amazon E2C and a machine property

[6] http://www.pentaho.com/optimize-the-data-warehouse

Table 5: Cost for Amazon E2C instances is calculated for a 3-year period paid on advance (additional disk and data transfer costs are not included).

Instance	CPUs	Memory (GB)	Disk	Cost (USD)
t2.small	1	2	EBS Only	332
t2.medium	2	4	EBS Only	664
t2.large	2	8	EBS Only	1328
m4.large	2	8	EBS Only	1581
m4.xlarge	4	16	EBS Only	3162
m4.2xlarge	8	32	EBS Only	6324
m4.4xlarge	16	64	EBS Only	12 648
m4.10xlarge	40	160	EBS Only	31 620
m3.medium	1	3	1 x 4 SSD	884
m3.large	2	6.5	1 x 32 SSD	1768
m3.xlarge	4	13	2 x 40 SSD	3512
m3.2xlarge	8	30	2 x 80 SSD	7025
Dell PowerEdge C6220 (at IMDEA)	16	96	4096 HDD	10 000

of IMDEA Networks. Table 5 shows the different configurations and costs of the test machines. The test was performed using Cassandra version 2.1.9 and using the built-in stress tool with the following command lines:

```
cassandra-stress write n=3500000 -mode native cql3
cassandra-stress read n=3500000 -mode native cql3
```

Figure 13 shows the time required to complete the test in each machine configuration (t2, m3, m4 configuration families available on AWS), given the number of write operations. Most configurations are not even able to complete the test for the highest numbers of operations, while the ones that do are significantly slower than the IMDEA-owned machine used in the test. However, the requirements of the MONROE system with less than 500 nodes are well below the values used in the stress test, and indeed, the required number of write operations per second (i.e., about 70,000 in 20 seconds with 500 nodes) is well below what can be handled by any of the server configurations used for the stress test.

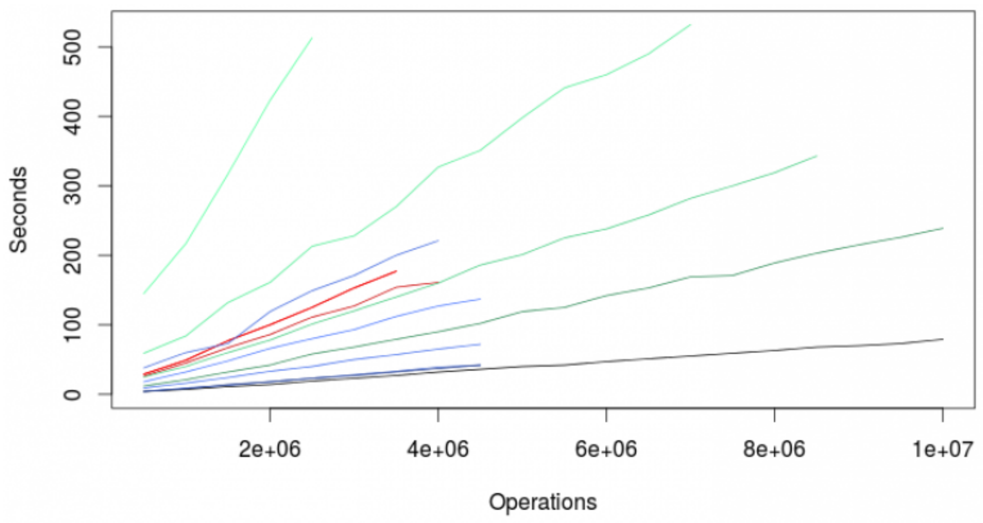


Figure 13: Stress test for Cassandra: Time required to complete a number of write operations. The t2 family is depicted in red, the m4 family in blue, the m3 family in green and the machine at IMDEA ("Archimedes") in black.

The conclusion of these tests is that Amazon E2C prices are too expensive, without reaching the performance levels offered by an owned server. If we can guarantee a stable Internet connection at one of our sites with enough bandwidth, buying a dedicated non-expensive machine (or perhaps two to improve concurrency and serve as a backup) seems the best option.

7 Visualisation

The visualisation tool has been designed to provide a graphical representation of the MONROE platform in terms of deployment of the nodes in the different countries. The scope of the application is also to present the status of each device and the results of the experiments which are running in the containers in order to realize a nearly real-time monitoring of the system.

