



mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Final Implementation of Software and Libraries - Final Release

Author(s):	POLITO	Ali Safari Khatouni, Stefano Traverso, Marco Mellia, Maurizio Matteo Munafò
	FUB	Edion Tego
	SSB	Gianni De Rosa, Stefano Pentassuglia
	TI	Fabrizio Invernizzi
	EURECOM	Marco Milanese
	ENST	Pellegrino Casoria, Danilo Cicalese, Dario Rossi
	NEC	Maurizio Dusi, Saverio Niccolini
	TID	Ilias Leontiadis, Matteo Varvello, Linas Baltrunas
	NETVISOR	Balázs Szabó, László Németh, Gábor Molnár, János Bartók-Nagy, Árpád Bakay
	FHA	Michael Faath, Rolf Winter
	ULG	Benoit Donnet, Korian Edeline, Yongjun Liao
	ETH	Brian Trammel
	FW	Eike Kowallik

Document Number: D2.3
Revision: 1.0
Revision Date: 26 June 2015
Deliverable Type: RTD
Due Date of Delivery: 30 June 2015
Actual Date of Delivery: 15 July 2015
Nature of the Deliverable: (S)oftware
Dissemination Level: Public

Abstract:

This deliverable collects the software released by the mPlane Consortium at month 32. This is a software deliverable, so this document briefly describes the software by collecting the information that is present on the website page at the time of writing. Software and instruction on how to access it must be accessed from <http://www.ict-mplane.eu/public/software>.

Keywords: mPlane software, SDK, proxy interface, probe

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 SDK and Probes.....	7
mPlane Reference Implementation (RI) / Software Development Kit (SDK)	9
Anycast	12
Blockmon	14
DATI proxy	17
ECN-Spider	19
Fastping	22
Firelog	24
GLIMPSE	28
MobileProbe	34
mSLAcert.....	37
OTT-probe.....	44
Scamper.....	51
Tracebox	56
Tstat	59
2 Discountinued probes and name changes.....	65

Document change record

Version	Date	Author(s)	Description
0.0	14 Apr 2014	T. Szemethy (NETvisor)	initial draft
0.1	4 June 2015	J. Bartók-Nagy (NETvisor)	refreshed template
0.9	14 July 2015	J. Bartók-Nagy (NETvisor)	merged contributions
1.0	15 July 2015	J. Bartók-Nagy (NETvisor)	reviewed final release

Executive Summary

This deliverable summarizes software that the Consortium made available at month 32 since the beginning of the project. According to the "WP2 - Programmable Probes" specifications, it is a software deliverable intended for public release. It consists of an SDK for probe development and includes numerous implemented examples.

The software that is part of this deliverable is listed below:

1. mPlane Reference Implementation (RI) / Software Development Kit (SDK) (1)
2. Anycast
3. Blockmon
4. DATI proxy
5. ECN-Spider
6. Fastping
7. Firelog
8. GLIMPSE
9. MobileProbe
10. mSLAcert
11. OTT-probe
12. Scamper
13. Tracebox
14. Tstat

(1) Please note that the final version of SDK is part of WP5, we provide here version 0.9.0, used by the probes.

For each probe, a web page has been created following the same structure when possible, and an archive is also provided so that to allow people to download and to run the released software. The webpage gives detailed information of what has been developed within the mPlane project.

Each description page can be accessed from <http://www.ict-mplane.eu/public/software>.

Please note that this page serves as a software developer hub and lists some other software as well, that has been developed by mPlane partners but are **not** part of this deliverable. The list above includes software that either was entirely developed, or that received significant contributions and updates within the project and is therefore part of the 2.3 deliverable.

1 SDK and Probes

In the following, we attach the printed version of each probe's description webpage. *It should be taken as a D2.3 release time snapshot*, since the product pages are due to continuous changing.

The software included in D2.3 deliverables are:

1. mPlane Reference Implementation (RI) / Software Development Kit (SDK) (1)
2. Anycast
3. Blockmon
4. DATI proxy
5. ECN-Spider
6. Fastping
7. Firelog
8. GLIMPSE
9. MobileProbe
10. mSLAcert
11. OTT-probe
12. Scamper
13. Tracebox
14. Tstat

(1) Please note that the final version of SDK is part of WP5, we provide here version 0.9.0, used by the probes.

General description of the report pages

For each tool, a web page has been created following the same structure when possible, and an archive is also provided so that to allow people to download and to run the released software. The webpage gives detailed information of what has been developed within the mPlane project.

