mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Deployment Experiences

Author(s):	NETVISOR	A. Bakay B. Szabo, G. Rozsa, L. Nemeth
	POLITO	S. Traverso
	FUB	E.Tego, F. Matera, A. Valenti
	SSB	S. Pentassuglia, G. De Rosa
	TI (ed)	F. Invernizzi
	ALBLF	Z. Ben Houidi
	EURECOM	M. Milanesio
	ENST	J. Auge, D. Cicalese, D. Rossi, D. Zeaiter Joubmlatt
	NEC	M. Dusi, S. Nikitaki
	TID	I. Leontiadis, M. Varvello, L. Baltrunas
	FTW	A. D'Alconzo, P. Casas, A. Bär
	FHA	M. Faath, R. Winter
	ULG	Korian Edeline, Benoit Donnet
	FW	M. Scarpino, E. Kahveci, A. Sannino, E. Kowallik

Document Number:	D5.3
Revision:	1.0
Revision Date:	31 March 2015
Deliverable Type:	RTD
Due Date of Delivery:	31 March 2015
Actual Date of Delivery:	31 March 2015
Nature of the Deliverable:	(R)erport
Dissemination Level:	Public

Abstract:
This document describes the requirements of the integrated mPlane prototype and test beds and it reports on the status of the deployment at the time of this writing.
In particular, details about the design and implementation of the target deployment scenario are described along with the progress of the integration activities.
Details about the system architecture, element functions and interactions, use cases description and integration plans can be found in other mPlane deliverables available at the official mPlane website (`http://www.ict-mplane.eu/`).
In contrast to this deliverable, D5.1 contained a report about the project's data collection track record and D5.2 reported on the mPlance 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.

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

SEVENTH FRAMEWORK PROGRAMME

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

Document change record

Version	Date	Author(s)	Description
0.1	2 March 2015	F. Invernizzi (TI)	initial draft
0.2	27 March 2015	F. Invernizzi (TI) and All	consolidated draft
0.3	31 March 2015	R. Winter (FHA)	review
0.4	8 April 2015	S. Traverso (POLITO)	review
0.5	10 April 2015	M. Milanesio (EURECOM)	final review

Executive Summary

This document reports the first deployments experiences and the status of the mPlane prototype at the time of delivery of the document. In the first part requirements are described in terms of network, software and specific use cases requirements, then the status of the deployment and specific experiences are reported.

1 Deployment requirements

The mPlane integrated prototype combines a number of different systems, from different developers and designed for different purposes. In order to successfully deploy this mPlane integrated prototype, a number of requirements, documented in the following subsections, need to be fulfilled. These have been logically grouped into:

- Network requirements: these refer to network infrastructure and integration details needed to have mPlane elements not only talking to each other (basic connectivity) but also to other network related aspects that are needed to place probes in realistic networking environments, in order to have meaningful test results;
- Software requirements: all details related to the software packages, libraries and operating system requirements, which are needed by specific implementations of mPlane components such as probes, supervisors or reasoners;
- Use case requirements: the mPlane work has been use case-driven from the onset of the project and some use cases can involve specific requirements not related to other categories in order to be successfully deployed;
- Data privacy and security: details related to user data privacy protection and systems security;
- Good practices: given the size and diversity of the mPlane echo system, good practices which have been proven to be important during the work are reported, as they are considered pillars of a good deployment process.

1.1 Network requirements

One of the first and most obvious requirements in order to build a distributed prototype, integrating a large number of different components, is to have a common, shared network infrastructure, able to interconnect all functional elements of the reference architecture and granting all capabilities needed to deploy the project's chosen use cases. To this end, the mPlane project has developed a network core lab, deployed at two lab facilities hosted by the two industrial project partners Telecom Italia (Torino) and Fastweb (Milano) which are directly interconnected using a dedicated fiber link. This solution leads to a well structured, controlled shared lab that is able to offer traffic generation and network impairment functionalities, along with real user traffic data and typical ISP network provider infrastructure.

The overall structure of the core network lab/integrated test plants is depicted in Figure 1, which also highlights the different functionalities provided by each test plant and shows where mPlane components are planned to be deployed.

The two test plants only represent the core of the integrated prototype. All partner lab facilities are interconnected to this shared resource by means of standard Internet connections, simply exposing the relevant interfaces. In particular all mPlane partners can interact and interconnect to functionalities exposed by this shared infrastructure by means of:

- SSH: enabled on all mPlane network and functional elements for management purposes;
- Supervisor - backend: this is the place where all mPlane elements can do mPlane protocol related activities, such as registering capabilities, specifications, etc.;
- Supervisor GUI: the mPlane supervisor can expose a simplified HTTP interface in order to present a easy-to-grasp, high-level view of the underlying mPlane activities and to enable a

