



mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Integrated Prototype - Final Release

Author(s):	POLITO	A.S. Khatouni, S. Traverso, M. Mellia, M.M. Munafò
	FUB	E. Tego, F. Matera
	SSB	G. De Rosa, S. Pentassuglia
	TI	F. Invernizzi
	ALBLF	Z. Ben Houdi
	EURECOM	M. Milanesio
	ENST	P. Casoria, D. Cicalese, D. Rossi
	NEC	M. Ahmed, K. Kutzkov
	TID	I. Leontiadis, M. Varvello, L. Baltrunas
	NETVISOR	B. Szabó, L. Németh, G. Molnár, J. Bartók-Nagy (ed), Á. Bakay
	FTW	P. Casas, A. D'Alconzo
	FHA	M. Faath, R. Winter
	ULG	B. Donnet, K. Edeline, Y. Liao
	ETH	B. Trammell
	FW	E. Kowallik

Document Number: D5.4
Revision: 1.0
Revision Date: 15 September 2015
Deliverable Type: RTD
Due Date of Delivery: 31 August 2015
Actual Date of Delivery: 15 September 2015
Nature of the Deliverable: (S)oftware
Dissemination Level: Public

Abstract:

This deliverable assesses the completion of the integrated prototypes of the mPlane platform (including the results of integration tests). In addition to this documentation, the final release of the mPlane library and software are also part of this deliverable.

The details of the system architecture, element functions and interactions, use cases descriptions and integration plans can be found in previous mPlane deliverables available at the official mPlane website <http://www.ict-mplane.eu/>.

As a prelude to this deliverable, D5.1 contained a report about the project's data collection track record, D5.2 reported on the mPlane components to be included in the integration test plants, including a description of the mPlane SDK and the use cases to be realized in an integrated manner, and D5.3 described the requirements of the integrated mPlane prototype and test beds.

Keywords: mPlane, integration, unified, probe, supervisor, repository, reasoner, use case

Disclaimer

The information, documentation and figures available in this deliverable are written by the mPlane Consortium partners under EC co-financing (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission.

The information in this document is provided ``as is'', and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.

Contents

Disclaimer.....	3
Document change record.....	5
Executive Summary.....	6
1 Platforms used for integration.....	7
1.1 Telecom Italia Lab	7
1.2 Telecom Italia Virtual Appliance	9
1.2.1 Virtual image	9
1.2.2 Installed software	9
1.2.3 Configuration	10
1.3 NETVisor mPlaneProbe - an embedded mPlane platform.....	13
1.3.1 Description	13
1.3.2 Architecture	13
1.3.3 Features.....	15
1.3.4 Application notes and software availability	15
1.4 ISCOM-FUB test bed	17
1.5 Polito integration platform.....	19
2 Components used in integrations.....	20
2.1 mPlane SDK.....	20
2.2 Supervisor	22
2.2.1 Client Shell	22
2.2.2 Supervisor GUI.....	22
2.3 Probes	27
2.4 Repository tools	28
2.5 Reasoners and Analysis modules	29
3 Integration tests.....	31
3.1 Tests performed by TI	31
3.2 Tests performed by FUB	37
3.3 Tests performed by Polito.....	39
4 Current state of Use Case prototypes.....	41
4.1 Estimating Content and Service Popularity for Network Optimization	41
4.2 Active Measurements for Multimedia Content Delivery Use Case	43
4.3 Quality of Experience for Web Browsing Use Case	49
4.4 Mobile Network Performance Issue Cause Analysis Use Case	50
4.5 Anomaly Detection and Root Cause Analysis in Large-scale Networks Use Case	51
4.6 Verification and Certification of Service Level Agreements Use Case.....	54
4.7 Passive Content Curation Use Case.....	56
5 Software availability.....	58

Document change record

Version	Date	Author(s)	Description
0.0	28 August 2015	J. Bartók-Nagy (NETvisor)	initial draft
0.3	31 August 2015	J. Bartók-Nagy (NETvisor)	finalized draft
0.9	04 September 2015	J. Bartók-Nagy (NETvisor)	pre-release version
1.0	15 September 2015	J. Bartók-Nagy (NETvisor)	final version

Executive Summary

This deliverable assesses the completion of the integrated prototypes of the mPlane platform (including the results of integration tests). It briefly describes the different platforms used by partners to integrate and test the software solutions developed within the project. In addition to this documentation, the final release of the mPlane library and software are also part of this deliverable and made available through the mPlane website.

As a prelude to this deliverable, D5.1 contained a report about the project's data collection track record, D5.2 reported on the mPlane components to be included in the integration test plants, including a description of the mPlane SDK and the use cases to be realized in an integrated manner, and D5.3 described the requirements of the integrated mPlane prototype and test beds, respectively.

Being part of the previous deliverables, the details of the system architecture, element functions and interactions, use cases descriptions and integration plans can be found in the corresponding mPlane documents and website, available at the official mPlane website (<http://www.ict-mplane.eu/>) and will not be discussed here in details.

In Chapter 1 we introduce the integration platforms developed/deployed for the mPlane project. Being as broad as possible, we define "integration platform" as an environment which is capable and used to run more than one mPlane component, so this term covers both the complete test environment set up for mPlane project at one of our partners and a single virtual device or appliance which hosts only some of the mPlane components and connect to other virtual or physical mPlane elements. We will not describe here the final testbed the consortium is preparing in the Fastweb testplan. It will be used for demonstration, where all use cases will be demonstrated, and described in Deliverables of WP6.

In Chapter 2 we shortly enumerate the mPlane components which will take part in the integration and demonstration phases. Most sections in this chapter will be only a short overview of the components, since they have been discussed in detail in the previous deliverables. The only exceptions are those components which has not been shown in other deliverables like the Supervisor.

In Chapter 3 we describe those tests done by Consortium Partners which should be considered as "shared" or "multi-use" integrations, because they will be used in several use cases.

In Chapter 4 we report the (integration) status of the Use Cases, which will be part of the demonstration phase of the project.

In Chapter 5 we give the availability of the official access details of the mPlane software libraries and documentation.

1 Platforms used for integration

In this chapter we introduce the integration platforms developed/deployed for the mPlane project. Being as broad as possible, we define "integration platform" as an environment which is capable to and used to run more than one mPlane component, so this term covers both the complete test environment set up for mPlane project at one of our partners and a single virtual device or appliance which hosts only some of the mPlane components and connect to other virtual or physical mPlane elements.

We will cover the following mPlane test platforms:

- Telecom Italia Lab
- Telecom Italia Virtual Appliance
- NETVisor mPlaneProbe - an embedded mPlane platform
- ISCOM-FUB test bed
- Polito integration platform

The consortium is also working to setup a completely integrated testbed deployed at FastWeb premises that will be mostly used for demonstration purposes and not for integration testing. It has been described in D6.1, "The mPlane demo infrastructure section, and it has been left out from this deliverable (also to avoid repetitions).

In the following we describe the different platforms, deliberately providing a brief description to avoid entering into very narrow details.

1.1 Telecom Italia Lab

For integration tests, a specific lab architecture has been deployed, as described in previous documents (see Deliverable 5.2). For sake of completeness here a brief recap is included. The platform includes all servers and nodes connected to the Telecom Italia Testplant. The whole network is directly interconnected to the Internet in order to expose public services. A direct connection has been setup by means of a dedicated fiber link to the Fastweb Lab premises. Static routing between the two networks grants that only mPlane traffic is transported over this link. In the context of this lab premises, the public Supervisor managed by Telecom Italia has been configured and is currently exposed to Internet. It can be reached at `supervisor.ict-mplane.eu` and it is currently open and used by the virtual appliances (see next section). Any client provided by the correct certificate can then access it and use it.

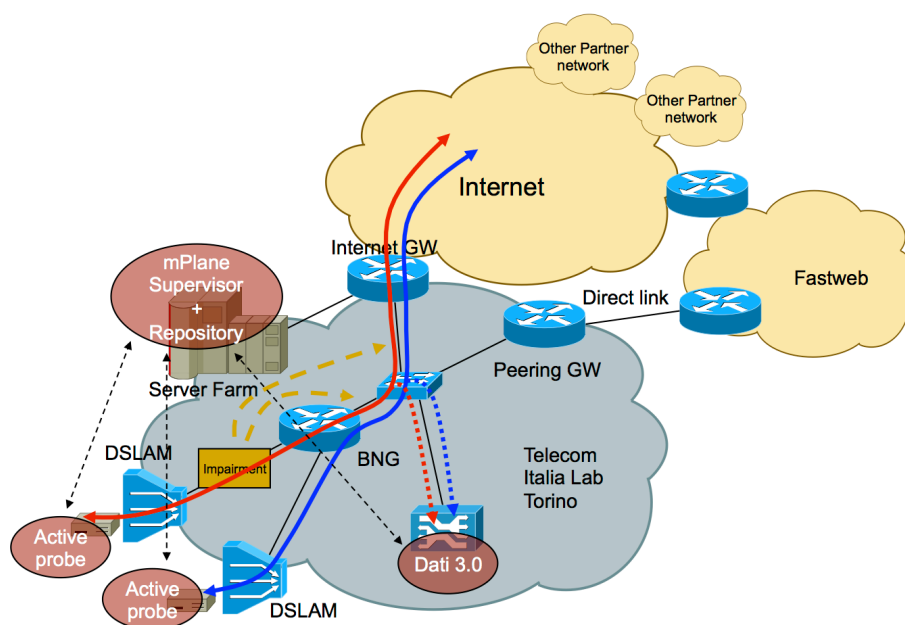


Figure 1: Telecom Italia LAB network

1.2 Telecom Italia Virtual Appliance

The goal of the mPlane virtual appliance is to provide users with a pre-packaged, ready to use system that can be easily and quickly configured with basic parameters (network configuration, identity, ...) and make use of in order to do experiments with an mPlane infrastructure or simply taste the potential of the protocol. For these reasons the system comes pre-configured pointing to the public supervisor activated in Telecom Italia premises (`supervisor.itc-mplane.eu`) and with a number of client certificates recognized by the supervisor as valid so that a user can choose between them for identify itself.

Since the probe is one of the output of integration activities, the system contains active and passive probes ready to interoperate with mPlane.

1.2.1 Virtual image

The image is formatted as an open virtualization file in order to grant a portable, efficient and simple way to distribute and deploy on different virtualisation platforms. The OVF file contains a single virtual system deployed and tested in a cloud environment with this characteristics:

- Hypervisor: VMware ESXi, 5.5.0, 1623387
- VIM: VMware Vsphere 5.5

The system is an Ubuntu 14.01(64-bit) configured with:

- 1 CPU
- GB Memory
- 1 Hard disk (30GB thin provisioned)
- NIC 1 (vlan 102) that corresponds to eth0 (management interface)
- NIC 1 (vlan 601) that corresponds to eth1 (tstat capture interface)

1.2.2 Installed software

In order to have a full view of mPlane's potential, several different systems has been installed, being capable of to do a number of measurements. In particular

- **nodejs probe**: This is the general purpose latency and hop count active probe developed by TI
- **tstat**: The passive probe developed by POLITO
- **mplane SDK**: The publicly available mPlane SDK with all the available proxies at the time of publishing the document.

Some system and life cycle management tools has also been installed:

- **pm2**: This is a process manager that offers a simple way to manage the installed mPlane components (e.g. start, stop, show log).
- **pm2-GUI**: Web interface of pm2
- **mplane.sh**: A simple shell script that is a workaround to pm2.

All software dependencies and libraries have been installed with supported releases in order to have all needed software working.

1.2.3 Configuration

The system has standard SSH configured for remote management. You can log into the system using the following credentials: user:mpplane, password:Mplan6. The mpplane user has sudoers privileges to act as root.

The system is configured with two network interface:

- `eth0`: this is the management interface, configured for auto configuration with dhcpd. The image has this interface configured on vlan 102: set the correct vlan on the cloud configuration system, if required, to connect this interface to your management network.
- `eth1`: this is the interface tstat will collect statistics from. It is configured to be on vlan 601: change this accordingly to your network configuration.

1.2.3.1 Certificate selection

A valid certificate is needed in order to have valid communication with an mPlane element. The virtual image comes with a number of certificates signed by a self-signed root certification authority recognized by public mPlane Supervisors. Certificates can be found in `/usr/mpplane/certs`: select one of your choice and make a symlink in `/usr/mpplane` that looks like following ones:

- `my_cert.crt -> certs/probes/Probe_TI.crt`
- `my_cert-plaintext.key -> certs/probes/Probe_TI-plaintext.key`


1.2.3.2 Components configuration

Configurations of mPlane official SDK based probes are contained in the file `/usr/mpplane/protocol-ri/mpplane_components.conf`. A nodejs based probe is included in `/usr/mpplane/TI_generic_probe`: in order to change the configuration refer to `/usr/mpplane/TI_generic_probe/probe.json`. Please refer to official mPlane documentation for configuration details.

1.2.3.3 Managing the mpplane probe

In order to simplify lifecycle management, a process manager (PM2) has been installed and configured with a shell wrapper, named `mpplane`, configured in the system `PATH`. In order to start needed precesses for the probe, exec command `mpplane start`. This command will start both the components (nodejs and python), the tstat passive monitor and a WEB based GUI of the process manager (see below). With the `mpplane` command you can monitor, stop, start and see logs from all the processes (see following figures).

```
root@mplane:/usr/mplane# mplane start
```



App name	id	mode	pid	status	restart	uptime	memory	watching
TI_mPlane_node_probe	4	fork	5752	online	0	2s	58.906 MB	disabled
Tstat	5	fork	5756	online	0	2s	1.355 MB	disabled
Mplane_RI	6	fork	5763	online	0	2s	1.371 MB	disabled
PM2_WEB	7	fork	5771	online	0	2s	40.559 MB	disabled

Use 'pm2 show <idname>' to get more details about an app


Figure 2: mPlane start

```
* PM2 monitoring :
```

* TI_mPlane_node_probe	[]	0 %
[0] [fork_mode]	[]		58.906 MB
* Tstat	[]	0 %
[1] [fork_mode]	[]		1.352 MB
* Mplane_RI	[]	0 %
[2] [fork_mode]	[]		1.371 MB
* PM2_WEB	[]	0 %
[3] [fork_mode]	[]		41.180 MB

Figure 3: mPlane monitor

```
root@mplane:/usr/mplane# mplane status
```



App name	id	mode	pid	status	restart	uptime	memory	watching
TI_mPlane_node_probe	0	fork	1055	online	0	8s	58.906 MB	disabled
Tstat	1	fork	1058	online	0	8s	1.352 MB	disabled
Mplane_RI	2	fork	1061	online	0	8s	1.371 MB	disabled
PM2_WEB	3	fork	1068	online	0	8s	40.418 MB	disabled

Use 'pm2 show <idname>' to get more details about an app

```
root@mplane:/usr/mplane#
```

Figure 4: mPlane status

1.2.3.4 WEB process manager

As described in previous paragraph, the mplane shell command starts a web GUI that is accessible from https://<your_IP>:9001 (user mplane , password Mplan6), from which you can monitor, start, stop all the processes as with the CLI version.