From an architectural point of view, the visualisation tool is composed of two layers:

- The **back-end** server: This module creates a stable connection with the CASSANDRA database and performs queries to retrieve the node capabilities, the device positions and the monitoring information. It exposes the data to the upper layer through a REST interface.
- The **front-end** client: This component basically manages the web pages and their contents including third-party libraries which are used to create an easy layout for the user (i.e charts, maps, tables, etc.).

The back-end layer is based on the *NodeJS* technology and the *Express* web application framework. In short, NodeJS is an open-source and cross-platform runtime environment which easily allows developers to build scalable server-side web applications using its event-driven architecture and a non-blocking (asynchronous) API. In this context, Express is the de-facto standard application server framework for NodeJS due its lightness and stability.

The front-end layer is strictly derived from the *AngularJS* javascript framework that realizes the model-view-controller (MVC) design pattern and extends the HTML language with new directives and filters. The key point of the framework is the facility to introduce new controllers and services in the application with few lines of code.

Finally, we illustrate the visualization tool with some figures to show how the application looks like in the prototype implementation.

Figure 14 shows the position of the MONROE nodes on the globe. The open source WebGL Earth 3D library[7] is used to build the object. Figure 15 shows the position of a mobile node that is deployed in Sweden close to the university of Karlstad. The well-known Google map library[8] has been introduced to draw the map with a marker that points to the position of the node. Clicking on the node image, the user is redirected to a web page that includes the tracking of the node (see Figure 16), the round trip time and the signal strength (see Figure 17) for the last 10 minutes. Figure 17 highlights some examples of the multiple charts used to show the time series data. The charts are built using the Highcharts library[9] and its available types, e.g RTT as a line chart, the packet loss as a speedometer gauge, connection type as a pie chart and the signal strength as a column chart.

8 Open code

All the open source SW generated to implement the MONROE prototype described in this demo deliverable is available at `https://github.com/MONROE-PROJECT`. More specifically:

[7] `http://www.webglearth.org`
[8] `https://developers.google.com/maps/`
[9] `http://www.highcharts.com/products/highcharts`

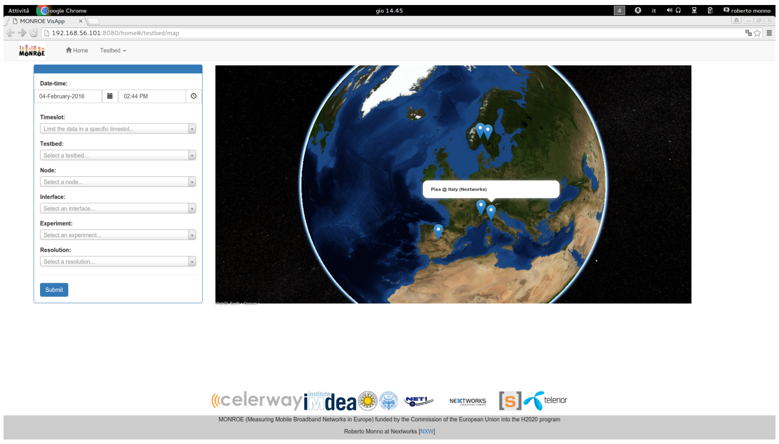
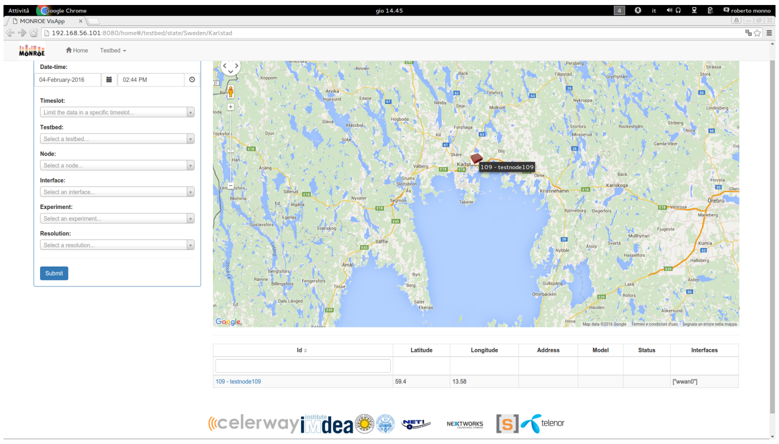


Figure 14: MONROE data-centres on the globe.