Each tool's page can be accessed from <http://www.ict-mplane.eu/public/software>.

Please note that this page serves as a software developer hub and lists some other software as well, that has been developed by mPlane partners but are **not** part of this deliverable.

Each page tries to follow the same document structure, so that a uniform description of the tools and software be offered. The intended purpose of the sections (if applicable and presented) are:

- **Description:** A description of the software is provided here, optionally with a 'Usage scenario' subsection. In case the software already has an existing, publicly available webpage, we may give here only a brief description and provide pointers to the original webpage(s) to avoid replicating data.

- **Description / Usage scenarios:** While previous point concentrates on the product features, this one is about the working environment, how the tool fits into the typical measurement architecture.
- **Metrics and Capabilities:** Description of the measurement metrics the probe collects, usually with the capabilities JSON files, with some explanation on the parameters and results.
- **Probe execution environment:** this section describes the requirements needed to run the software: operating system support, required languages/platforms/libraires, etc. If probe has several components (e.g. a binary and a python/nodeJS proxy), they are also described here.
- **Quick start:** It provides a simple description on how to download, compile and run the software. Links to external repositories can be used, e.g., to <http://github.com> or to a private svn, in case the software is not natively hosted in the mPlane svn.
- **Quick start / Installation of standalone probe:** This subsection is about the installation of the standalone probe (if applies). It includes installation prerequisites (hardware, software libraries, etc) and the install process, detailed enough to make it possible to reproduce the install from zero.
- **Quick start / Usage of standalone probe:** This subsection describes how to use the probe in standalone, non-mPlane mode.
- **Quick start / Integration into an MPlane environment:** This subsection describes how the probe will cooperate with other third-party components (probes/repos/supervisors/reasoners) of the mPlane ecosystem. The software can either natively support the mPlane interface, or a proxy has to be used to expose its capabilities, to exchange commands, and to retrieve results.
- **Official version:** This is the official (frozen) version of the software as part of the D2.3 deliverable.
- **Official version / New features supported by the mPlane project:** In case the software has not been entirely developed within the project, but just its functionality has been extended, the new features added during the mPlane project are to be listed here.
- **Official version / Changes since D2.2:** New new features and/or main bugfixes added to the software since D2.2. Also includes the mPlane interfacing if it had not been ready before.
- **References:** Here will be provided the links to the source, binaries and additional documentation, including dissemination materials as well.
- **References / Links to sources, binaries:** This section lists all the needed links to the proprietary binaries, sourcefiles (Python, NodeJS proxy), etc which are needed to make the software run.
- **References / Links to additional documentation:** If there is any extra documentation regarding the probe, eg. detailed install/user's guides, it is listed here.
- **References / Dissemination:** List the available product related publications, product videos on the mPlane Youtube channel, or any other dissemination material.

[Home](#) > [SOFTWARE](#) > [mPlane RI - Reference Implementation](#)

mPlane Software Development Kit (SDK) for Python 3

[View](#) [Edit](#)

Description

This module provides a mPlane Software Development Kit (SDK) for Python 3 and contains the mPlane protocol reference implementation. It is intended for the use of component and client developers to interoperate with the mPlane platform. It requires Python 3.3 or greater.

The core classes in the `mplane.model` and `mplane.scheduler` packages are documented using Sphinx; current Sphinx documentation can be read online [here](#).

The mPlane Protocol provides control and data interchange for passive and active network measurement tasks. It is built around a simple workflow in which **Capabilities** are published by **Components**, which can accept **Specifications** for measurements based on these Capabilities, and provide **Results**, either inline or via an indirect export mechanism negotiated using the protocol.

Measurement statements are fundamentally based on schemas divided into Parameters, representing information required to run a measurement or query; and Result Columns, the information produced by the measurement or query. Measurement interoperability is provided at the element level; that is, measurements containing the same Parameters and Result Columns are considered to be of the same type and therefore comparable.

The mPlane Protocol provides control and data interchange for passive and active network measurement tasks. It is built around a simple workflow in which Capabilities are published by Components, which can accept Specifications for measurements based on these Capabilities, and provide Results, either inline or via an indirect export mechanism negotiated using the protocol.

Measurement statements are fundamentally based on schemas divided into Parameters, representing information required to run a measurement or query; and Result Columns, the information produced by the measurement or query. Measurement interoperability is provided at the element level; that is, measurements containing the same Parameters and Result Columns are considered to be of the same type and therefore comparable.