WEB GUI is useful for inspecting logs and system status as shown in following figures.

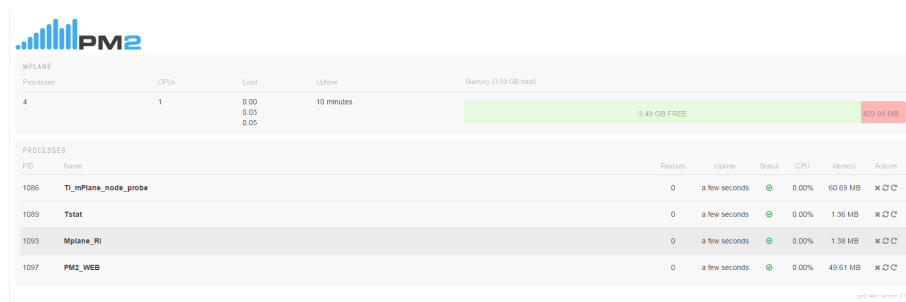


Figure 5: PM2 WEB GUI

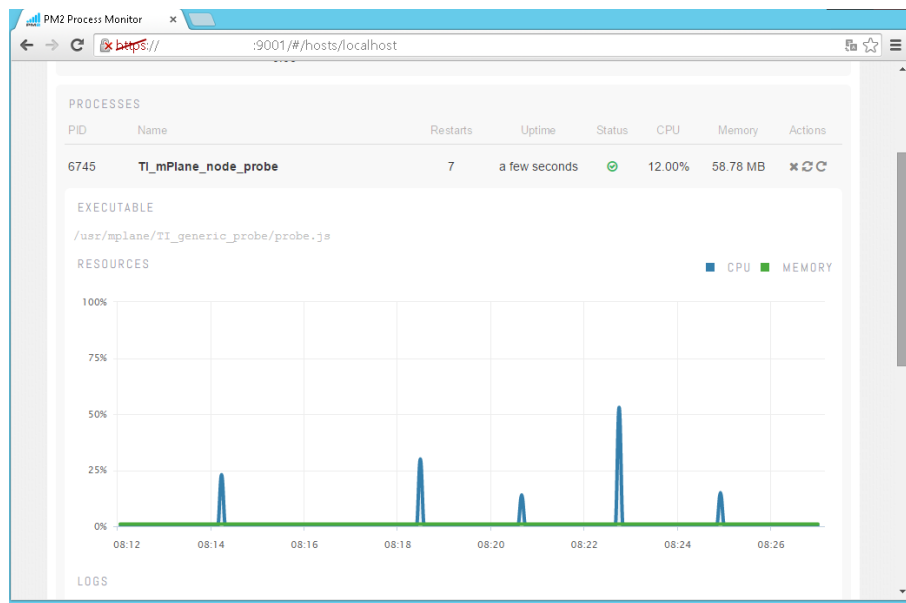


Figure 6: PM2 WEB GUI - system load

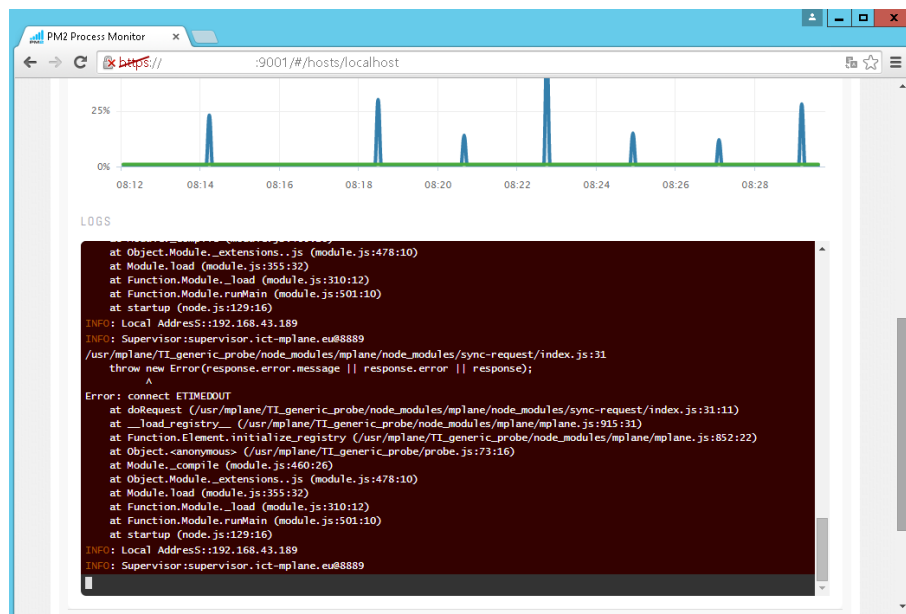


Figure 7: PM2 WEB GUI - process logs

1.3 NETvisor mPlaneProbe - an embedded mPlane platform

1.3.1 Description

mPlaneProbe is an embedded mPlane implementation running on NETvisor's Miniprobe appliance.

The Miniprobe itself is an IPTV and OTT monitoring appliance, designed for use at subscriber premises and similar locations: it is an easy to setup and use box, with small resource footprint and low power consumption.



Figure 8: NETvisor's Miniprobe appliance

The mPlaneProbe is an enhancement to Miniprobe, extending its capabilities with mPlane functionalities. It features

- OpenWrt-based firmware with custom-developed monitoring extensions and a port of the mPlane SDK/RI
- Embedded Repository (EZ-Repo) for storing data from internal and external mPlane Probes
- Local WEB GUI for configuration and for inspecting measurement results.

1.3.2 Architecture

The mPlaneProbe could be used in two scenarios:

- **"All-in-one" configuration**
This is a "self-contained" configuration, with mPlane probes, a local repository (EZrepo), and an

(optional) supervisor GUI. See Figure 9

- **"Repository" configuration**

In this configuration local probes are not used, just the local repository and the supervisor GUI, so the mPlaneProbe box is more like a management server (Figure 10) and it offers both a supervisor and a repository.

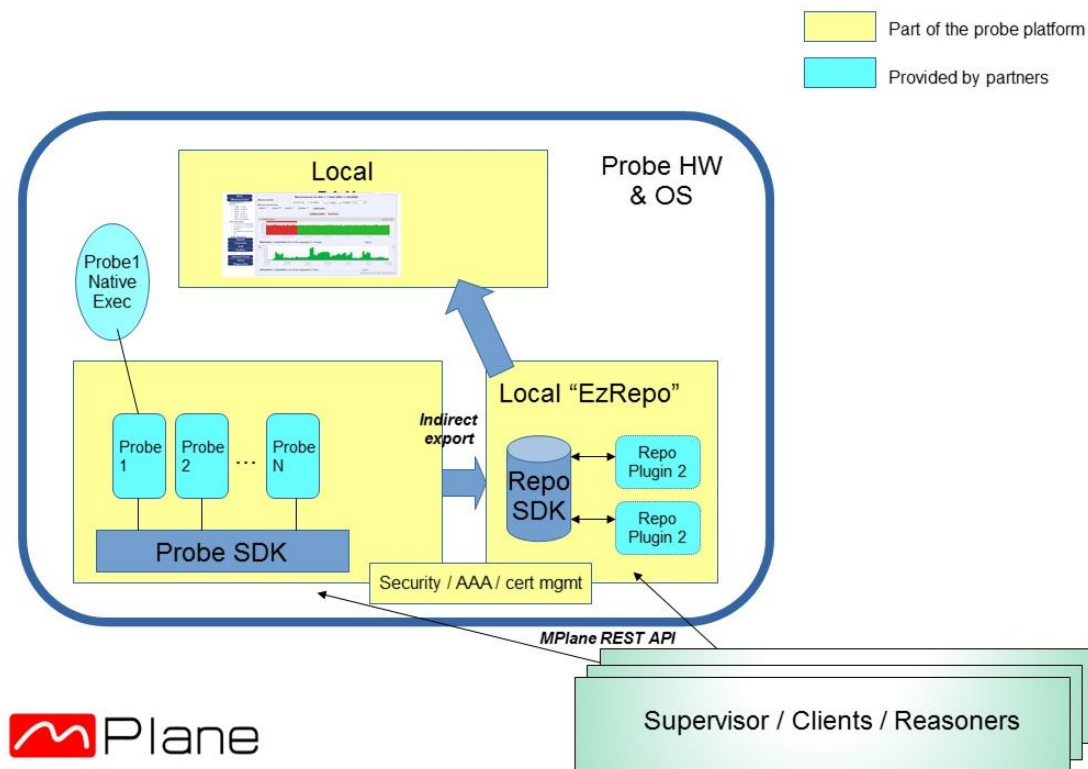


Figure 9: Miniprobe in all-in-one setup

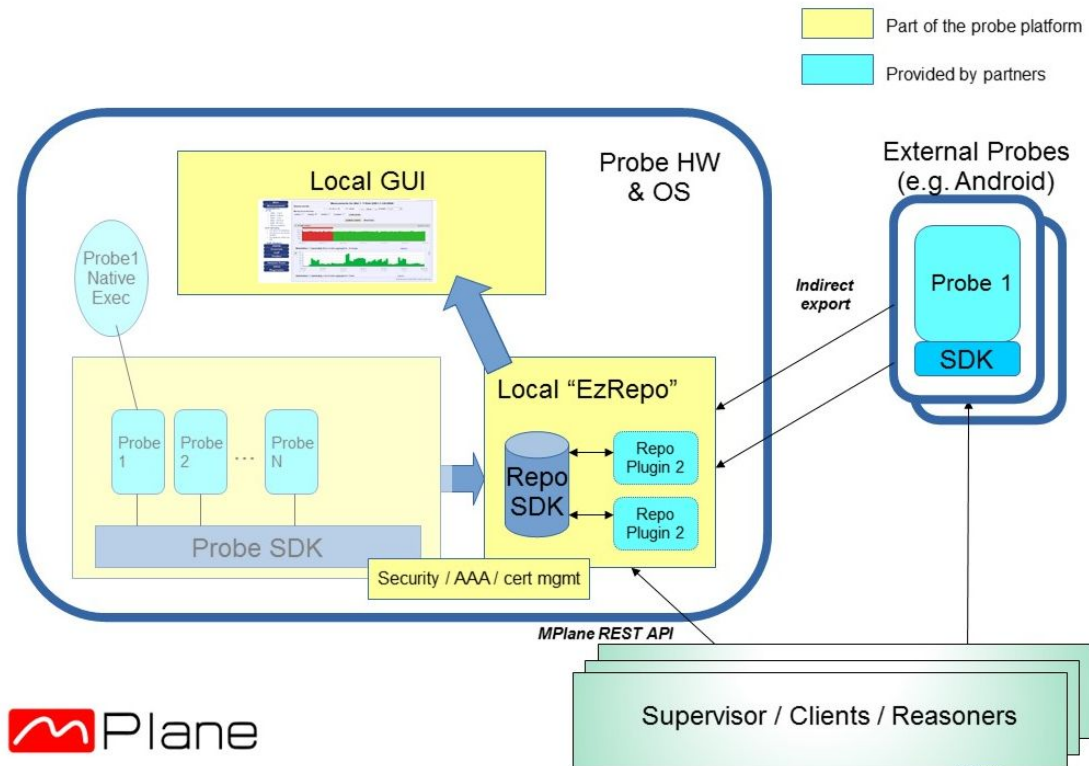


Figure 10: Miniprobe as external probe

1.3.3 Features

mPlaneProbe was designed to be able to host any kind of mPlane compliant component (probe and/or repository, see "Application notes and software availability" below). It can

- launch (activate) any mPlane compliant probe component installed and make it register to a supervisor
- connect the probes to an (optional) repository
- launch a supervisor GUI to make it easy to handle the measurements.

Figure 11 shows the GUI of the mPlaneProbe.

1.3.4 Application notes and software availability

mPlaneProbe uses a modified version of the MiniProbe's OpenWrt based firmware, to obtain it please contact tufa@netvisor.hu.

mPlaneProbe was tested with the following standard mPlane softwares:

- **supervisors:** standard supervisor, svgui
- **repositories:** EZrepo
- **probes:** pinger, ott-probe, GLIMPSE.