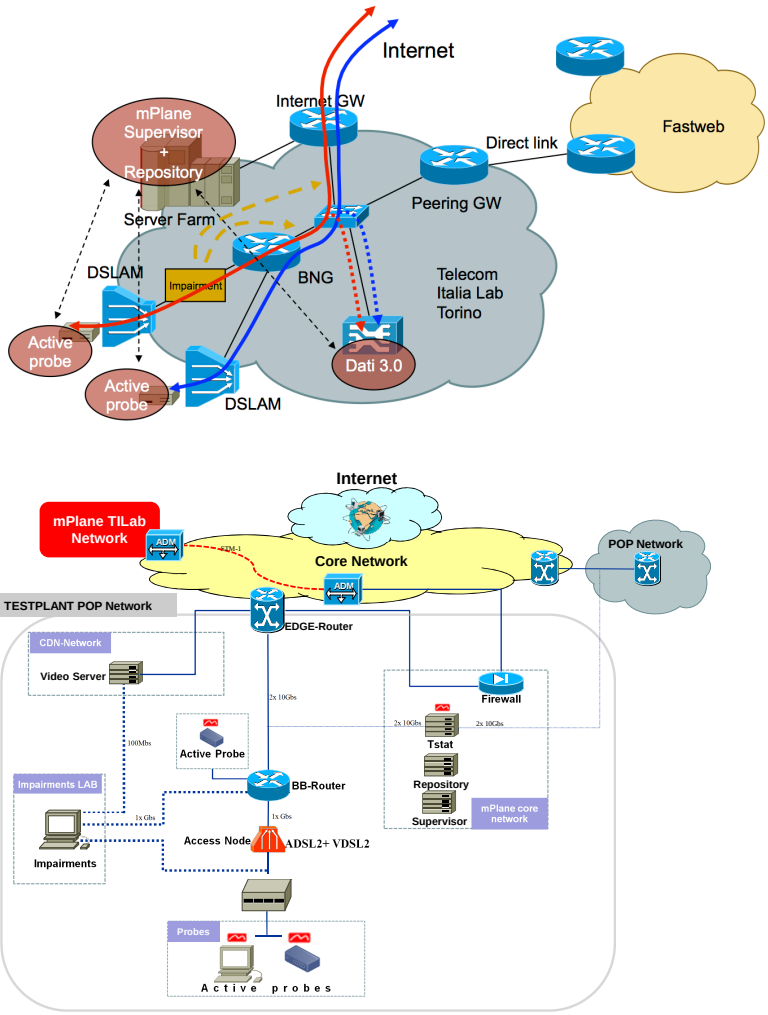


Figure 1: Telecom Italia and Fastweb interconnected test plants

user to issue measurements without requiring protocol-level understanding of mPlane;

- Reasoner GUI: some reasoner implementations can expose dedicated agent GUIs in order to present reasoning results in a specialized, reasoner-specific representation;
- Management interface: leveraging on the PM2 (`https://github.com/Unitech/pm2`) web interface, a dedicated web GUI is exposed on some mPlane instances (e.g., supervisors) for simplification of certain management operations.

The shared test plant will host an mPlane supervisor instance, available to all mPlane lab facilities by means of previously mentioned services. In order to enable the collection of useful measurements and to provide a larger set of functionalities, all partners are required to host at least one instance of all needed probes, up and running from mid 2015 to the end of the project (preferably beyond the project's lifetime). Details on IP address numbering, naming, interconnections and specific enabled service details are not reported here for security reasons.

Requirement	Description	Provided by	Notes
Internet IP reachability	IP addresses reachable from the Internet.	All sites	Public static IPs are not required. IPs can be NAT public addresses.
Network impairment	In some test scenario specific network conditions need to be emulated (packet loss, latency, ...) in a controlled manner.	Telecom Italia	
Realistic user traffic	Traffic generated by users using a realistic network scenario (e.g. ADSL access, ISP network, ISP peering, OTT) in a controlled manner.	Telecom Italia and Fastweb	
Real user traffic	Traffic generated by real user	Fastweb	Security consideration applicable
Multiple probe vantage points	Some use cases require multiple measurements taken at different network points	All partners	
Multiple administrative domains	Some use cases can take advantage of operating in different administrative domains	All	
Distributed environment	Mandatory for testing real scenarios	All	

Table 1: Network requirements.

Table 1 lists all network requirements considered mandatory in order to complete prototype preparation, testing and deployment.

1.2 Software requirements

The implementation of the mPlane integrated prototype requires a well-defined set of software libraries, operating systems and other software solutions that the various components implemented by the project are build with/designed for. As described in previous mPlane documents (see, for

Description	FQDN	Partner	Notes
Public supervisor	supervisor.ict-mplane.eu	TI	Main public supervisor
Public supervisor	supervisor2.ict-mplane.eu	FW	Secondary public supervisor
Public TI probe	probeTI.ict-mplane.eu	TI	Probe
Public PoliTO probe	probePOLITO.ict-mplane.eu	POLITO	Probe
Public FUB probe	probeFUB.ict-mplane.eu	FUB	Probe
Public SSB probe	probeSSB.ict-mplane.eu	SSB	Probe
Public ALBLF probe	probeALBLF.ict-mplane.eu	ALBLF	Probe
Public EURECOM probe	probeEURECOM.ict-mplane.eu	EURECOM	Probe
Public NEC probe	probeNEC.ict-mplane.eu	NEC	Probe
Public TID probe	probeTID.ict-mplane.eu	TID	Probe
Public NETVISOR probe	probeNETVISOR.ict-mplane.eu	NETVISOR	Probe
Public FTW probe	probeFTW.ict-mplane.eu	FTW	Probe
Public FHA probe	probeFHA.ict-mplane.eu	FHA	Probe
Public ULG probe	probeULG.ict-mplane.eu	ULG	Probe
Public ETH probe	probeETH.ict-mplane.eu	ETH	Probe
Public A-LBELL probe	probeA-LBELL.ict-mplane.eu	A-LBELL	Probe
Public FW probe	probeFW.ict-mplane.eu	FW	Probe

Table 2: Public elements details.

example, D5.2), the consortium is working on two distinct, publicly available reference implementations, based on Python and nodejs respectively. These serve the project and the community at large as a reference to test and program mPlane components against. Given the significance of these mPlane implementations, the particular software requirements for these two are listed separately in following tables.

Name	Release	Description
PyYAML	>=3.11	YAML parser and emitter
tornado	>=4.1	Web framework
urllib3	>=1.8.2	HTTP library

Table 3: python mPlane module requirements.

For deployment, OS compatibility is an important aspect to being considered, in particular for end-system-based probes, less for mPlane components such as supervisors, reasoners or repositories. During the development activities of the mPlane project, a number of particular operating systems have been employed for some of the components and are therefore considered as supported. These are described in Table 5.

In order to deploy mPlane components on a number of measuring vantage points large enough for the purposes of the project, a simple deployment solution - given the operating system list above - was adopted by all partners in order to painlessly install a good number of mPlane-ready probes. To this end, a virtual appliance containing a complete mPlane environment is being prepared, includ-

Name	Release	Description
nodejs	>= 0.6.1	This is the base nodejs environment
lodash	>=2.4.0	Javascript objects and generic util library
cron	>=1.0.4	mPlane multi measure and timing implementation
sha1	>=0.1.1	mPlane objects token generation
util	>=0.10.0	Generic nodejs util functions
sync-request	>=2.0.1	HTTP synchronous request

Table 4: nodejs mPlane library requirements.