Quick Start

To install from PyPI, type:

```
$ pip3 install mplane-sdk
```

To install from GitHub type:

```
$ git clone https://github.com/fp7mplane/protocol-ri
$ cd protocol-ri
$ python3 setup.py install
```

This section describes how to get started with the mPlane SDK, the included component runtime `mpcom`, the debugging client `mpcli`, and the demonstration supervisor `mpsups`. It presumes that the software is run from the root directory of a working copy of the <https://github.com/fp7mplane/protocol-ri> Git repository. The demonstration supervisor and the sample configuration files are *not* installed with the release SDK module.

Run Supervisor, Component and Client

To start the demonstration Supervisor with the included sample configuration and certificates, change to the repository directory and run:

```
scripts/mpsup --config ./conf/supervisor.conf
```

To run the Component:

```
scripts/mpcom --config ./conf/component.conf
```

At this point, the Component will automatically register its capabilities to the Supervisor. Now launch the Client:

```
mpcli --config ./conf/client.conf
```

As soon as it's launched, the Client connects to the Supervisor and retrieves the capabilities. To get a list of commands available, type `help`. The minimum sequence of commands to run a capability and retrieve results is:

1. `listcap` will show all the available capabilities.
2. `runcap <name_or_token>` runs a capability from the `listcap` list. You will be asked to insert parameter values for that capability.
3. `listmeas` shows all pending receipts and received measures.
4. `showmeas <name_or_token>` shows the measure (or pending receipt) in detail.

While executing these operations, the supervisor and the component will print some status update messages, giving information about the communications going on.

Sample Configuration Files

Sample configuration files are located in `protocol-ri/conf/` in the mPlane GitHub repository.

component.conf

[TLS] - paths to the certificate and key of the component, and to the root-ca certificate (or ca-chain) *[Roles]* - bindings between Distinguished Names (of supervisors and clients) and Roles *[Authorizations]* - for each capability, there is a list of Roles that are authorized to see that capability *[module_]* - parameters needed by specific component modules (e.g. ping, tStat, etc). If you don't need a module, remove the related section. If you add a module that needs parameters, add the corresponding section.

[component] - miscellaneous settings:

- `registry_uri`: link to the registry.json file to be used
- `workflow`: type of interaction between component and client/supervisor. Can be component-initiated or client-initiated. This must be the same both for the component and the client/supervisor, otherwise they will not be able to talk to each other.

To be properly set only if component-initiated workflow is selected:

- `client_host`: IP address of the client/supervisor to which the component must connect
- `client_port`: port number of the client/supervisor
- `registration _ path`: path to which capability registration messages will be sent (see [register capability](#))
- `specification _ path`: path from which retrieve specifications (see [retrieve specification](#))
- `result _ path`: path to which results of specifications are returned (see [return result](#))

To be properly set only if client-initiated workflow is selected:

- `listen_port`: port number on which the component starts listening for mplane messages

client.conf

[TLS] - paths to the certificate and key of the client, and to the root-ca certificate (or ca-chain) *[client]* - miscellaneous settings:

- `registry_uri`: link to the registry.json file to be used
- `workflow`: type of interaction between client and component/supervisor. Can be component-initiated or client-initiated. This must be the same both for the client and the component/supervisor, otherwise they will not be able to talk to each other.

To be properly set only if component-initiated workflow is selected:

- `listen_host`: IP address where the client starts listening for mplane messages
- `listen_port`: port number on which the client starts listening
- `registration _ path`: path to which capability registration messages will be received (see [register capability](#))
- `specification _ path`: path where specifications will be exposed to components/supervisors (see [retrieve specification](#))
- `result _ path`: path where results of specifications will be received (see [return result](#))

To be properly set only if client-initiated workflow is selected:

- `capability _ url`: path from which capabilities will be retrieved

supervisor.conf

Since the Supervisor is just a composition of component and client, its configuration file is just a union of the file described above. *[client]* section regards the configuration of the part of the supervisor facing the component, in other words its "client part" *[component]* section regards the configuration of the part of the supervisor facing the client, in other words its

"component part"

Learning More

See [doc/HOWTO.md](#) in the GitHub repository for information on getting started with the mPlane SDK for demonstration purposes.

See [doc/conf.md](#) for an introduction to the mPlane SDK configuration file format.

See [doc/client-shell.md](#) for an introduction to `mpcli` debug client shell.

See [doc/component-dev.md](#) for an introduction to developing components with the mPlane SDK and running them with the `mpcom` runtime.

See [doc/protocol-spec.md](#) for the mPlane protocol specification.