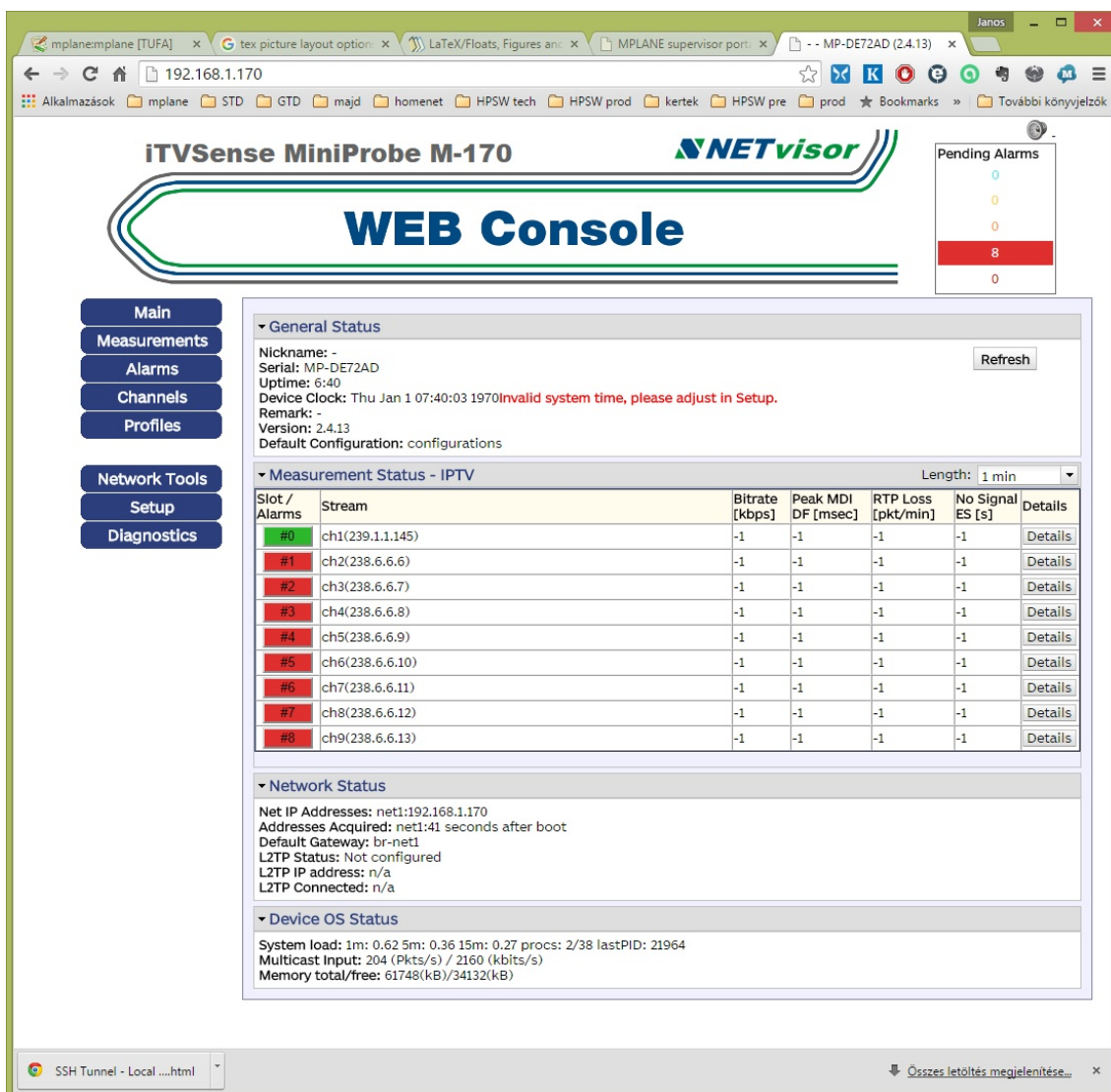


Figure 11: Miniprobe main menu

Porting of probes with native code

It is possible to port any probe binary as long as the source code for the binary is supported by OpenWrt, which applies for C/C++. Any scripting language (amongst them Python and nodeJS) can be used if they are supported by OpenWrt - in this case binary porting is not required.

1.4 ISCOM-FUB test bed

In the framework of Mplane project FUB used the ISCOM test bed to verify its active probe (mSLAcert) and to experimentally investigate several SLA environments concerning wireline access networks, including processes and degradations occurring in the core part. Such a test bed was already described in D5.2 and in Fig. 12 we report its scheme. Here we remember that the test bed represents a regional network with core routers connected with G.652/G.655 optical fibers contained in the Roma-Pomezia cable (50 km), with the access segments based on xDSL and FTTx architecture, mainly based on a Giga-bit Passive Optical Network (GPON). The core consists of four IP routers Juniper M10i ($J_i, i=1,\dots,4$) and three Alcatel SR7750 routers (ALCi, $i=1,\dots,3$) [3]. The group of the Alcatel routers is a Carrier Ethernet cloud based on PBB-TE (Provider Backbone Bridging-Traffic Engineering) technology [3] and they were adopted to define specific class of services.

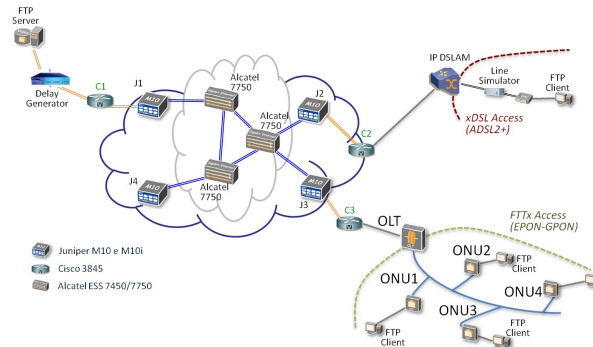


Figure 12: Network test bed.

The mSLAcert active probe, just developed in the framework of Mplane to verify and certify SLA, was deployed in this network, compliant with the mPlane Reference Implementation architecture. The probe works as specified by the mPlane protocol. First the probe tries to connect to a supervisor, then immediately after it is connected to the supervisor it announces its capabilities to the supervisor. After that it waits for the specification from the supervisor. Once the probe receives specification it starts the requested measurement, and at the end of the test it sends the receipt to the supervisor. The supervisor can then check the results of the received receipt. According to mPlane architecture the probe is located at the edge point of the network on which the measurement should be done, the probe is composed of two components, a server and a client, both part of the architecture are registered to the supervisor, the repository can be located at the mPlane server architecture, on which will be stored all the data of the measurements. The reasoner is also another component of the architecture, we treat the reasoner as any other component on the architecture, as the probe, the reasoner also has to register to the supervisor and get the capabilities from it. In Fig. 13 we show a photo of the network test bed including the PCs operating as Mplane probe and server. In the last mPlane configuration, also Tstat tool is included in order to monitor traffic in the core and edge test bed segment, and in particular is under investigation the correlation between active and passive measurements. More details regarding the Mplane probe links to internet and to the supervisor are described in D5.2 par. 3.3.



Figure 13: The Test Bed Setup at “Ministero dello Sviluppo Economico – Dipartimento Comunicazioni” with Mplane probe and server.

1.5 Polito integration platform

All mPlane components which interact with Tstat probe are currently being tested in the POLITO premises, and will be replicated in FW premises for the final demonstration. The current testing platform consists of a set of servers, each running one or multiple components. In particular,

- A supervisor is running on a first linux server, and allows any component to register and clients to interact with them
- A second high end server is running Tstat as a probe, and it is connected to the 10Gb/s links that are interconnecting the Politecnico di Torino Network to the Internet.
- A separate process on the same server offers indirect export capabilities
- A third server is running the content curation analysis modules, and is fed by data thanks to the direct synchronous module that exposes data from the Tstat probe.
- A forth server runs the Graphite proxy, and it receives the temporal data produced by Tstat using the asynchronous export module
- A fifth server is running DBStream repository, and receives the data from the Tstat probe thanks to the MATH export module.

At the moment, the platform is being used for the testing of demonstrations of all use cases involving Tstat, plus the extra case of Graphite for Tstat's data visualization.

As previously described, the platform builds around the Tstat probe which is currently installed at the backbone link of the campus network. Hence, all repositories and analysis modules in Polito premises are fed with the measurement data generated by such probe.

Sec. 3.3 reports a detailed description of the machines building the platform and the tests that have been performed on it.

2 Components used in integrations

In Chapter 2 we shortly enumerate the mPlane components which will take part in the integration and demonstration phases. Most sections in this chapter (probes, repositories, reasoners) will be only a short overview of the components, since they have been discussed in detail in the previous deliverables. The only exceptions are those components which has not been shown in other deliverables like the Supervisor.

2.1 mPlane SDK

The mPlane Software Development Kit (SDK) for Python 3 <https://github.com/fp7mplane/protocol-ri> provides the basis for the development of mPlane components and clients. With the exception of components and clients that use their own protocol implementations (e.g. the node.js implementation of the mPlane protocol), the SDK provides the glue for each probe, repository, supervisor, and reasoner implementation in the integrated prototype.

The SDK implements the protocol as specified in D1.4.

To review: mPlane is built around an architecture in which **components** provide network measurement services and access to stored measurement data which they advertise via **capabilities** completely describing these services and data. A **client** makes use of these capabilities by sending **specifications** that respond to them back to the components. Components may then either return **results** directly to the clients or sent to some third party via **indirect export** using an external protocol. The capabilities, specifications, and results are carried over the mPlane **protocol**, defined in detail in D1.4. Components can be pulled together into an infrastructure by a **supervisor**, which presents a client interface to subordinate components and a component interface to superordinate clients, aggregating capabilities into higher-level measurements and distributing specifications to perform them. A client which provides automation support for measurement iteration in troubleshooting and root cause analysis is called a **reasoner**.

An mPlane measurement infrastructure is built up from these basic blocks.

The mPlane protocol is, in essence, a self-describing, weakly-imperative, error- and delay-tolerant remote procedure call (RPC) protocol: each capability exposes an entry point in the API provided by the component; each specification embodies an API call; and each result returns the results of an API call. The protocol is defined in terms of an abstract information model, a data model binding to a set of representations, and bindings to a set of session protocols for the transport of mPlane messages. Presently, only a JSON representation and an HTTPS session protocol are defined and implemented by the SDK.

Using HTTPS as a session protocol, TLS is always used for confidentiality, integrity and authentication. Both mPlane components and clients are identified by TLS certificates; these certificates are verified by the corresponding peer, and the client identities upon which access control decision are made are derived from the client's subject DN. Though complex access control decisions are generally only taken by supervisors, the authorization framework is part of every component built using the SDK.

The SDK is composed of several modules:

- `mplane.model`: Information model and JSON representation of mPlane messages. This handles marshaling and unmarshaling of mPlane protocol messages.
- `mplane.scheduler`: Component runtime scheduler. Maps capabilities to Python code that implements them (in *Service*) and keeps track of running specifications and associated results (Job

and `MultiJob`). Used by the component framework to run component-specific measurement code based on a specification's temporal scope. To implement a component, place the code to run the measurement or query associated with a given capability in a subclass of `mplane.scheduler.Service`.

- `mplane.client`: mPlane client framework. Handles client-initiated (`HttpClient`) and component-initiated (`ListenerHttpClient`) workflows.
- `mplane.component`: mPlane component framework. Handles client-initiated (`ListenerHttpComponent`) and component-initiated (`InitiatorHttpComponent`) workflows.
- `mplane.tls`: Handles transport layer security for HTTPS, and maps local and peer certificates to identities for access control.
- `mplane.azn`: Handles access control, mapping identities to roles and authorizing roles to use specific Services.
- `mpcom`: Command-line component runtime. Wrapper around `mplane.component` to run components built as Python modules.
- `mpcli`: Simple command-line shell for debugging of components and supervisors.
- `mpsup`: Simple proxy supervisor for debugging of components and clients.

2.2 Supervisor

The Supervisor plays a critical role in orchestrating the different mPlane elements. To enable human interaction and viewing, we offer two different user interfaces with Supervisor:

- **Client Shell:** a command line interface to manage the basic interactions
- **Supervisor GUI:** a web based graphical interface to give an easy-to use alternative, with graphic viewing capabilities.

2.2.1 Client Shell

The mPlane Client Shell is a simple client intended for debugging of mPlane infrastructures. To start it, simply run `mpcli`. It supports the following commands:

- **seturl** Set the default URL for sending specifications and redemptions (when not given in a Capability's or Receipt's link section)
- **getcap** Retrieve capabilities and withdrawals from a given URL, and process them.
- **listcap** List available capabilities
- **showcap** Show the details of a capability given its label or token
- **when** Set the temporal scope for a subsequent `run` command
- **set** Set a default parameter value for a subsequent `run` command
- **unset** Unset a previously set default parameter value
- **show** Show a previously set default parameter value
- **run** Run a capability given its label or token
- **listmeas** List known measurements (receipts and results)
- **showmeas** Show the details of a measurement given its label or token.
- **tbenable** Enable tracebacks for subsequent exceptions. Used for client debugging.
- **help** Displays this summary. Shut down the shell by typing EOF (control-D).

2.2.2 Supervisor GUI

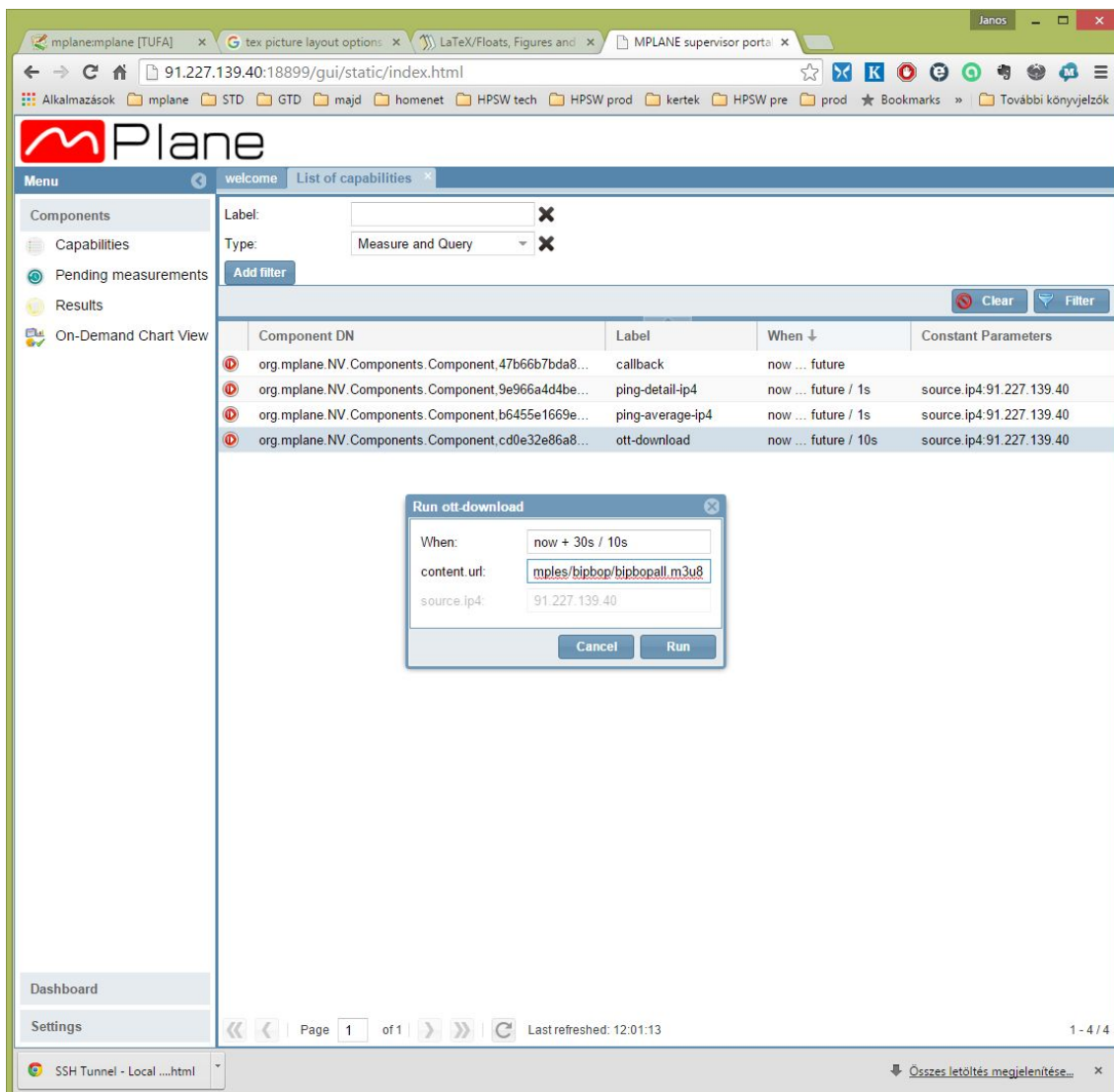


Figure 14: Supervisor GUI - Run capabilities menu

The Web GUI is an ongoing development with extensions implemented as needed by the Use Cases and other test scenarios. As of now, it offers "basic level" functionality like

- listing and filtering of capabilities offered by Components registered to a supervisor
- launching selected measurement tasks (specifications) based on the registered capabilities (See Figure 14)
- show and terminate running (pending) measurements
- display the results of finished measurements (See Figure 15)
- display metric results in graphs (See Figure 16)
- Ability to display various results on a screen area with multiple charts. Each chart can be individually configured to display one or several data series.