Name	Tested version(s)	Elements tested
FreeBSD	9.0, 10.1	(nodejs, python) Supervisor, Client, Tstat, Dati proxy, Demo Reasoner
Linux Ubuntu Server	10.4	(nodejs, python) Supervisor, Client, all supported probes, Demo Reasoner
OpenWRT	14.07	(python) Supervisor, Tstat
OS X	10.10.2 (Yosemite)	(nodejs) Supervisor, Client, Dati proxy, Demo Reasoner, GLIMPSE
Windows	7	(nodejs) Supervisor, Client, Dati proxy, Demo Reasoner
Android	4.4.2	Rooted devices, tested on Galaxy S2, Nexus S, Nexus 5 (GLIMPSE on non-rooted Android devices)
Linux Fedora	21	(python) Supervisor, Client, ping proxy, tstat proxy

Table 5: Supported operating systems.

ing all required probe instances and supplemental software (control and management software, certificate generation tools, etc.). All partners will be required to install at least one instance of this appliance, configuring it and keeping it up and running. It needs to be connected to one of the public supervisors available in the shared lab facility. This ensures to have a running prototype for all project participants to experiment with, ready for integrated testing, measurement collection and proof-of-concept show case.

The virtual appliance will be an Ubuntu server version 10.4 with software installed as detailed in Table 6.

Software Name	Version	Notes
mPlane python libraries	VERSION	
tstat probe	VERSION	
mPlane nodejs libraries	>=0.6.4	
mPlane nodejs base probe	>=0.0.3	Availabel measures: ping, traceroute, HTTP latency
Certificate management scripts	n.a.	Script and root certificates for generating and manage probes certificates
PM2 manager	>=0.12.8	System management
PM2-WEB	>=2.1.3	WEB management GUI
GLIMPSE	>=2.4.0 from GLIMPSE-mplane branch	Exposed capabilities available at `http://www.measure-it.net/static/capabilities`

Table 6: Virtual appliance installed software.

1.3 Use case specific requirements

This paragraph details specific requirements needed by some use cases. Where a requirement is a specific probe, for sake of simplicity, that probe requirements are described. Where not listed, it is intended that the use case applies for above described requirements.

1.3.1 mSLAcert requirements

mSLAcert is an active probe and its main task is to verify the SLA between client and ISP. In particular when it runs either on a PC connected to the home modem (all the other devices are disconnetted from the modem) or on a 3G-4G device mSLAcert measures the line (or channel) capacity, the current throughput and RTT. In this form mSLAcert already is able to give some information regarding some problems on the line and on QoE troubles (i.e. too high RTT caused by network congestion). To run mSLAcert the following software are required:

- Python version >=3
- Yalm
- Tornado
- Iperf
- Running all under linux (tested on Ubuntu 14.04)

1.3.2 Firelog requirements

Firelog can be installed on any Linux PC (e.g., Ubuntu/Debian): the code is publicly available on Github[1]. The probe comes as a standalone Python (>= 3.0) application with the following dependencies.

[1] https://github.com/marcomilanesio/qoe-headless-probe

- Python version >=3 (standard libraries numpy, urllib, sqlite3, etc)
- Apache Flume > 1.4[2]
- phantomJS > 1.9.7[3]
- Tstat 2.4 modified[4]
- libpcap 0.8 (system dependent, on all apt repositories) for compiling Tstat
- root access (for compiling scripts and Tstat)

In order to run the probe, the user has to:

1. clone the repository
2. download and unpack the required software
3. compile **as root** the scripts in src/script folder, set the 4755 mask to the executable
4. *configure*, *compile* and *make* tstat (**do not install**)
5. change the probe.conf file accordingly
6. execute the *main.sh* script

The probe works even if no repository is set up. The results are stored in form of a local sqlite3 database and in a JSON file in the session_bkp folder.

1.3.3 Mobile Probe requirements.

The mobile probe requires Android version 4.4.2 or higher. It also requires a rooted phone (as it captures data from the network interfaces). To (optionally) run a demo on Youtube videos the official Youtube app is required. It has been tested on Cyanogen 11 and on Galaxy S2, Nexus S, and Nexus 5 devices.

For the router probe (the other end of the mobile probe), a router running openWRT is required. It has been successfully tested on an Ngear WNDR-3800 router running the Barrier-Breaker version (but any latest version should work).

For the reasoner and repository, a Linux distribution with the latest version of Apache Server, MongoDB, Oracle Java and Python are required. Also, the Weka 3.7.11 (or later) library is required.

1.3.4 GLIMPSE requirements

GLIMPSE can be installed on Debian-based systems via apt from the GLIMPSE repositories, which will take care of dependencies. On Arch-Linux GLIMPSE can be installed via the Arch User Repository which also takes care of the dependencies. The GLIMPSE client code is publicly available from Github[5]. Compiling the code requires a few non-standard libraries, which need to be pre-installed on the system, which are detailed below:

- For all operating systems: QT 5 (>=5.4) and dependencies of QT, qtsysteminfo, QtCreator recommended
- For Windows: winpcap (>= 4.1.3)
- For Linux: libwnck22 (>= 2.30.0-3)
- For Mac OS X: xcode

[2] https://flume.apache.org/download.html
[3] http://phantomjs.org/release-1.7.html
[4] http://firelog.eurecom.fr/mplane/software/eur-tstat-2.4.tar.gz
[5] `http://www.github.com/HSAnet/glimpse_client`

GLIMPSE probes do not require root access.

1.3.5 DBStream requirements

To run DBStream the libraries listed in Table7 are needed. Furthermore, `python-matplotlib`, `weka`, `golang` are needed in support of the ``Anomaly detection and root cause analysis in large-scale networks'' use-case. Yet for this use-case, DBStream should be able to import external data coming from geo-localization services such as MaxMind[6] and IP address analysis services such as Team Cymru Community Services[7].

In the integrated prototype DBStream runs on a dedicated machine at PoliTo premises. Namely, a single server equipped with 32 GB of RAM, one XEON E5 2640, supporting up to 12 threads (6 cores with hyper-threading), running at 2.5 GHz, and 4x 2TB disks (7200 RPM) running RAID 10.