Official version

- The official version of the SDK is available at <https://github.com/fp7mplane/protocol-ri>. Anyone wishing to use the SDK for development within the project should be tracking the master branch of the project on GitHub. The 0.9.0 release (current as of D2.3) is tagged as [sdk-v0.9.0](#).
- The 0.9.0 release of the SDK is also available from the Python Package Index (PyPI) as `mplane-sdk`.



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



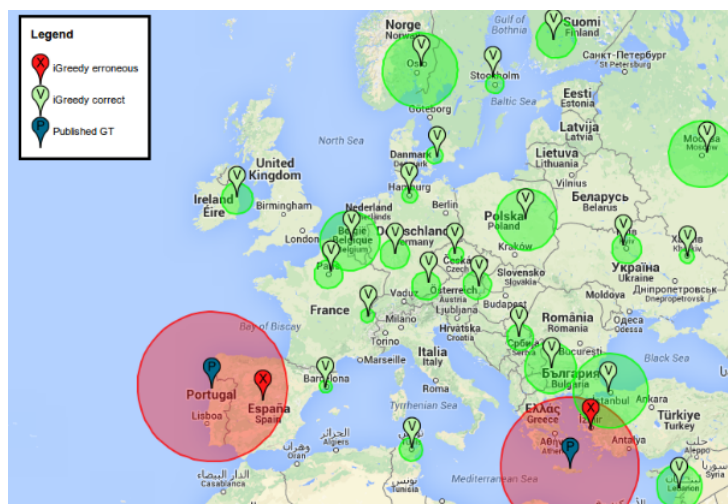
[Privacy](#)

[Home](#)[Anycast](#)[View](#)[Edit](#)

Description

Use of anycast IP addresses has increased in the last few years: once relegated to DNS root and top-level domain servers, anycast is now commonly used to assist distribution of general purpose content by CDN providers. Yet, most anycast discovery methodologies rely so far on DNS, which limits their usefulness to this particular service. This raises the need for protocol agnostic methodologies, that should additionally be as lightweight as possible in order to scale up anycast service discovery.

Our anycast discovery method allows for exhaustive and accurate enumeration and city-level geolocation of anycast replicas, with the constraints of only leveraging a handful of latency measurements from a set of known probes. The method is simple yet effective and maximizes recall by exploiting an iterative workflow to enumerate replicas (via optimization problem), while maximizing accuracy in the anycast instances geolocation despite latency noise (via classification problem).



New features supported by the mPlane project

The Anycast tool suite was entirely developed during the mPlane project. So thanks, mPlane!

Quick start

Installation and usage

For installation and usage of Anycast, see [GitHub](#).

Integration into an mPlane environment

Anycast tools comes with a native mPlane interface, exporting detection, enumeration and geolocation functions as separate capabilities. Examples are available at [GitHub](#).

References

Links to sources, binaries

Latest version is available at [GitHub](#).

Links to additional documentation

Results of a world-wide IPv4 census measurement campaign are available for browsing at <http://www.telecom-paristech.fr/~drossi/anycast>



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



Privacy

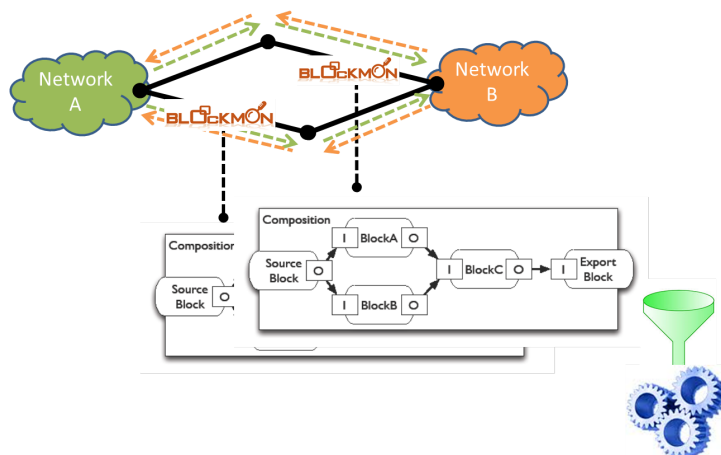

[Home](#) > [SOFTWARE](#) > [Blockmon](#)

Blockmon Node

[View](#)[Edit](#)

Description

Blockmon is a software allowing construction of flexible and high performance (rates in the 10Gb range) monitoring and data analysis nodes, where a node can be for example a hardware probe or a PC. Blockmon has started under the [EU FP7 DEMONS](#) project, and it is still under constant development under the EU FP7 mPlane mainly by [NEC](#). The official Blockmon page is available at <https://github.com/blockmon/blockmon>.