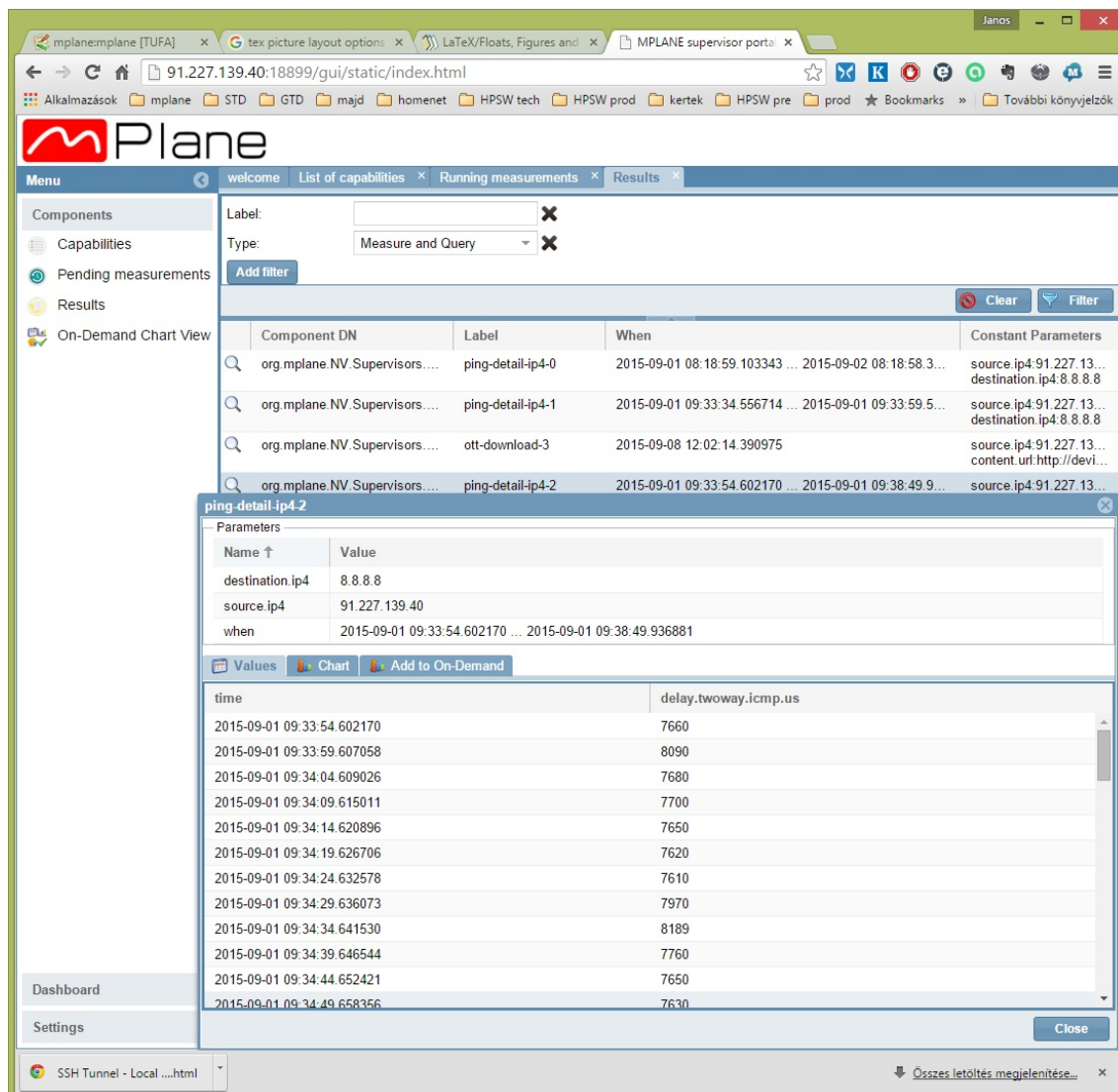


Figure 15: Supervisor GUI - Show numeric results

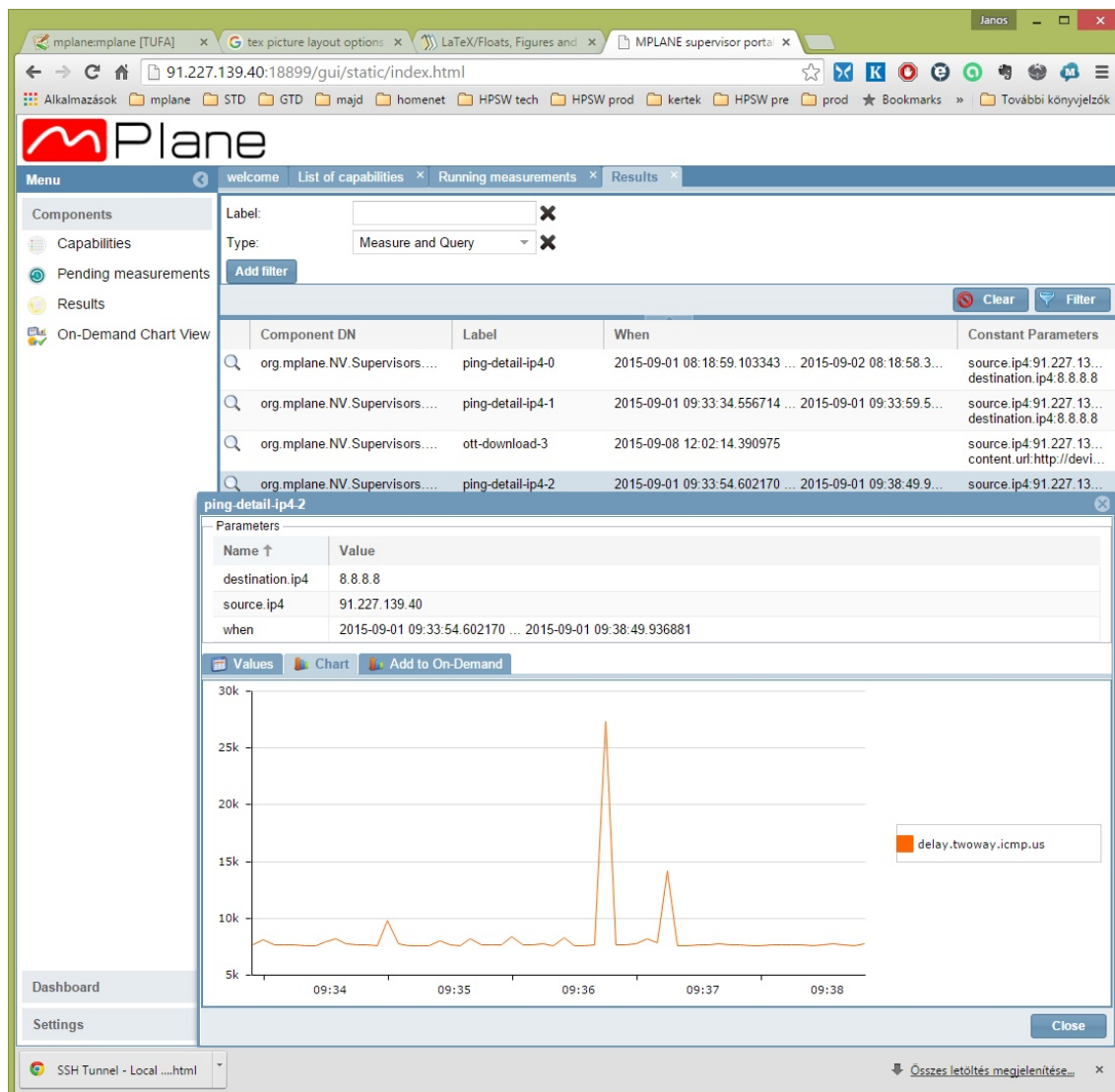


Figure 16: Supervisor GUI -Show results in graph

The definition, development and integration of the Supervisor GUI is a joint effort led by NETVISOR and TI; with other partners expected to collaborate in the future, with contribution of visualization components, accelerated, ergonomic control mechanisms, and further, UC-specific extensions.

Supervisor GUI architecture

The Supervisor GUI (`svgui`) has been implemented as a merged superset of the Clientshell and the Supervisor with an extra HTTP Listener client added to deal with the GUI. Thus when launched it can be used as a command line interface as well from the launching terminal window.

Usage

The GUI is accessible through a Web browser at `http://<svgui_host>:8899`. (The port and other `svgui` related parameters can be set in the corresponding config file under the `[gui]` tab.)

It can be launched from a terminal like

```
python3-m mplane.svgui --config conf/svgui.conf
```

The mPlane components use two-way public key encrypted communication via TLS, i.e. both party should authenticate itself during communication. Due to the dynamic nature of web access, we use here one-way authentication, so there is no need to install individual certificates in all of the browsers which access `svgui`.

2.3 Probes

Final release of the following tools has been included in D2.3 - versions on the webpage <http://www.ict-mplane.eu/public/software> are always to be considered the most up-to-date one.

Passive

- **Blockmon node:** a modular system for flexible, high-performance traffic (developed by NEC).
- **MobileProbe:** a tool for monitoring smartphone performance for Android devices (developed by TID).
- **Tstat:** a passive monitoring and DPI tool (developed by POLITO).

Active

- **Anycast:** a tool to detect, enumerate and geolocate anycast probes (developed by ENST).
- **ECN-Spider:** a tool to test ECN readiness and connectivity failure linked to ECN negotiation attempts (developed by ETH).
- **Fastping:** a fast ICMP scanner for TopHat (link is external) (developed by ENST).
- **GLIMPSE:** an end host-based network measurement tool (developed by FHA).
- **mSLAcert:** a tool for measurement of multi layer throughput and other active data for Service layer level agreement certification (developed by FUB).
- **OTT probe:** an active video content download performance evaluation tool (developed by NETVISOR).
- **RIPE Atlas probe:** a framework for distributed active measurements, using the RIPE Atlas platform (developed by FTW).
- **Scamper:** a sophisticated active probing tool (developed by CAIDA, integrated by ULG).
- **Tracebox:** a tool for topology discovery (developed by ULG).

Mixing active and passive

- **Firelog:** a hybrid probe for diagnosing Web browsing QoE (developed by EURECOM).

2.4 Repository tools

Final release of the following tools has been included in D3.4 - versions on the webpage <http://www.ict-mplane.eu/public/software> are always to be considered the most up-to-date one.

Query Engines

- **Blockmon Controller:** the controller for the distributed stream-processing platform Blockmon (developed by NEC).
- **DBStream:** a flexible and easy to use Data Stream Warehouse, designed for the on-line analysis of large amounts of network monitoring data (developed by FTW).
- **EZRepo:** an mPlane-compliant repository with measurement data preprocessing for root cause analysis (developed by NETVISOR).
- **MATH:** Mplane Authorized Transfer via HTTP, a tool to export bulk data in the form of logs from an mPlane probe (e.g., Tstat) and to import it into DBStream (developed by FTW).
- **Repository for Tstat RRDs:** a set of mPlane protocol-based tools to import logs and RRDs generated by Tstat (developed by POLITO and FTW).
- **MongoDB:** a proxy interface to use MongoDB as repository (developed by TID).
- **repoSim:** a ns2 based simulator to fine-tune the mPlane repository performance (developed by ENST and POLITO).

Schedulers

- **Hadoop Fair Sojourn Protocol:** a scheduler for Apache Hadoop (developed by EURECOM).
- **Schedule:** a tool for cache-oblivious scheduling of shared workloads (developed by FTW).

2.5 Reasoners and Analysis modules

Final release of the following tools has been included in D4.4 - versions on the webpage <http://www.ict-mplane.eu/public/software> are always to be considered the most up-to-date one.

Reasoners

- **nodejs reasoner**: basic mplane reasoner, written in nodejs (developed by TI)
- **Reasoner for Estimating Content and Service Popularity for Network Optimization**: it runs periodical analyses to estimate the popularity of contents and to issue a list of contents to cash in selected servers (developed by NEC and POLITO).
- **WeBrowse: Reasoner for Passive Content Curation Use Case**: orchestrates HTTP content extraction, analysis, filtering and promotion; it is able to perform on demand analysis on the popularity of web pages at different time scales (developed by ALBLF and POLITO).
- **Reasoner for Quality of Experience for Web Browsing Use Case**: it runs the diagnosis algorithms for QoE-related anomalies, using data collected by the Firelog probes and stored at a mPlane repository (developed by EURECOM).
- **Reasoner for Mobile Network Performance Issue Cause Analysis Use Case**: it is responsible for fetching new measurements from a mPlane repository for every available probe along a path from contents to customers, using the machine learning analysis modules to diagnose the problems (developed by TID).
- **mpAD_Reasoner: Reasoner for Anomaly Detection and Root Cause Analysis in Large-scale Networks Use Case**: orchestrates the detection and diagnosis of anomalies in large-scale services such as YouTube (developed by FTW).
- **Reasoner for Verification and Certification of Service Level Agreements Use Case**: it orchestrates the SLA-based testing phase, verifies that tests are properly done, and decide on whether to issue a certificate or not (developed by FUB).
- **RC1: Reasoner for Active Measurements for Multimedia Content Delivery Use Case**: it orchestrates the troubleshooting/diagnostic process, taking decisions based on metrics received from probes and/or repositories, using in addition network-topological information (developed by NETVISOR).
- **GLIMPSE End-Host Measurements**: traceroute-reasoner to determine multiple paths between a probe and a set target (developed by FHA).