Name	**Version**
postgresql	≥ 9.3
postgresql-client	≥ 9.3
postgresql-contrib	≥ 9.3
postgresql-plperl	≥ 9.3
postgresql-plpython	≥ 9.3
postgresql-plpython3	≥ 9.3
postgresql-matplotlib	≥ 9.3

Table 7: DBSTREAM libraries requirements.

1.3.6 Passive Content Curation Requirements

The media curation use case requires the following software to be installed. On the probe side:

- Tstat (TNG tstat-3.0, Traubi flavor).
- Python 3.2 or higher.

Above software setup has been tested on Ubuntu 10.04.

On the machine hosting the repository and the reasoner:

- MySQL 14.14 (version 5.5.41 or higher).
- Python 3.2 or higher installing MySQLdb and Urllib2 modules.
- Apache 2.4.7 or higher (to improve the website performance this has to be configured to work in ``worker'' mode, which requires fast-cgi and ph5-fpm packages to be installed).
- PHP 5.5.9 or higher.

Above software setup has been tested on Ubuntu 14.04.

[6] `https://www.maxmind.com`
[7] `https://www.team-cymru.org/`

1.3.7 Content Popularity Estimation Requirements

The content popularity estimation use case requires the following software to be installed. On the probe side:

- Tstat (TNG tstat-3.0, Traubi flavor).
- Python 3.2 or higher.
- mPlane Authorized Transfer via HTTP (MATH) developed by FTW (probe side).

On Debian Tstat can be installed via apt which will take care of resolving missing dependencies.

On the machine hosting the repository and the reasoner:

- MongoDB 3.0 or higher.
- mPlane Authorized Transfer via HTTP (MATH) developed by FTW (repository side).
- Python 3.2 or higher installing Numpy module.
- Apache 2.4.7 or higher.

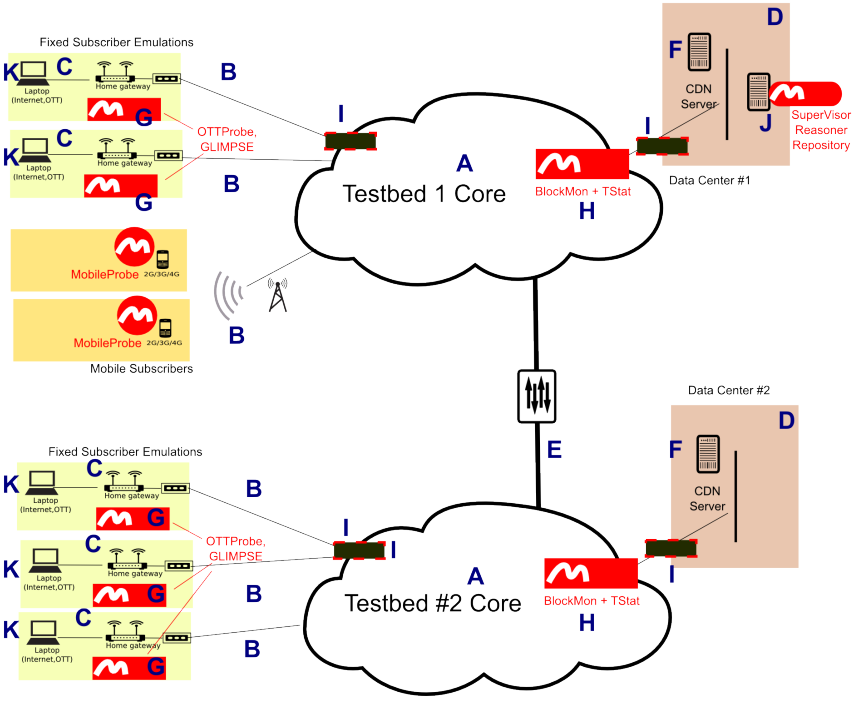
Above software setup has been tested on Ubuntu 14.04.

1.3.8 Multimedia Content Delivery Requirements

1.3.8.1 General Description

The UC demonstration emulates a comprehensive OTT content distribution network with multiple CDN servers as alternative content sources, and with multiple consumers connected through different mobile and fixed access technologies.

Passive and active probes throughout the network provide for service quality measurements. They are all controlled and used by a single Supervisor and Reasoner, which acts as a high-level Quality Assurance Center.



1.3.8.2 Infrastructure requirements for integration and demonstration

The demonstration of MMCD UC requires an extensive network of probes, machines and network links, detailed in Table 8.

In the integration phase most components are provided in NETvisor premises (possibly in smaller quantities and in some emulated form), with the exception of the Supervisor machine, which will be a publicly available server in the TI testbed.

Category	Name	ID in figure	Description	Required/recommended number of units
Network Sections	``Core network''	A	Connects the customer access lines to the datacenter, and to the external networks	1 (per testbed)
	Subscriber access lines	B	Emulated subscriber connections possibly using different technologies: xDSL, fiber, mobile, etc.	2 / 6 (per testbed)
	Subscriber home networks	C	Typical residential networks with 4-8 1GB (or 100 Mbyte) switched Ethernet ports, protected by a residential NAT/Firewall	For most of the access lines (except for 1-2 special mobile AP-s with a single mobile device only).
	Data center	D	Host virtual or physical server(s), e.g. CDN servers, Supervisor, and Probes	1 (per testbed)
	Inter-provider peering connection	E	Connects the two testbeds	1
Devices	CDN servers	F	Deployed in DC, configured to stream OTT services to clients. Low end servers, or equivalent virtual machines	1-2 (per testbed)
	Active probes (MiniProbes)	G	Running active tests at various customer networks, and also in the core and the data center	4-6 (per testbed) provided by NETvisor
	Passive probe	H	A server (possibly with 10G ports), which receives all traffic entering/exiting the DC. Will run TSTAT /Blockmon	1 (per testbed)
	Impairment emulation	I	A device to introduce impairments in some lines	1-2 (per testbed)
	Supervisor	J	A low-end server (or equivalent VM), which hosts the Supervisor, and our UC-specific repository and reasoner	1 (either at TI or at FW)
	Customer computers	K	A laptop or a mobile device (e.g. Android tablet or smartphone). Laptops have some desktop OS (Windows, Linux or IOS installed)	One per each subscriber network
Software (not provided by mPlane)	CDN Server	F	Apache with OTT module (installed by NETvisor)	One per CDN server

Table 8: Multimedia Content Delivery UC - Infrastructure requirements.