Id	Latitude	Longitude	Address	Model	Status	Interfaces
109 - testnode109	59.4	13.58				["wwan0"]

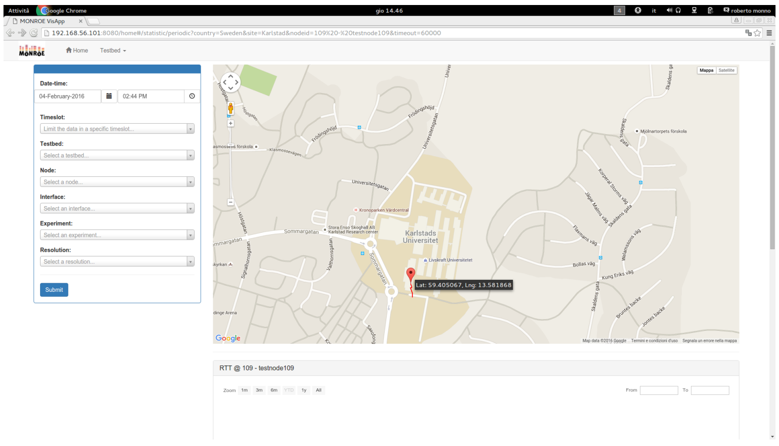
Figure 15: The testnode109 running at the Karlstad university.

- Experimenters portal: `https://github.com/MONROE-PROJECT/UserAccess`;
- Scheduling system: `https://github.com/MONROE-PROJECT/Scheduler`;
- Metadata generator and (meta-)data exporter: `https://github.com/kristrev/data-exporter`;
- `Tstat` container: `https://github.com/MONROE-PROJECT/mPlane`;
- Database: `https://github.com/MONROE-PROJECT/Database`;
- Visualisation: `https://github.com/MONROE-PROJECT/Visualisation`.

Management and maintenance SW as well as the core components will not be made open source since they are background to MONROE project.

9 Conclusions

In this demo deliverable, we have presented the prototype of the MONROE platform, developed in the framework of the Fed4FIRE testbed federation paradigm. In particular, we have presented the software components developed in the project and that includes seven main parts: (*i*) User access, (*ii*) Scheduling system,


Attività
Google Chrome
gio 14.46
roberto monno
MONROE VisApp
192.168.56.101:8080/home#/statistic/periodic?country=Sweden&site=Karlstad&nodeid=109%20-%20testnode109&timeout=60000
Home
Testbed
Mappa
Satellite
Karlstads Universitet
Google

Date-time: 04-February-2016 02:44 PM

Timeslot: Limit the data in a specific timeslot...

Testbed: Select a testbed...

Node: Select a node...

Interface: Select an interface...

Experiment: Select an experiment...

Resolution: Select a resolution...

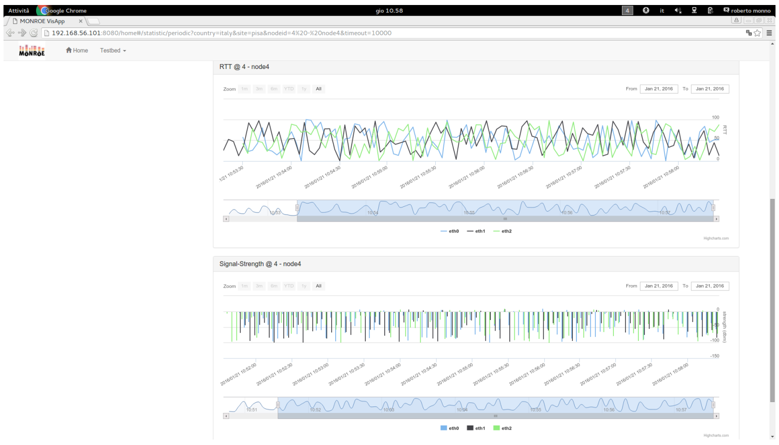
Submit

Lat: 59.405067, Lng: 13.581868

RTT @ 109 - testnode109

Zoom 1m 3m 6m YTD 1y All From To

Figure 16: Tracking of the node.



RTT @ 4 - node4

Zoom 1m 3m 6m YTD 1y All From Jan 21, 2016 To Jan 21, 2016

Signal-Strength @ 4 - node4

Zoom 1m 3m 6m YTD 1y All From Jan 21, 2016 To Jan 21, 2016

Figure 17: RTT and signal strength charts.

(*iii*) Management and maintenance system with Inventory, (*iv*) MONROE node SW, (*v*) Repositories and Data importer, (*vi*) Database, and (*vii*) and Visualization. The majority of the code produced so far in the projects has already been released for open access. All the software components will be improved and enhanced based on the use-cases and the comments or requests of the experimenters during the second year of the project.