Analysis Modules

- **Content Popularity Estimation Analysis Modules**: predicts the popularity of objects by analysing the global growth patterns and identifying individual sequences of growth and how they lead to one another (developed by NEC and POLITO).
- **Content Curation Analysis Modules**: Classification and Content Promotion (developed by ALBLF and POLITO).
- **SEARUM: Exploring correlations by means of distributed association rule mining**: a Hadoop MapReduce implementation of Association Rule Mining techniques (developed by POLITO).
- **Mobile Network Performance Issue Cause Analysis**: identify existence, location and class of detected problems (developed by TID).

- **iGreedy: Anycast Enumeration and Geolocation:** a tool able to detect, enumerate and geolocate anycast replicas (developed by ENST).
- **Spark Jobs: Quality of Experience for Web browsing Analysis Module:** these modules compute various statistics across data from different probes (developed by EURECOM).
- **Statistical Anomaly Detection:** detects abrupt changes in the statistical distribution of any monitored traffic feature (developed by FTW).
- **Entropy-based Detector:** detects anomalies based on the identification of abrupt changes in the empirical entropy of any monitored feature (developed by FTW).
- **Network Proximity Service - DisNETPerf:** performs analysis of reverse path performance using the RIPE Atlas framework (developed by FTW and ULG).
- **LPR: an MPLS Tunnel Diversity Module:** a traceroute-based technique to classify MPLS tunnels according to their actual purposes (developed by ULG).
- **Middlebox Taxonomy:** classifies middle-boxes according to the path impairment the cause (developed by ULG).
- **IGP Weight Inference:** infers IGP weights for an ISP network, based on collected traces (developed by ULG).

3 Integration tests

In this chapter we describe those tests done by Consortium Partners which should be considered as "shared" or "multi-use" integrations, because they will be used in several use cases. The use case specific tests are discussed in Chapter 4.

3.1 Tests performed by TI

TI Virtual Appliance testing

As a result of integration activities, a virtual appliance containing a preconfigured instance of a number of mPlane ready measures has been deployed. This section reports tests done on the virtual appliance in order to verify it can be deployed with one of the publicly available Supervisors to realize experiments. Since two type of libraries has been tested, tests are divided in SDK (python) probe and nodejs probe. Follows a third section with tests related to the WEB GUI.

Test #1	SDK proxy correctly download distributed registry	
Description	Status	Notes
Test that SDK proxy is able to access, download and integrate the chain of registries.	PASSED	--

Test #2	SDK proxy connects to the supervisor	
Description	Status	Notes
Test that SDK proxy provided in the appliance correctly connects to the supervisor.	PASSED	--

Test #3	Public supervisor recognises SDK capabilities	
Description	Status	Notes
Test that Probe export SDK python capabilities to the supervisor.	PASSED	--

Test #4	Specification to SDK measures	
Description	Status	Notes
Test that the supervisor can instantiate specification to the SDK python probe.	PASSED	--

Test #5	Results from SDK specification	
Description	Status	Notes
Test that the supervisor receives result from specifications send to a SDK based capability to the SDK python probe.	PASSED	--

Table 1: Python probe tests.

```
root@mplane:/usr/mplane/protocol-r1# ./start.sh
tracebox
tracebox
ping
ping
trace
tracelb
ping
ping
tracebox
tracebox
trace
tracelb
Added <service for <capability: measure (tstat-log_tcp_complete-core) when now ... future token 8a6c71ae schema 9003efe0 p/m/r 0/3/4>>
Added <service for <capability: measure (tstat-log_tcp_complete-end_to_end) when now ... future token eb52be13 schema c60ef114 p/m/r 0/3/7>>
Added <service for <capability: measure (tstat-log_tcp_complete-tcp_options) when now ... future token 5721f207 schema 339ca144 p/m/r 0/3/46>>
Added <service for <capability: measure (tstat-log_tcp_complete-p2p_stats) when now ... future token f2dea3dd schema aid8ed17 p/m/r 0/3/6>>
Added <service for <capability: measure (tstat-log_tcp_complete-layer7) when now ... future token 081ddab2 schema ac2cc50d p/m/r 0/3/4>>
Added <service for <capability: measure (tstat-log_rrds) when now ... future token 3155943c schema 00386e43 p/m/r 0/3/9>>
Added <service for <capability: measure (tstat-exporter_rrd) when past ... future token 65097e89 schema 1ccc4d88 p/m/r 1/3/3>>
Added <service for <capability: measure (tstat-log_http_complete) when now ... future token e0aeg9f7 schema f777d147 p/m/r 0/3/17>>
Added <service for <capability: measure (tstat-exporter_streaming) when now ... future token 0aa73b8a schema 6305d155 p/m/r 4/3/0>>
Added <service for <capability: measure (tstat-exporter_log) when past ... future token 7044cd15 schema 70085842 p/m/r 1/3/0>>
Added <service for <capability: measure (fastping-ip4) when now ... future / 1s token 5564696 schema dd3da093 p/m/r 11/0/1>>
Added <service for <capability: measure (anycast-detection-ip4) when now token d23b7dc8 schema 692f5397 p/m/r 2/0/1>>
Added <service for <capability: measure (anycast-enumeration-ip4) when now token 60b0ff1b schema 77c8e698 p/m/r 2/0/2>>
Added <service for <capability: measure (anycast-geolocation-ip4) when now token 3cdd5825 schema ad64613 p/m/r 2/0/2>>
Added <service for <capability: measure (blockmon-packets) when now ... future / 1s token 64fd4760 schema 7aefb9e p/m/r 0/3/2>>
Added <service for <capability: measure (blockmon-flows) when now ... future / 1s token a43a55d3 schema e19d4cb7 p/m/r 0/3/9>>
Added <service for <capability: measure (blockmon-flows-tcp) when now ... future / 1s token 315f4b68 schema 1204d683 p/m/r 0/3/9>>
Added <service for <capability: measure (blockmon-tstat) when now ... future / 1s token ddac63e schema 42edff40 p/m/r 0/3/0>>
Added <service for <capability: measure (scamper-tracebox-standard-ip4) when now ... future token 7ed3741c schema db8bf88 p/m/r 2/3/2>>
Added <service for <capability: measure (scamper-tracebox-specific-ip4) when now ... future token bbf56f7b schema 563619bb p/m/r 4/3/2>>
Added <service for <capability: measure (scamper-ping-average-ip4) when now ... future / 1s token 1da0b7b5 schema 306fa934 p/m/r 9/3/4>>
Added <service for <capability: measure (scamper-ping-detail-ip4) when now ... future token ee28140a schema fad18385 p/m/r 9/3/2>>
Added <service for <capability: measure (scamper-trace-standard-ip4) when now ... future token b412607a schema 1490ee2 p/m/r 19/3/3>>
Added <service for <capability: measure (scamper-tracelb-standard-ip4) when now ... future token b47cde40 schema b5a842a0 p/m/r 13/3/1>>
Added <service for <capability: measure (scamper-ping-average-ip6) when now ... future / 1s token 41234e1d schema d2487ae9 p/m/r 9/3/4>>
Added <service for <capability: measure (scamper-ping-detail-ip6) when now ... future / 1s token bbf56f7b schema 693e205e p/m/r 9/3/2>>
Added <service for <capability: measure (scamper-tracebox-standard-ip6) when now ... future token 36e893eb schema c2cfa207 p/m/r 2/3/2>>
Added <service for <capability: measure (scamper-tracebox-specific-quotesize-ip4) when now ... future token 10f2d70c schema d144021c p/m/r 4/3/2>>
Added <service for <capability: measure (scamper-tracebox-specific-quotesize-ip6) when now ... future token 7f49f498 schema f18264ca p/m/r 4/3/3>>
Added <service for <capability: measure (scamper-trace-standard-ip6) when now ... future token bb0270d0 schema 72e7d87f p/m/r 19/3/3>>
Added <service for <capability: measure (scamper-tracelb-standard-ip6) when now ... future token 1bc7131a schema 22c552e6 p/m/r 13/3/1>>
```

Figure 17: Test 1 - registry download

2015-09-10 08:10:56,379 Starting new HTTPS connection (1): supervisor.ict-mplane.eu

```
capability registration outcome:
scamper-tracebox-specific-ip6: ok
tstat-log_tcp_complete-end_to_end: ok
tstat-exporter_streaming: ok
scamper-trace-standard-ip6: ok
callback: ok
blockmon-flows-tcp: ok
scamper-tracebox-standard-ip6: ok
anycast-detection-ip4: ok
tstat-log_tcp_complete-p2p_stats: ok
scamper-tracebox-standard-ip4: ok
scamper-tracebox-specific-quotesize-ip4: ok
scamper-tracebox-specific-ip4: ok
scamper-ping-detail-ip6: ok
tstat-exporter_rrd: ok
tstat-exporter_log: ok
scamper-tracelb-standard-ip4: ok
ott-download: ok
anycast-enumeration-ip4: ok
scamper-ping-average-ip4: ok
tstat-log_tcp_complete-layer7: ok
tstat-log_tcp_complete-tcp_options: ok
tstat-log_tcp_complete-core: ok
blockmon-tstat: ok
scamper-trace-standard-ip4: ok
fastping-ip4: ok
scamper-ping-detail-ip4: ok
blockmon-packets: ok
scamper-tracebox-specific-quotesize-ip6: ok
anycast-geolocation-ip4: ok
tstat-log_rrds: ok
scamper-tracelb-standard-ip6: ok
scamper-ping-average-ip6: ok
tstat-log_http_complete: ok
blockmon-flows: ok

checking for specifications...
```

Figure 18: Test 2 - connection to the Supervisor

Above tables and pictures report on tests carried out.


```

|implane| listcap
Capability anycast-detection-ip4 (token d23b7dc83f81309e24886c271ecd551f)
Capability anycast-enumeration-ip4 (token 60b0ff1b56bb8be0cf6ddcd53e78640)
Capability anycast-geo-location-ip4 (token 3cd58254c702238170ef0047ccd6f9e)
Capability blockmon-flows (token a43a53d3cddf1916c9d00d64e41540e0)
Capability blockmon-flows-tcp (token 515f4b686bdd81870313af8ae1c0794c)
Capability blockmon-packets (token 64fd476051e0fa5ccf2209f12727060)
Capability blockmon-tstat (token ddac6b3ed50e445b4bf53cf483060a2a)
Capability callback (token 96a966ee70215088afca9a5e8e2a2ce3)
Capability fastping-ip4 (token 55664696bcd29d96bcc1d0b050525502)
Capability ott-download (token 00f2b47680fba943e9827839a42d981)
Capability scamper-ping-average-ip4 (token e05e80718551d3be969d414343132c17)
Capability scamper-ping-average-ip6 (token 557e9871af6b4d76669a14369280efd5)
Capability scamper-ping-detail-ip4 (token 8ccfcd10431f5e0a2ff20311c885c812)
Capability scamper-ping-detail-ip6 (token cd2ed52287fbf38f4809381cb62ebd39)
Capability scamper-trace-standard-ip4 (token 61a9111536736f03de061c0665dcea7c)
Capability scamper-trace-standard-ip6 (token 704c688ef2eeb9e0982967c3bc80fa81)
Capability scamper-tracebox-specific-ip4 (token bbf56f7b24b093d37cac7429a1997697)
Capability scamper-tracebox-specific-ip6 (token 10f2d7bc753b56d61ce56b28b39bd1af)
Capability scamper-tracebox-specific-quotesize-ip4 (token 19a10be1b0adce437ecc6c050179dac4)
Capability scamper-tracebox-specific-quotesize-ip6 (token 7f49f498896dfef952c1a40da70ac222)
Capability scamper-trace-standard-ip4 (token 7ed3741c98afd17ff96227f3c5995fd4)
Capability scamper-trace-standard-ip6 (token 36e893eb7c70bb5c90fb62c0bb377377)
Capability scamper-tracelb-standard-ip4 (token f6b60bab77defc387de845a755d3399a)
Capability scamper-tracelb-standard-ip6 (token cb67738db2bc586e9b3be59673aad50e)
Capability tstat-exporter_log (token 7044cd1530a7b92f34e82613935a8fac)
Capability tstat-exporter_rrd (token 65097e89ede579197e5413f5086c5e09)
Capability tstat-log_streaming (token 0aa73b8ae670fcc69cf7852cf3865c9b)
Capability tstat-log_http_complete (token e0aea9f776493d3b7493f7a013e29bf0)
Capability tstat-log_rrds (token 9155945c8ca531377f3785a33e39fc23)
Capability tstat-log_tcp_complete-core (token 8a6c71ae4b58c1311d81a6c505d29fc6)
Capability tstat-log_tcp_complete-end_to_end (token eb52be13bfe3f06f1080619e025b3a22)
Capability tstat-log_tcp_complete-layer7 (token 081ddab20b9cbd05f1e032146afeb403)
Capability tstat-log_tcp_complete-p2p_stats (token f2dea3ddd23ac4354bc1f2141ba0a467)
Capability tstat-log_tcp_complete-tcp_options (token 5721f207dd78693479b2a7219ef20ecc)

```

Figure 19: Test 3 - capability registered on the Supervisor

Test #6	Nodejs probe correctly download distributed registry	
Description	Status	Notes
Test that nodejs probe is able to access, download and integrate the chain of registries.	PASSED	--

Test #7	Nodejs probe connects to the supervisor	
Description	Status	Notes
Test that nodejs probe provided in the appliance correctly connects to the supervisor.	PASSED	--

Test #8	Public supervisor recognises nodejs probe capabilities	
Description	Status	Notes
Test that nodejs probe export capabilities to the supervisor.	PASSED	--

Test #9	Specification to nodejs measures	
Description	Status	Notes
Test that the supervisor can instantiate specification to the nodejs probe.	PASSED	--

Test #10	Results from nodejs specification	
Description	Status	Notes
Test that the supervisor receives result from specifications send to a nodejs based capability.	PASSED	--