1.3.8.3 mPlane component requirements

This use case relies on mPlane Components detailed in Table 9.

Component	Provided by	Source	Status
Supervisor	Multiple partners (SDK branch in GitHub)	GitHub, SDK branch	Available
Probes			
OTT Probe	NETvisor	GitHub	Available
GLIMPSE	FHA	GitHub	Pending confirmation
Blockmon+Tstat	NEC, Polito (with possible NETvisor extensions)	GitHub	Pending conformation
MobileProbe	TID	Mailed binaries	Pending confirmation
Repositoriy			
OTT QoS Period Repo	NETvisor	GitHub	Release in April 2015
Supervisor plugins	(deployed on the supervisor server)		
OTT-Sepcific reasoner	NETvisor	GitHub	Release in July 2015
Supervisor GUI	NETvisor and others	GitHub	Available

Table 9: Multimedia Content Delivery UC - Component requirements.

1.4 Data privacy and security

The mPlane prototype implementation comes with two transport types - HTTP and HTTPS. The use of plain HTTP is intended to be used only for test and debugging purposes. Its use in production is obviously discouraged due to its use of plain-text, i.e., unencrypted data transmission that will expose the entire infrastructure to malicious attacks. The other reason is that unique identities are provided by X.509 certificates which are used to perform authentication and authorization checks between the components of the system.

For the purpose of the integrated prototype, all the mPlane partners are provided with a set of shell scripts that they can use to generate their own certificates, all belonging to the same PKI (and hence issued by the same CA). The CA private key needed to generate those certificates is communicated in a confidential way to the partners requesting it, in order to preserve an acceptable level of trust for the PKI. This approach is secure enough for the prototyping phase, but in a production environment (by a partner or external entity choosing to make use of the open source mPlane implementations) there will be no scripts available to the partners. In production, there has to be a proper authority in charge for issuing certificates. Currently, since the prototype is still in a deployment and test phase, all the certificates belong to the same domain, meaning that all the components can communicate with each other. In case the infrastructure grows and hence it becomes more complex, more domains will be defined. As a general principle however, only the components within the same domain will be able to communicate with each other directly, while inter-domain communication will be allowed only through the respective supervisors.

Authorization is performed on the basis of the Distinguished Names extracted from the certificates. The user-role and capability-role associations are stored in a file loaded by the components at start-time. Those controls work on the principle ``everything which is not allowed is forbidden'', hence it is necessary to manually add every new capability of an identity to the configuration file in order to make things visible to other components.

The protection of the personal data gathered by each probe must take place in the probe itself: all the data communicated by the probe to other components must be in aggregated form, or, alternatively, sensible information must be anonymized or pseudonymized before transmission.

2 Deployment status

This paragraphs report the status of deployment and implementation at the date of the publication of the document (April 2015), with respect to the above described requirements.

2.1 Network requirements deployment status

Following table reports the status of deployment of network requirements.

Requirement	Provided by	Deployment status
Internet IP reachability	All sites	FQDNs registered
Network impairment	Telecom Italia	Ready
Realistic user traffic	Telecom Italia and Fastweb	Ready
Real user traffic	Fastweb	Ready - Security considerations applicable
Multiple probe vantage points	All partners	Supervisors and template probes ready and online. Partners working on installing specific instances
Multiple administrative domains	All	All partners working on installing specific probe instances
Distributed environment	All	All partners working on installing specific probe instances

Table 10: Network requirements.

The interconnection between FASTWEB and TI-Lab test plants has been established by a STM-1 link (155Mb). The link and routing would enable components on the TI-Lab network to exploit the FASTWEB mPlane core network features. TI-Lab mPlane network is configured as a backend network on firewall and accessibility tests have been performed positively.

The installed components are being listed in details in D6.1. Among those elements, network configuration of the following components have been deployed and tested successfully:

- bk501mpl (NAS)
- as501mpl (Supervisor)
- db501mpl (DBStream)
- am501mpl (Mini probe)

Referred to the demo architecture presented in D6.1 the following activities have been completed:

- Access rules between internet and backend-frontend networks have been configured and tested on the firewall
- Two residential access lines (ADSL2+ and VDSL2) have been installed within FW test plant
- An internal CDN network has been set up with a fiber optic terminated router connected to Internet with a public IP address

- Two splitters which will enable Tstat probes to intercept real and generated traffic are being placed within demo architecture. Traffic switching activity is being planned internally and is in progress

2.2 Software requirements deployment status

This chapter reports the activities lead by Fastweb in testing the produced software, limits and bugs found and corrective actions taken.

2.2.1 Tested software

2.2.1.1 Tstat

FW started deploying and testing Tstat at the end of the first year of mPlane: version 2.3 was additionally installed on some of FW's pre-mPlane active probes.

A similar hybrid approach has been deployed by FW on a server, using version 2.4 of Tstat with success. This resulted in the paper written together with POLITO "Exploiting Hybrid Measurements for Network Troubleshooting". It has been accepted for the 16th International Telecommunications Network Strategy and Planning Symposium (Networks 2014).

For the mPlane demo a high performance architecture is needed, as 10 GBit/s fiber optical network links are planned to be analyzed by Tstat. At the end of the fourth mPlane quarter FW started testing a prototype developed by POLITO, using the libDNA/PF_RING software together with a dedicated Tstat branch. FW used a traffic generator in order to create UDP and TCP flows, and HTTP connections. But a high packet loss was experienced on the prototype side, for the 100 MBit/s test series and with higher transmission rates. Additionally, the amount of TCP flows seen and logged by Tstat was lower than those sent and correctly received by the traffic generator. Furthermore, some test results of the deployment in POLITO were different compared to those described above. To address this last point, FW made a detailed proposal of a reproducible and automatic deployment method, based on binary distribution (like Debian packages for example). The feedback of POLITO was positive.