Table 2: Nodejs probe tests.

```
|mplane| showcap tstat-log_tcp_complete-core
capability: measure
  label      : tstat-log_tcp_complete-core
  link       :
  token      : 8a6c71ae4b58c1311d81a6c505d29fc6
  when       : now ... future
  registry   : http://www.ict-mplane.eu/registry/demo
  metadata   ( 3):
                System_type: tStat
                System_ID: tStat-Proxy
                System_version: 0.1

results      (42):
  source.ip4
  source.port
  packets.forward
  packets.forward.syn
  packets.forward.fin
  packets.forward.rst
  packets.forward.ack
  packets.forward.pure_ack
  packets.forward.with_payload
  packets.forward.rxmit
  packets.forward.outseq
  bytes.forward
  bytes.forward.unique
  bytes.forward.rxmit
  destination.ip4
  destination.port
  packets.backward
  packets.backward.syn
  packets.backward.fin
  packets.backward.rst
  packets.backward.ack
  packets.backward.pure_ack
  packets.backward.with_payload
  packets.backward.rxmit
  packets.backward.outseq
  bytes.backward
  bytes.backward.unique
  bytes.backward.rxmit
  start
  end
  duration.ms
  source.TTFP.ms
  destination.TTFP.ms
  source.TTLP.ms
  destination.TTLP.ms
  source.TTFA.ms
  destination.TTFA.ms
  source.ip4.isinternal
  destination.ip4.isinternal
  tstat.flow.class.conn
  tstat.flow.class.p2p
  tstat.flow.class.http
```

Figure 20: Test 3 - capability registered on the Supervisor

```
|mplane| runcap blockmon-flows
|when| = now ... future / 1s
ok
Specification blockmon-flows-0 successfully pulled by eu.ict-mplane.probeTI
Specification blockmon-flows-1 successfully pulled by eu.ict-mplane.probeTI
Specification blockmon-flows-2 successfully pulled by eu.ict-mplane.probeTI
Specification blockmon-flows-3 successfully pulled by eu.ict-mplane.probeTI
Specification blockmon-flows-4 successfully pulled by eu.ict-mplane.probeTI
Specification blockmon-flows-5 successfully pulled by eu.ict-mplane.probeTI
```

Figure 21: Test 4 and 5 - specifications and results

```
root@mplane:/usr/mplane/TI_generic_probe# node probe.js --debug
INFO: Local Address::192.168.43.189
INFO: Supervisor:supervisor.ict-mplane.eu@8889
+ Registry downloaded from http://www.ict-mplane.eu/registry/demo (225)
+ Registry downloaded from http://ict-mplane.eu/registry/demo (225)

+++++
+ Elements in my registry 450
+++++
```

Figure 22: Test 6 - registry downloaded

Test #11	Capabilities exposed to the GUI	
Description	Status	Notes
Test capabilities are visible from the GUI.	PASSED	--

Test #12	Running measurements exposed to the GUI	
Description	Status	Notes
Test running specifications are visible from the GUI.	PASSED	--

Test #13	Results exposed to the GUI	
Description	Status	Notes
Test results of specification are visible from the GUI.	PASSED	--

Table 3: WEB GUI tests.

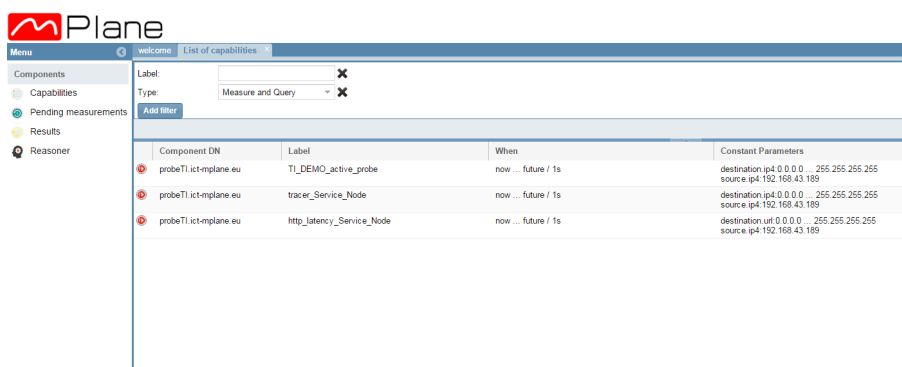


Figure 23: Test 11 - capabilities exposed to the GUI

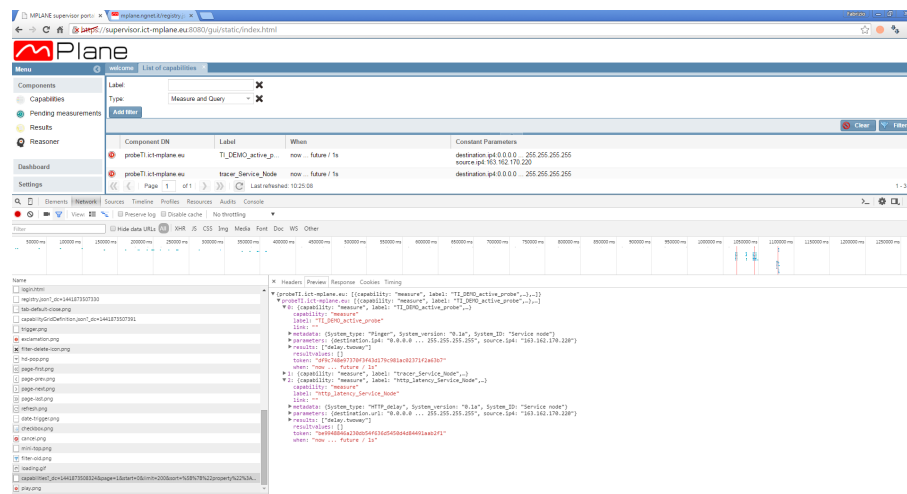


Figure 24: Test 11 - capability details

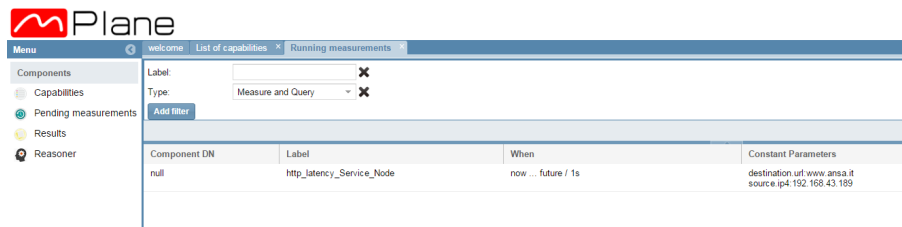


Figure 25: Test 12 - pending measures



Figure 26: Test 12 - pending measures

```
root@mplane:/usr/mplane/TI_generic_probe# node probTl.js
INFO: Local Address::163.162.170.220
INFO: Supervisor:supervisor.ict-mplane.eu@8890
+ Registry downloaded from http://mplane.ngnet.it/registry.json (254)
+ Registry downloaded from http://www.ict-mplane.eu/registry/demo (225)
+ Registry downloaded from http://mplane.ngnet.it/registry_anycast.json (116)
+ Registry downloaded from http://mplane.ngnet.it/registry_fastping.json (123)
+ Registry downloaded from http://mplane.ngnet.it/registry_scamper.json (289)

+++++
+ Elements in my registry 1007
+++++
*****
REGISTERING MY CAPABILITIES
*****

-----
checking for specifications...

-----
INFO: Something to do for me...(8.8.8.8)
delay.twoway <8.8.8.8>:6.11
INFO: Something to do for me...(HTTP Latency: www.ansa.it)
Request took: 39 ms
```

Figure 27: Test 12 - Specification received

3.2 Tests performed by FUB

Most of the tests carried in such a test bed and regarding the reliability of the SLA verification and certification methods defined in Mplane can be found in [1]. Recently the integration tests carried out in this Lab have regarded the correlation between active and passive QoS measurements trying to understand the relation among some information mainly regarding the congestion windows and the throughput degradation with respect the line capacity. This approach was adopted also to make a comparison among our mSLAcert method and other current methods to verify the QoS in high speed access networks (100 Mb/s), with particular regards to the last ETSI method defined in 2014 [2].

The experimental investigation was carried out in this test-bed [3], even though for our aims it was used only in part as reported in Fig. 28. In particular two HP prodesk 490 G1 mt were adopted as server and client, with 16GB of RAM and a 1000Mbps network interfaces; on both the machines Linux OS (Ubuntu 14.04) was adopted. The client had installed on his PC the Agent probe, and it was connected to the network through a Netgear FS605 switch, which on his end is connected to a Alcatel 7750 router. We have also introduced an bit error rate corruption, with different combinations of RTT.

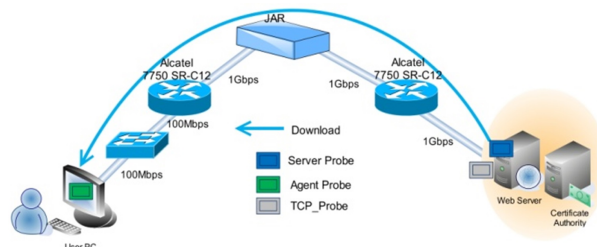


Figure 28: Reduced experimental set-up to investigate the correlation between active and passive QoS measurements.

The Alcatel routers were connected to JAR, a Network emulation device, which can process traffic up to 10Gbps. We have used JAR to introduce additional RTT to the network and packet loss. And on the server side we have the same, with the only difference that the server directly connects to the router through a 1000Mbps connection. The active probe adopted a data transfer of 2GB using two different types of TCP standards: CUBIC and New Reno. A report of the throughput every second was set; furthermore Tcp_probe produced a list of a few various data as the threshold window and congestion window. We also carried out experiments under various bit error rate, RTT, jitter and packet losses.

All the experiments were done with PC default settings, since the aim of this work is to investigate on the evaluation of the line capacity at the disposal of the user, to verify and certify his SLA and not the tuning of the PC parameters to improve the performance. Indeed, most users do not have sufficient knowledge to proceed to change such parameters. Also from the server side, the server cannot adapt his settings to network condition for each user, since it would be not practical and it would waste a lot of service time. All this experimental investigation has been reported in the submitted paper “Experimental investigation on TCP throughput behavior in High Speed Access Networks” to IET Networking. For sake of brevity here we only report the correlation between the active measurement (Throughput) for TCP New Reno and Cubic (Fig. 29) and the congestion windows (Fig. 30) for a 100 Mb/s access in the case of a RTT=100 ms.

For example, from these results, we can see that New Reno, takes much more time to transmit the data, and to increment his throughput up to reach the bandwidth corresponding to the line capacity. On the other hand New Reno can reach a higher throughput value with respect to CUBIC.

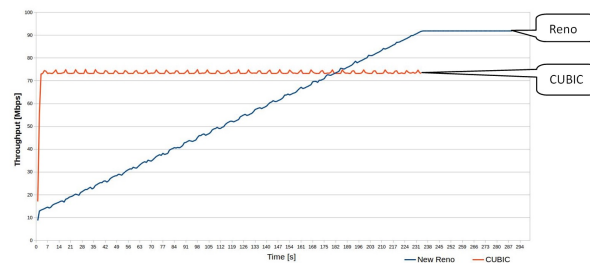


Figure 29: Throughput vs. time, during the transfer of 2048MB of data on a line of 100Mbps with an RTT of 100ms and Jitter 0ms.

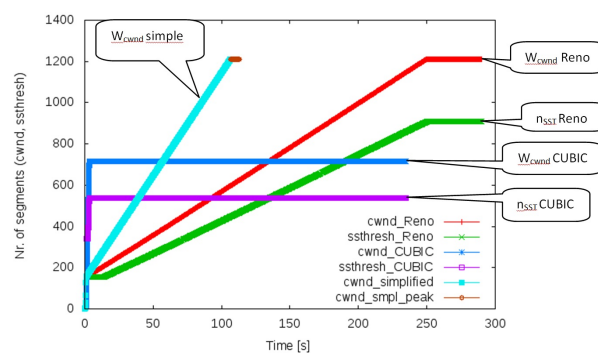


Figure 30: Passive monitoring of the congestion window and threshold window, for a line of 100Mbps and a RTT of 100ms.

3.3 Tests performed by Polito

The tests performed in Polito premises have been run for multiple purposes. First, to test in a real scenario all the indirect export capabilities offered by Tstat, i.e., RRD, streaming and bulk exporters. Second, to test the integration with repositories and analysis modules which rely upon Tstat data to run analysis.

Figure 31 resumes the integration tests that have been performed in Polito premises so far (solid interaction lines). Interactions that are planned to be performed soon are represented with dotted lines.

In the following we resume the status of the testing activities for each use cases and extra task related to Tstat.

Passive Content Curation: a complete, fully mPlane-compliant deployment of the demonstration for this use cases is installed in Polito network. The deployment builds on two machines, one running Tstat probe and its mPlane interface, and another one running the supervisor and the interface for the WeBrowse module. The demo is available at <http://webbrowse.polito.it>. We are currently testing the interactions with the reasoner and the analysis modules.

Content Popularity Estimation: we are deploying the demonstration for this use cases on three machines: the first runs Tstat, a second runs the supervisor and a third runs MongoDB repository and the analysis modules. The reasoner for this use case will be deployed soon.

Automatic Root Cause Analysis for Network Troubleshooting: we are deploying the demonstration for this use cases on three machines: the first runs Tstat, a second runs the standard mPlane Supervisor and the specific mpAD_Reasoner, and a third runs an instance of the DBStream repository and the Anomaly Detection analysis module. All these components are currently running with real traffic and interact through the mPlane RI interfaces. The data exporting/importing from Tstat to DBStream is achieved through the MATH protocol, specifically considering two types of Tstat logs: `log_tcp_complete` and `log_video_complete`. In addition, we have been testing the RIPE Atlas proxy integration, running the DisNETPerf Analysis Module to monitor the performance of the inter-AS paths from selected YouTube servers towards the monitoring vantage point.

Visualization of Tstat Data: a fully mPlane-compliant deployment of this demonstration is deployed in Polito network. This demonstration builds on two machines: one running Tstat probe and its mPlane interface, and another one running the supervisor, the repository Graphite, the tool for data visualization, and its mPlane interface.

Next, we describe for each machine available in the testing platform its hardware details and the components hosted.

- The Tstat probe (DPDK version) runs on a machine (Server 1 in Fig. 31) connected to the backbone link of the campus network and generates logs and Round Robin Databases (RRDs) obtained by observing the traffic flowing. This machine runs the mPlane interface for Tstat and exports all its capabilities for measuring and exporting tasks. The machine is an Intel Xeon equipped with an 8-core 2.7GHz CPU, a 16GB RAM memory and installing a NIC interface with two 10GB/s ports.
- A second Intel Xeon machine (Server 2 in Fig. 31), equipped with a 2.53 GHz 16-core CPU, a 16GB RAM memory executes the Python3 supervisor of the mPlane reference implementation. The supervisor proxies all interactions among the components installed within the Polito premises, and allows clients (even external to Polito network) to connect. The same machine hosts the mPlane compliant version of WeBrowse, developed within the use case "Passive Content Curation". In this case, the analysis modules building WeBrowse have been embedded in an instance

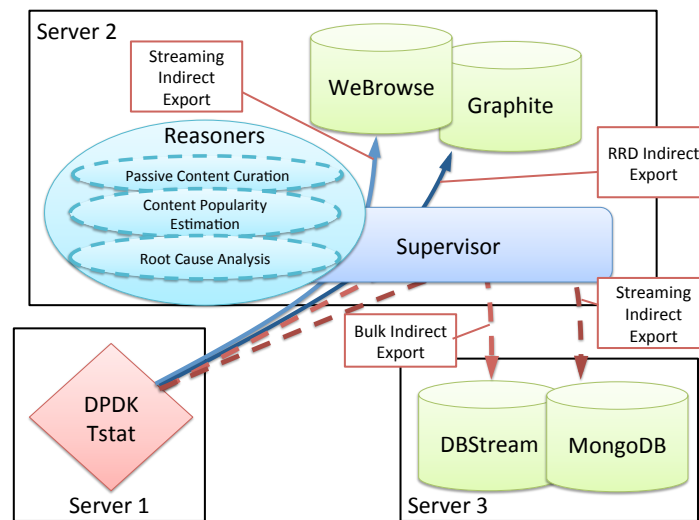


Figure 31: The integration platform available in Polito network, the deployed components and tests performed (solid lines) and to perform in near future (dashed lines).

of the Tstat repository. This repository interface receives a real-time stream of Tstat logs generated by the probe. Then, the analysis modules process such data to feed the WeBrowse website <http://webbrowse.polito.it>.

- A third Intel Xeon machine (Server 3 in Fig. 31), equipped with a 2.53 GHz 16-core CPU, a 16GB RAM memory executes the repository DBStream, that is fed with the logs exported by the Tstat probe using the bulk indirect export of logs based on MATH protocol. The same machines hosts MongoDB and the analysis modules related to the Content Popularity Estimation use case.

4 Current state of Use Case prototypes

In this chapter we report the (integration) status of the Use Cases, which has been selected as part of the demonstration phase of the project.

4.1 Estimating Content and Service Popularity for Network Optimization

The goal of this use case is to optimize the QoE of the user and the network load by inferring the expected-to-be popular contents and identifying optimal objects to cache in a given portion of the network. To achieve this goal, we exploit the mPlane architecture in order to collect a large number of online traffic information requested by the users in several points in the network. The acquired information is exploited in order to predict the content popularity and suggest efficient caching replacement strategies to the Reasoner.

The code for this use case has been released on GitHub at <https://github.com/fp7mplane/demo-infra> and the instructions on how to configure and run it are provided on the official mPlane website at <https://www.ict-mplane.eu/public/content-popularity-estimation>. The page describes step by step instructions for the probe, the supervisor, the repository and the analysis module.

The testing of each component (being it probe, repository, supervisor or reasoner) and how it communicates with the remaining architecture is ongoing.

Test #1		Probe functionalities	
Description	Status	Notes	
Test that the probe can extract from network traffic the information needed, i.e., video id and timestamp.	Done	--	

Test #2		Repository functionalities	
Description	Status	Notes	
Test that the repository can correctly store and retrieve the information.	Done	--	

Test #3		Analysis module functionalities	
Description	Status	Notes	
Test that the analysis module can predict content popularity.	Done	--	

Test #4		Reasoner functionalities	
Description	Status	Notes	
Test that the reasoner can infer a list of content to put into the cache.	Done	--	

Table 4: Tests for the use case "Estimating Content Popularity for Network Optimization".

4.2 Active Measurements for Multimedia Content Delivery Use Case

Test environment

Figure 32 shows a typical MultimediaContent Delivery test network environment. It contains a full set of mPlane components for integration testing:

- The standard Supervisor (with `svgui`)
- Probes)
- An EZrepo repository receiving results from all probes.
- An instance of the "RC1" [Root Cause 1] Reasoner.

The architecture is fully MPlane conformant, i.e. all components communicate through mPlane interfaces as outlined in WP1.

We have done the tests it in NETvisor's extended network environment, with standard mplane probes (GLIMPSE, OTT probe, pinger). The supervisor (with GUI), the EZrepo and the RC1 has been deployed on the same virtual server. During the tests we used probes on different platforms (Ubuntu servers and Miniprobes), installed at NETvisor, at an ISP's virtual server, and at residential (home) premise respectively, as shown in Figure 32.

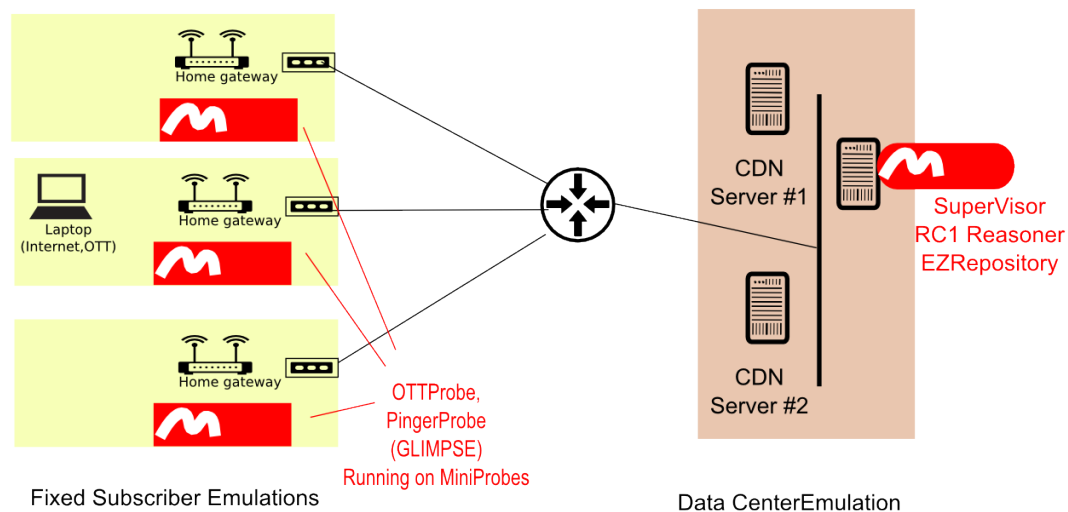


Figure 32: Multimedia Use Case test network architecture

EZrepo repository

One key component of the MMUC architecture is EZRepo which:

- receives measurement results from probes through "indirect export" protocols like UDP and TCP (under development)(also allowing for SSL, if needed).
- evaluates the incoming results through an Mplane configurable classifier, which applies quality-level "grades" to each record.

- stores the records in its data store.
- makes it possible to execute searches through the `QueryByCriteria` capability against the measurement results stored. This capability accepts a filter composed from parameters, and a *grading criteria*. The returned results show the total number of measurements found and the element count of the subset with matching grades.

RC1 reasoner

The other principal component is the RC1 Reasoner. It is a rule-based reasoner that periodically executes queries (mainly from the repository) and based on the results, it draws conclusions, i.e.

- identifies an issue: it launches further measurements, or executes further queries
- identifies hypothesis: starts a targeted proof of a concrete problem by further measurements and queries
- identifies a diagnosis: states that it has identified a problem with a high probability.

It is to be noted that the reasoner is *topology-aware*: it should now about probes, servers and the network paths between them. This is an important source of diagnosis information, but also critical in the sense that credible topology DB needs to be maintained in the reasoner. For the integration tests we are using a preconfigured, fix topology, but we acknowledge that this could become a key task for a large-scale deployment. We are planning to do the next step, i.e. start using dynamic topologies in the demo phase (WP6).

Integration tests

Test #1		OTT-Probe test	
Description	Status	Notes	
OTT-Probes can actively access content on CDN and produce credible metrics or detect errors when content is not available.	Done	OTT probe tested from different premises (NETVISOR Testbed, external provider's network and residential premises respectively), on different platforms (OpenWrt on Miniprobe, Ubuntu14.04). See attached measurement screenshot 33	

Test #2		Passive probe tests	
Description	Status	Notes	
Tstat and MobileProbes can inspect passing OTT traffic, and indicate problems if those occur.	in progress	Probes deployed and run at NETvisor site. Evaluation of long-term measurements is in progress.	

Test #3		Repository	
Description	Status	Notes	
All probe measurements are available in the central repo, or in the on-Probe repositories (like EZRepo).	Done	On-probe repository ported and installed, data collection set up.	

Test #4		Reasoner and Supervisor functionalities	
Description	Status	Notes	
Supervisor receives triggers and reasoner will successfully provision measurements on some Probes	Done (as integration tests)	RC1 Reasoner deployed and set up, together with GLIMPSE, OTT and pinger probes. Measurement triggering works with updated RC1 reasoner.	

Table 5: Tests for the Active Measurements for Multimedia Content Delivery Use Case

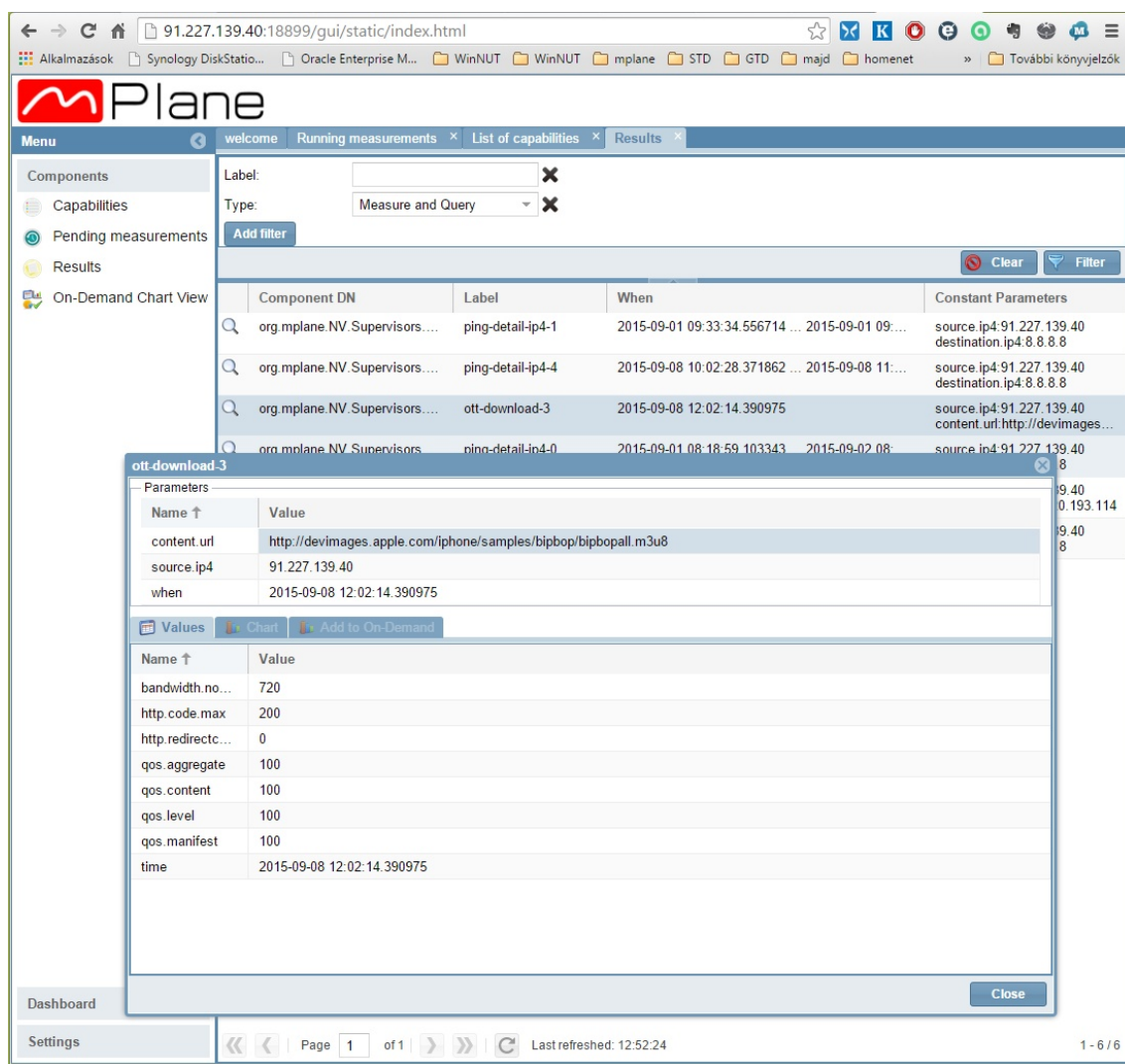


Figure 33: Multimedia Use Case tests - test 1

Functional tests

While integration testing focuses on the technical interoperability of the components used, the goal of functional tests is to prove whether the system assembled above is indeed capable of monitoring and troubleshooting multimedia content delivery services. These tests obviously need realistic and complete network environments, where it is possible to simulate errors at different parts of the system.

Tests of this kind have also begun, on NETvisor's test network described above. This is a rather minimal test network, but contains three active probes in "customer premises" and two central content servers, enough to analyze system response to various kinds of errors.

Initially, we are considering the following problem types:

- (A) Partial internet access problem (line works but slow) for a subscriber
- (B) Either of the content server is down
- (C) Media (http) server on content server is down or slow.
- (D) Some piece of OTT content is missing from either server.
- (E) Manifest error on some piece of content (i.e. error in content creation process).

The RC1 resoner is configured with simple rules to diagnose problems and start additional measurements for verification:

- Initially each of the 3 probes measures the ping availability of both servers (10 pings per minute per direction), and each probe makes a test content download from a fixed server in every 5 minutes.
- The Server down events (B) are diagnosed practically immediately, using a single rule, without requiring additional measurements.
- If pings fail from one location only, then error of type (A) is suspected, but not immediately diagnosed.
- Another rule is configured to start additional content downloads from other probes if a probe indicates an OTT access problem or if an external trigger (e.g. a real customer complaint) is applied. The probe which reported the problem is configured to access some other content from some other server.
- After 5 minutes the the resoner evaluates the results and identifies if the problem is general, or client specific.
- The resoner can thus make a diagnosis between the cases A, C, D and E.

Table 34 shows the fidelity of decisions as a function of time, by decision categories (i.e. the number of correct and incorrect decisions over time in a given event class). The error events were generated in an automated way, and diagnosis time was recorded, along with the correctness of diagnosis. In case the diagnosis has changed in response to an error, both diagnosis results are shown.