At the end of the second year of mPlane FW has started to deploy and test the new approach of POLITO. It is based on DPDK, developed by Intel and released as Free Software, together with a dedicated Tstat branch. FW provided a patch for the script configure_machine.sh to fix the command for setting the number of Hugepages (nr_hugepages) in the virtual file system of the Linux kernel (under /sys). POLITO confirmed both, the issue and the solution. The results have been always positive for test from 3 minutes up to 64 hours, and up to 10 GBit/s TCP traffic (FTP and HTTP):

`Packet loss in an order of` 10^{-5}
`TCP flows not recognized by Tstat as completed in an order of` 10^{-3}

The aforementioned reproducible and automatic deployment method has been implemented by FW for this Tstat DPDK approach.

Finally, FW tested successfully the different types of anonymization to be applied for the logs written by Tstat, to respect Privacy Law and FW's policies.

2.2.1.2 Tstat proxy

FW tested successfully the mPlane Proxy for Tstat. This is the mean of interaction between the Tstat probe and the mPlane Protocol. The test was done by executing in the mPlane Client the command to increase the log details of Tstat for 5 minutes. As expected, this resulted in a change of Tstat's configuration file runtime.conf, done properly by the Proxy. Therefore Tstat increased the verbosity of the logged information, for 5 minutes as requested.

2.2.1.3 Firelog

FW tested Firelog and reported a blocking bug to EURECOM. They fixed it in this later version: `https://github.com/marcomilanesio/qoe-headless-probe.git`

In order to make the install script setup.sh work better on different GNU/Linux distributions and other operating systems, FW provided the following patches to EURECOM:

Patch 1: adds support for ifconfig not in the $PATH of a normal user (default on Debian)
Patch 2: avoids compiling as root
Patches 3a and 3b: add support for all POSIX shells, avoid dependency on bash
Patch 4: adds support also for su, avoids dependency on sudo (not default on Debian)
Patch 5: adds support for additional network interface names (used by default on CentOS 7)

Furthermore, a prove of concept for a patch to avoid dangling Tstat processes has been given. FW provided also an example for the usage of the command trap.

2.2.1.4 mSLAcert

FW analyzed the version of September 2014 and tested the later 1.0.1 release, reporting bugs and suggesting improvements. In particular, FW recommended to avoid the dependency on a complete graphical Desktop Environment, as it is not really necessary. This has been implemented by FUB in the 2.0.1 release. Regarding the need of an external iperf instance, FUB documented their solution for mPlane compliance in D6.1.

2.2.1.5 GLIMPSE

FW tested an early version of GLIMPSE, specifically the Debian package of the console client created by FHA: glimpse-console_0.1alpha1-74_amd64.deb

FW reported that some unnecessary dependencies were required to be installed, and FHA confirmed this as an issue. They were already working to avoid the installation of the Graphical User Interface (GUI) parts from the cross-platform application framework QT for the console- version.

During the start up of the probe, the error "Username to[o] long" occurred. This bug has been discovered because of a long FW email address used for registering.

During the mPlane Coding Session at Turin (January 19th-21st, 2015) FW provided a technical reference to FHA on how to implement support for different init systems in the scripts of their recent Debian packages of GLIMPSE.

2.2.1.6 DBStream

FW tested successfully the deployment of DBStream, by importing FTW's virtual image into the free version of VirtualBox (on Debian).

2.2.1.7 mPlane Python Reference Implementation

FW did some successful tests of the development branch of the mPlane Protocol Reference Implementation which adds the security architecture including Public Key Infrastructure (PKI), Certification Authority (CA) and certificates.

FW wrote 45 test cases for the modules azn.py, tls.py and utils.py of the mPlane Python Reference Implementation. 32 of them have already been reviewed and merged by ETH. The other 13 test cases have been accepted and will be merged soon by ETH.

2.2.1.8 mPlane Node.js Reference Library

The mPlane Node.js implementation was presented by TI to FW and SSB during a meeting on October 15th, 2014. FW tried a deployment on an available notebook with Debian/GNU Linux installed, supported by TI.

But a problem with dependencies of external Node.js packages (from `http://npmjs.org/`) has been experienced. Those used to create at runtime a textual (ASCII) version from the image of the mPlane logo needed even to compile non-Node.js code which failed. TI solved the issue for all Operating Systems by using the textual version of the logo directly in the source code, avoiding all the runtime dependencies for the image conversion.

Later FW did some tests of the mPlane Node.js Reference Library. Two errors have been noticed during these tests. And two patches have been provided by FW regarding the supervisor, specifically for the configuration file supervisor.json to solve both errors.

The first error was about not found certificate, key and CA files. It was caused by an absolute and specific path existing only of the development machine, and solved by using a relative path to these files inside the source code tree.

The second error occurred because the supervisor tried to create the log file in the directory /var/log/mplane which usually does not exist. The second patch changed the destination of the log file to the root directory of the code tree ("./supervisor.log"). With both patched for supervisor.json applied, the command "node supervisor.js" did not report any error, and the Supervisor started successfully.

2.3 Use case deployment status

2.3.1 mSLAcert requirements deployment.

The 2.0.1 mSLAcert release has been tested in ISCOM-FUB LAB on different GPON accesses (30-100 Mb/s downstream; 1-10 Mb/s upstream) in different core network conditions in terms of delay, jitter and bit error rate. To fully deploy mSLAcert, the following are required:

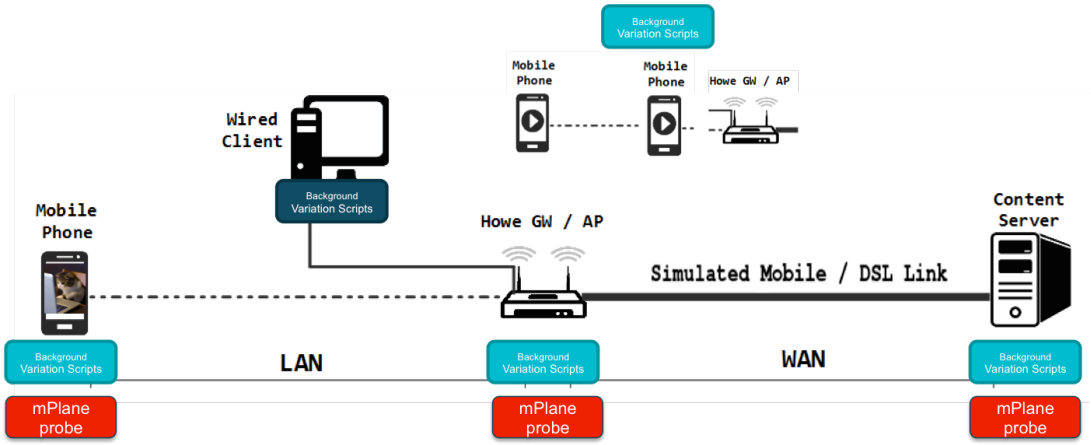


Figure 2: Mobile probe setup

- 2 Linux PC (preferred Ubuntu 14.04), on one needs to be running mSLAcer server and on the other one mSLAcert Agent.
- Both PCs need to be directly reachable, the must not be behind a NAT.
- To fully test the use case we also need the following technologies, even in part if not possible:
  1. twisted pair accesses: ADSL2+, VDSL2 (if possible also in vectoring version), G.fast;
  2. Optical fiber accesses (FTTB.FTTH): GPON and P2P;
  3. fast Ethernet links: 0.1-1 GbE;

Independently on the access technology, we need the ability to add delay, jitter and bit error rate.

2.3.2 Firelog deployment

To deploy Firelog and test the WebQoE use case the following requirements have to be meet:

- 1 cluster OpenStack (Fastweb) as repository, with HDFS as storage, and the following additional software/tools:
  - Apache Flume: a Flume sink is needed to receive data from the probes
  - Apache Spark: in order to run the analysis modules
- 1 web server for hosting the reasoner interface with access to the repository
- 1 or more Firelog probe (can be on linux PC / plugPC)
- (Optional) Ability to add delay and jitter (e.g. NETEM)

The probe is ready, and it will soon export capabilities in the mPlane reference implementation. There is a repository currently running at Eurecom premises for testing purposes. The web application for interfacing with the reasoner will be ready in the next months.

2.3.3 Mobile Probe deployment

To deploy the mobile probe the following are required:

- One or more mobile Android phones to load videos and run the mobile probe.
- One Open-WRT based wireless access point for the mobile phones to connect and to collect measurements.

- A video server to load videos and collect measurements.
- A server that hosts the reasoner and the repository (or two servers if these are separate).

For Demo purposes the following are required.

- Other wireless devices to cause interference or traffic on the wireless medium (e.g., a wireless linux-based terminal).
- Other wired devices or a traffic generator to cause traffic on the LAN/ADSL side.
- Ability to cause impairments on the internal network and to the video server.
- A visualization server to display the results and to possibly control the experiment (e.g., start/stop impairments)

An example is shown in Figure 2

2.3.4 Anomaly detection and root cause analysis in large-scale networks

For this use-case, the integration consists of the following activities:

- Integration of DBStream and Tstat: DBStream will import data from Tstat using a newly implemented External Protocol referred to as mPlane Authorized Transfer via HTTP (MATH). Data will be first transferred from Tstat probes to a NAS, serving as a proxy. On the NAS a program called `math_probe` will host Tstat logs. On the DBStream machine another program called `math_repo` will fetch the logs from the NAS in monotonically increasing time order and import them into DBStream. The exact details of the MATH protocol implementation will be described in D3.4.
  Current status: The Indirect Export module MATH is now capable to import Tstat `log_tcp_complete` in version 15 from a remote probe or NAS into DBStream. In the integration test, we were able to import four days of Tstat logs, which correspond to 200GB, in 2 hours and 10 minutes. Therefore, it is possible with the current implementation of MATH to import about 45 days of Tstat logs (or 2.3 TB) in a single day. At the time of writing MATH is fully integrated into DBStream. Further integration towards the mPlane architecture is ongoing work.
- Integration of RipeAtlas probe in mPlane: We envision two slightly different types of integration. In fact, we expect to use RipeAtlas to lunch two types of active measurements: 1) distributed pre-scheduled and continuously running active measurements (i.e., ping and Traceroutes), 2) on demand individual measurements triggered by the reasoner. The main difference is that in the first case results are exported to the repository (i.e, DBStream), whereas in the second case they are returned to the (use-case specific) reasoner via the supervisor. In the first case an importing mechanism similar to MATH will be adopted.
  Current status: The deadline for this integration is end of May (see D5.2, sec. 6.5.4). However, we are trying to anticipate the date in order to early start testing the integrated prototype.
- Integration of the analysis modules: the ADtool, the Entropy-based analysis, and the Routing anomalies analysis modules are need for the use-case objective.
  Current status: ADToll and the Entropy-based analysis modules are already integrated in DBStream and have been tested with production traffic from a nation-wide mobile network.

In accordance to the deployment plan for the use-case (ref. D5.2, §6.5.4), at the time of writing DBStream has been deployed and tested in the demo test plant on a dedicated repository consisting of a HP DL380p Gen8, 2 x Xeon E5-2620 6 core, 128 GB RAM, 24 TB Storage. For further details please refer to D6.1,§ 3.6.2.

2.3.5 Passive Content Curation Deployment

All the components involved in this use case have been deployed in the Politecnico di Torino network. The content curation service is available at the website `http://webrowse.polito.it`. To build this setup, the following hardware has been employed:

- One Linux machine running the Tstat probe enabled to monitor and log HTTP traffic. The setup must be configured so that Tstat can monitor the HTTP GET requests contained in the traffic generated by a pool of users.
- One Linux machine running the Repository, i.e., the python scripts responsible for extracting interesting URLs.
- One Linux machine running the presentation module such as, e.g., the website. This machine must install Apache, PHP5 and MySQL.

The repository receives the HTTP logs generated from Tstat using a newly implemented external protocol. Such external protocol streams the logs over the network in real-time. The current deployment is not mPlane compliant yet. However, as the all involved components already have their mPlane proxies, we plan to make the system mPlane compliant in the following weeks.

Finally, considering the Demo context, the presentation module and the repository can co-exist on the same machine.

2.3.6 Content Popularity Estimation Deployment

For this use case, the following hardware requirements have to be met:

- One Linux machine running the Tstat probe enabled to monitor and log HTTP traffic. The setup must be configured so that Tstat can monitor the HTTP GET requests contained in the traffic generated by a pool of users.
- One Linux machine running the Repository. This machine must be equipped with the external protocol mPlane Authorized Transfer via HTTP (MATH) developed by FTW, MongoDB and the python scripts responsible for extracting and sorting HTTP requests.
- One Linux machine running the presentation module such as, e.g., a website to test the content popularity prediction results.