Error Scenario ID and Description	RCA Diagnosis time (minutes) and outcome							
	< 1	1 to 3	3 to 5	5 to 7	7 to 10	10 to 15	15 to 20	> 20
A- Access problem	*	*						
	*	*						
	*	*						
B -Content server down			*					
		*	*		*			
			*	*				
C -Media server down				*	*			
				*	*			
				*	*	*		
D -Ott content is missing			*	*		*		
			*	*		*		
			*	*				
E- Manifest error /ingress problem	*	*					*	
	*	*						
	*	*						
0 – Return to normal operation				*		*	*	*
						*	*	*
						*	*	*
Diagnosis outcome codes * =correct * =incorrect								

Figure 34: Fidelity of decisions by event categories over time

We conclude that at the current stage of functional tests the results are satisfactory, but at the minimum, so additional investigation of the mechanism and tuning of parameters is required to get a convincing result. It is also speculated, how our mechanisms will work in a real environment, with more devices and content in each role. The remaining tests planned for WP5 and WP6 will probably answer these questions.

4.3 Quality of Experience for Web Browsing Use Case

The QoE use case aims at identifying the root cause for a poor performance in web browsing. The Firelog probe will collect passive and active measurements and return a local view diagnosis (single probe) for a specific session. Exported data will be processed and aggregated to be further exploited for providing a global diagnosis for a specific browsed url.

A basic demo needs only a single Firelog probe, which integrates all the logic for a single vantage point.

Extended demo needs: a set (or just one) Firelog probe and the mPlane supervisor for controlling them; the DISC repository and the Reasoner web application running the diagnosis algorithm.

From the Deliverable D.5.2, here is the status.

Test #1		Probe functionalities	
Description	Status	Notes	
Browsing, network sniffing, diagnosis, exporting.	Done	--	

Test #2		mPlane Probe compliancy	
Description	Status	Notes	
Probe management via mPlane supervisor.	Done	Tested on the master branch on github	

Test #3		mPlane Repository compliancy	
Description	Status	Notes	
Repository interaction via its component.	Pending	indirect export is used for probe-to-repo communication	

Test #4		Reasoner interaction	
Description	Status	Notes	
Computing diagnosis from data on the repository.	Done	Not through the repository component	

Table 6: Tests for the use case "Quality of Experience in Web Browsing session".

4.4 Mobile Network Performance Issue Cause Analysis Use Case

This use case tackles the identification of poor QoE on mobile video delivery. It is composed of three main components: a set of probes (as described in WP2), a repository that collects the data in JSON format (as described in WP3) and a machine-learning-based reasoner (as described in WP4). Each of these components have been tested individually and the whole integrated test has been done in three phases: i) controlled (laboratory) experiments where we control the quality of the network and we induce faults, ii) semi-control experiments where the videos are played through normal Internet links but faults are induced artificially and iii) in-the-wild experiments where we didn't interfere with the video delivery process. In all three cases the system was able to successfully identify the root cause of the problems (e.g., accuracy was >85%).

The code for this use case and the instructions on how to configure and run it are provided on the official mPlane website at <https://www.ict-mplane.eu/public/mobile-probe-android>. Furthermore, the proxy that allows to pull data from the repository using the mPlane reference implementation can be found here: <https://github.com/fp7mplane/components/blob/master/mongoDB-mobileProbe/>

Test #1		Probe functionalities	
Description	Status	Notes	
Test that probes can monitor video delivery from all vantage points	Done	--	

Test #1		Video server	
Description	Status	Notes	
Test Video servers both in TID and in FW premises	Done	--	

Test #2		Repository functionalities	
Description	Status	Notes	
Test that the mongoDB repository can correctly store and retrieve the information.	Done	--	

Test #2		Repository functionalities	
Description	Status	Notes	
Test that the mongoDB proxy can extract data using the Reference Implementation	Done	--	

Test #3		Analysis module functionalities	
Description	Status	Notes	
Test that the machine learning component can detect the root cause of video delivery using our dataset both in controlled and in-the-wild experiments.	Done	--	

Test #4		Automation and visualization	
Description	Status	Notes	
Test that the reasoner automatically performs the whole work-flow and visualize the results	In progress	--	

Table 7: Tests for the use case.

4.5 Anomaly Detection and Root Cause Analysis in Large-scale Networks Use Case

Test environment

Recall that the Anomaly Detection and Root Cause Analysis in Large-scale Networks Use Case targets the continuous monitoring of large-scale network traffic, aiming at detecting and diagnosis anomalies potentially impacting a large number of users. In preparation for its deployment, we have assembled a full set of components for integration testing. The following components are currently deployed at Polito premises, and we plan to fully migrate the complete setup to the Fastweb test plan for demo purposes:

- The standard Supervisor.
- Two types of probes: Tstat and RIPE Atlas.
- Different Analysis Modules: ADTool for anomaly detection and DisNETPerf for distributed active measurements.
- A DBStream repository receiving results from the probes, and continuously running the Anomaly Detection module.
- An instance of the mpAD_Reasoner Reasoner.

The architecture is fully mPlane compliant, i.e. all the components communicate through mPlane interfaces, relying on the mPlane RI. Below we briefly describe the role of each component:

- **Tstat:** passive measurements are performed through a Tstat passive probe, attached to a PoP-link aggregating a large number of users.
- **RIPE Atlas Proxy:** active measurements are performed through the RIPE Atlas monitoring framework, relying on the mPlane RIPE Atlas proxy to instantiate new measurements.
- **DBStream:** a DBStream repository stores the passive and active measurements obtained from the aforementioned probes, and runs the Anomaly Detection module which shall unveil the traffic anomalies. Data importing from Tstat to DBStream is achieved through the mPlane MATH protocol.
- **Supervisor & Reasoner:** a Supervisor with an integrated Reasoner (mpAD_Reasoner) supports the mPlane components registration and the iterative data analysis process.
- **mPlane GUI:** a mPlane GUI attached to the Supervisor (svgui) permits to manage and visualize the corresponding components.

Integration tests

Table 7 reports the current status of the integration tests as foreseen in deliverable D5.2. In particular, 6 out of the 9 planned tests are already in Done status, whereas the remaining 3 are in progress.

Test #1	Probe functionalities
---------	-----------------------

Description	Status	Notes
Test that the probe can extract from network traffic the YouTube related data.	Done	Tstat's log_video_complete extracted from real traffic at Fastweb premises offers visibility on YouTube related data.

Test #2	Probe functionalities	
Description	Status	Notes
Test that the RIPE Atlas framework measurements can be correctly instantiated and retrieve from the mPlane framework.	Done	Several tests consisting of instantiating standard ping and traceroute measurements among RIPE Atlas boxes using the RIPE Atlas mPlane proxy have been successfully conducted. Measurements are stored at the DBStream repository for further analysis.

Test #3	Repository functionalities	
Description	Status	Notes
Test that DBStream can correctly store and retrieve the information.	Done	Several tests involving the importing of data from Tstat to DBStream using the MATH protocol were conducted, both in a local deployment at FTW, as well as at Polito premises, using real traffic data.

Test #4	Analysis module functionalities	
Description	Status	Notes
Test that the anomaly detection analysis module can correctly detect significant anomalies.	Done	The AD Analysis Module has been running on top of DBStream for a couple of months and is capable of detecting anomalies, both at the core of a cellular network (at FTW premises) and at Polito premises.

Test #5	Reasoner functionalities	
Description	Status	Notes
Test that the reasoner can correctly instantiate new active measurements once an anomaly has been detected, to test the performance of the downlink paths.	in progress	--

Test #6	Probe to repository communication	
Description	Status	Notes
Test that the probe can periodically export the information towards the repository.	Done	Data importing/exporting between Tstat and DBStream is currently running at Polito premises, using the MATH protocol.

Test #7	Repository to analysis module communication	
Description	Status	Notes

Test that the anomaly detection analysis module can retrieve data from the repository.	Done	The AD Analysis Module directly runs on top of DBStream and we have verified that in can normally retrieve and store measurements at the corresponding tables.
--	------	--

Test #8	Analysis module to reasoner communication	
Description	Status	Notes
Test that the reasoner can retrieve anomaly alarms from the analysis module.	in progress	--

Test #9	Integration of the all components	
Description	Status	Notes
Test that the overall system functions properly.	in progress	--

Table 8: Tests for the Anomaly Detection and Root Cause Analysis in Large-scale Networks Use Case.

4.6 Verification and Certification of Service Level Agreements Use Case

The goal of mSLAcert is the verification and certification of service-level agreements. This is done in terms of RTT and throughput. The probe measures RTT by calculating the mean of ten samples and the throughput is measured using two different transfer protocols, TCP and UDP, as described in D2.2. The probe is fully developed and is mPlane compliant. The code is available online, on the components folder of mplane repository. Further detail on how to run the probe are available on <https://www.ict-mplane.eu/public/mslacert-active-probe>.

mSLAcert was tested in virtual environment, with Ubuntu (linux) machines, the probe was adopted in FUB testbed where its reliability was verified in twisted pair (ADSL2+, VDSL) and optical fiber (GPON) accesses. FUB is collaborating with Fastweb to implement mSLAcert on their testbed and testing it remotely. mSLAcert was added to the virtual machine developed by TI.

Test #1		Probe functionalities
Description	Status	Notes
RTT test.	Done	Tests were carried out in FUB LAB in GPON access over two days, in detail and average, introducing different values of packet losses.

Test #2		Probe functionality
Description	Status	Notes
TCP throughput TCP throughput.	Done	Tests were carried out in FUB LAB in GPON access over two days, in detail and average, introducing different values of packet losses.

Test #3		Probe functionality
Description	Status	Notes
UDP throughput.	Done	Tests were carried out in FUB LAB in GPON access over two days, in detail and average, introducing different values of packet losses average.

Test #4		Probe functionality
Description	Status	Notes
TCP and UDP throughput.	In progress.	Tests were carried out in FUB LAB in GPON access over two days, in detail and average, introducing different values of packet losses including some core bandwidth bottleneck. Comparison between TCP and UDP values.

Test #5		Probe functionality
Description	Status	Notes
TSTAT measurement of TCP throughput.	In progress.	Comparison between active and passive tests were carried out adopting proprietary passive traffic analyser of the LAB.

Table 9: Tests for the use case "Service Level Agreement".

4.7 Passive Content Curation Use Case

The goal of this use is to help users finding relevant content on the web, based on the passive observation of content requests flowing in the network. The intuition behind it is that the crowd of users acts as smart robots that browse the Internet and fetch new content. The more a web page attracts users, the higher the chances that this content is relevant for the rest of users. This use case implements therefore a form of crowd sourcing that does not need user engagement.

To this end, the mPlane architecture facilitates the deployment and the supervision of such a use case. Probes like tstat have the capability of extracting HTTP logs from network traffic. Thanks to the mPlane architecture, streaming such logs from multiple probes to one or more repositories is easy. The analysis modules of the content curation use case, plugged on the repository importers analyze HTTP logs online to infer relevant content that will be promoted to users. The reasoner orchestrates these operations through the supervisor, and is able to perform on demand analysis on the popularity of web pages at different time scales.

The code for this use case will be released on GitHub at <https://github.com/fp7mplane/demo-infra> and the guidelines on how to run it are available in both D4.4 and on the official mPlane website at <https://www.ict-mplane.eu/public/web-content-promotion-and-curation>. The page describes the steps that are need to set up the probe, the repository, the supervisor as well as the reasoner.

Because of a failure of the tstat probe, we could not test the entire setting (with mPlane components and reasoner). A command line version (through an mPlane client) was running successfully for few weeks on the Polito network prior to the failure. We will start the testing of the full setting as soon as the tstat probe is back.

Test #1		Probe functionalities	
Description	Status	Notes	
Test that the probe can extract the needed information from network traffic, i.e., URL, Referer, timestamp, anonymized user id.	Done	--	
Test #2		Repository functionalities	
Description	Status	Notes	
Test that the scalable data analysis algorithm running on the repository works correctly (to detect user-URLs and interesting URLs).	Done	--	
Test #3		Analysis module functionalities	
Description	Status	Notes	
Test that the two analysis modules can (1) extract content-URLs, and (2) update the list of URLs to promote.	Done	--	
Test #4		Reasoner functionalities	
Description	Status	Notes	
Test that the reasoner (orchestrator in our case) can instrument correctly the probes, repositories and analysis modules.	In progress	Tested manually using a shell mPlane client	
Test #5		Probe to repository communication	
Description	Status	Notes	
Test that the probe can export correctly HTTP logs towards the repository.	Done	--	
Test #6		Repository to analysis module communication	
Description	Status	Notes	
Test that content-portal analysis module can retrieve interesting-URLs from the repository.	Not Applicable	Design change	
Test #7		Analysis module to reasoner communication	
Description	Status	Notes	
Test that the reasoner can retrieve URLs to promote from the analysis modules.	--	--	
Test #8		Integration of the all components	
Description	Status	Notes	
Test that the overall system works properly, and that the promoted URLs output correctly on the presentation layer (web	In progress	delayed because of a long probe failure	

5 Software availability

Most of the software developed during the mPlane project are under continuous improvement, including hot fixes, patches, enhancements, therefore the attachments and "frozen links" in the corresponding deliverables can be considered as "static snapshots" only.

The official mPlane software site at <https://www.ict-mplane.eu/public/software> enumerates all mPlane related software deliverables, with links to the corresponding individual software pages. These individual software pages contain also the links to the software download pages on GitHub (see below) or (in some cases) to external sites.

The up-to-date stable version of the mPlane softwares referenced in this deliverable are always available on GitHub under <https://github.com/fp7mplane>. The product pages should contain the installation and configuration notes as well.

References

- [1] A. Rufini, M. Mellia, E. Tego, and F. Matera. Multilevel bandwidth measurements and capacity exploitation in gigabit passive optical networks. *IET Communications*, 8:8, 11/2014 2014.
- [2] E. T. S. I. E. E. . . -. V1.2.1). *Speech and multimedia Transmission Quality (STQ); QoS and network performance metrics and measurement methods; Part 4: Indicators for supervision of Multiplay services*, May 2014.
- [3] A. Valenti, A. Rufini, S. Pompei, F. Matera, S. Di Bartolo, C. Da Ponte, D. Del Buono, and G. Beleffi. Qoe and qos comparison in an anycast digital television platform operating on passive optical network. In *Telecommunications Network Strategy and Planning Symposium (NETWORKS), 2012 XVth International*, pages 1--6, Oct 2012.