The repository receives the HTTP logs generated by Tstat using the external protocol MATH. Such external protocol moves logs from Tstat's temporary local storage to the repository, which import them into MongoDB. Currently we are working at building the system and gluing together the various components. However, as the all involved components already have their mPlane proxies, we plan to have a fully mPlane compliant system in the following month.

For the Demo purpose the presentation module and the repository can co-exist on the same machine.

2.3.7 GLIMPSE Deployment Requirements

For the integrated prototype only the console based GLIMPSE-probe is being used and requires the following:

- A server with Ubuntu (>= 14.04) with the GLIMPSE apt package repository and the glimpse-

console app installed
- An mPlane supervisor (component-initiated)
- An mPlane repository

2.3.8 Multimedia Content Delivery Status

Table 11 details the deployment status of planned components in the development environment for this UC (NETvisor), and also in the common mPlane testbeds (of TI and FW).

Component	**Status in development environment**	**Status in Integration/Demo Testbed**
Supervisor	Operating (from develop branch)	Under installation
Probes		
OTT Probe	Operating	Operating on MiniProbes
GLIMPSE Pending	(being ported to mPlane)	Pending (being ported to mPlane)
Blockmon+Tstat	Pending (under development)	Pending (under development)
MobileProbe	Being Integrated	Pending
Repositoriy		
OTT QoS Period Repo	Pending, under development	Pending
Supervisor plugins		
OTT-Specific reasoner	Under integration	Pending
Supervisor GUI	Operating (with develop branch)	Pending

Table 11: Multimedia Content Delivery UC - Components Status.

References

[1] About jenkins.debian.net. https://jenkins.debian.net/userContent/about.html.

[2] Comparison of free and open-source software licenses. http://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses.

[3] Continuous integration. http://en.wikipedia.org/wiki/Continuous_integration.

[4] Frequently asked questions: Gnu licenses. http://www.gnu.org/licenses/gpl-faq.html.

[5] Gnu faq: Why should i put a license notice in each source file? http://www.gnu.org/licenses/gpl-faq.html#NoticeInSourceFile.

[6] Itp: tstat -- tcp statistic and analysis tool / license of ns.c? https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=323913#72.

[7] Pep 8 -- style guide for python code. https://www.python.org/dev/peps/pep-0008/.

[8] Travis ci. https://travis-ci.org.

[9] Various licenses and comments about them. http://www.gnu.org/licenses/license-list.html.

[10] J. Schäfer. Emacs python development environment *Elpy*. https://github.com/jorgenschaefer/elpy.

A Good Practices for Development and Deployment

In this section we provide some generally known and recommended good practices of interest for the project. Some references will be given, and examples of experiences within mPlane are provided.

A.1 Releases

Software should be released as early as possible, as often as possible. Releases should be preferred, rather than snapshots or tags in a version control repository.

The released source tree should

- *not* include any auto-generated file
- *nor* embed any third-party library.

For established software, developed since some time, also releases of a stable branch with security and important bug fixes only should be provided.

A.2 Licenses, Copyright

A License should be chosen for any project. There are many different Free Software Licenses available, examples of lists with additional information are [2] and [9].

An own license text should definitely not be written as it becomes very difficult to avoid problems like contradictions. The attribution of Copyright and the chosen license should be documented in each file of the source tree [5].

An example with a relationship to mPlane is the passive probe Tstat: it could not be included in Debian since 2005 even though the work of packaging was done years ago for the first time. But there are Source Code files of Tstat with none or insufficient information about the applied (Free) Software Licenses. [6] is reporting one of these files. The history of packaging efforts regarding Tstat is available on the same (whole) web page.

It is important to note that Free Software Licenses are not always compatible between each other [4]. Specifically, the GNU GPL version 3 is incompatible with version 2.

A.3 Dependencies

Packages of the Distribution/Operating System should be used, as these have been tested. Furthermore, they are supported which includes security and other updates.

An example within mPlane is the probe mSLAcert: the way to install the dependency Tornado was changed from an external source (pip3) to the official package of the GNU/Linux Distribution, python3-tornado.

Dependencies should be documented explicitly, for example not just Python 3 but Yaml for Python 3: python3-yaml

If possible the dependencies should be reduced. Within mPlane this has been done for GLIMPSE (details in 2.2.1.5), Firelog (Sec. 2.2.1.3), mSLAcert (Sec. 2.2.1.4), and the Node.js Reference Implementation (Sec. 2.2.1.8).

A.4 Different Hardware Architectures / Porting

If possible

- different Hardware Architectures

- different Kernels
- different Operating Systems

should be supported.

Tstat for example has been successfully deployed and tested within mPlane on a Hardware Mini Probe with a MIPS Processor. Additionally, Tstat works on FreeBSD.

GNU autoconf/automake is one good way to achieve multi-platform support. In fact, Tstat is using it.

A.5 Documentation

As much as possible a user/tester/developer needs to know should be documented, in particular

- the runtime dependencies of a software,
- the build dependencies of a software and for its documentation.

Manual (man) pages should also we written.

A.6 Coding Style and Automatic Syntax Checking

A Coding Style should be chosen for the Source Code. For Python there is an official one [7].

An Automatic Syntax Checker should be always used (if available for the Programming Language used).

For example for Python there is Elpy [10], an extension for the Emacs Text Editor.

A.7 Automatic Testing of Source Code

Unit Tests should be written for all Methods/Functions of the Source Code. In mPlane this has already been partially done for the Python Reference Implementation, as well as for Reference Implementation written in Node.js. Details about the Python Unit Tests are described in 2.2.1.7.

End-To-End Tests should be defined and implemented, too. In mPlane these would help the interoperability of the two Reference Implementations.

"Complete" Test Suites should be a goal.

Continuous Integration (CI) [3] should be used.

For the mPlane Reference Implementation written in Node.js, some tests are run on the Travis CI online service [8], promoted for projects hosted on GitHub.

Debian has chosen Jenkins [1] , an extensible Continuous Integration server released as Free Software.