Blockmon is based around the notion of *blocks*, which are small units of processing (e.g., packet counting). Blocks are connected and communicate between each other via gates. Among other functionalities, Blockmon provides an implementation of [Tstat passive probe](#) as result of collaboration activities within the mPlane project.

A set of inter-connected blocks represents a *composition*, which defines the application to run on top of the platform. Users can express their compositions in terms of XML files.

More information on Blockmon and its performance are provided in

Simoncelli, D., M. Dusi, F. Gringoli, and S. Niccolini, [Stream-monitoring with blockmon: convergence of network measurements and data analytics platforms](#), SIGCOMM Comput. Commun. Rev., 2013

and on the [Blockmon official website](#).

Quick start

We recommend the user to refer to the latest version available on github:

```
git clone https://github.com/blockmon/blockmon.git
```

To guide the user through the installation process, an INSTALL file is provided.

The README file contains information on how to create compositions with existing blocks and how to start developing new blocks for any need.

Users can run pcapsrcctr.xml as the sample application that comes with the code. The application simply counts packets coming to a network interface and consists of the declaration of a set of blocks and their connections, all in XML format, as shown below:

```
<composition id="mysnifferctr" app_id="boh">
  <install>
    <threadpool id="sniffer_thread" num_threads="2" >
      <core number="0"/>
    </threadpool>
  </install>
</composition>
```

```

</threadpool>

<block id="sniffer" type="PcapSource" invocation="async" threadpool="sniffer_thread">
  <params>
    <source type="live" name="eth0"/>
    <!--bpf_filter expression="!tcp"/-->
  </params>
</block>

<!-- NOTE: passive blocks shouldn't have a threadpool assigned to them -->
<block id="counter" type="PktCounter" invocation="direct">
  <params> </params>
</block>

  <connection src_block="sniffer" src_gate="sniffer_out" dst_block="counter" dst_gate="in_pkt"/>
</install>
</composition>

```

Once the correct interface has been set as parameter for the capture block, you can run it by typing (note, sudo privilege might be required):

```
$ ./blockmon pcapsrcctr.xml
```

Integration into an MPlane environment

Blockmon is able to communicate with other mPlane components, thanks to the interface that makes it mPlane compliant. Please refer to the following GitHub repository for the latest version of the interface:

```
git clone https://github.com/fp7mplane/components.git COMPONENTS_DIR
```

Once Blockmon node is properly installed, get the latest version of the mPlane protocol RI at the following GitHub repository:

```
git clone https://github.com/fp7mplane/protocol-ri PROTOCOL_RI_DIR
```

and add the Blockmon components to protocol RI.

The following instructions assume you are in the **[COMPONENTS_DIR] folder**

1. Set the parameters in the file **blockmon-probe/blockmon.conf** (e.g., path to certificates, supervisor address, client port and address, and roles)
2. Set the environment variable MPLANE_RI to point to [PROTOCOL_RI_DIR]

```
$ export MPLANE_RI=[PROTOCOL_RI_DIR]
```

3. Add a softlink to a working Blockmon standalone executable (e.g., to the folder [BLOCKMON_DIR])

```
$ cd blockmon-probe
$ ln -s [BLOCKMON_DIR]/blockmon blockmon
```

4. Run Blockmon probe (you might need administrative privileges if you capture from a network interface)

```
$ python3 blockmon.py --config blockmon.conf
```

New features supported by the mPlane project

Thanks to the support of the mPlane project we extended Blockmon functionalities with the following features:

- **integration with Tstat tool.** Regarding its ability to collect information on the traffic that crosses a link, we created a block which integrates the **Tstat tool**;
- **exchanging data across nodes.** We then introduced the ability of inter-data communication across multiple Blockmon nodes, so that nodes can exchange or send information to aggregating nodes, thus allowing distributed computation;
- **bug fixes.** The platform is constantly maintained and bug fixes are pushed to the public git repository.

Official version

References

Links to sources, binaries

Blockmon node: standalone version

- May 8th, 2014 - frozen release for D2.2:
svn checkout <https://svn.ict-mplane.eu/svn/public/software/blockmon>
- For the latest release, please always refer to the github repository
git clone <https://github.com/blockmon/blockmon.git>

Blockmon node: mPlane interface

- For the latest release, please always refer to the github repository
git clone <https://github.com/fp7mplane/components.git>

Links to additional documentation

Dissemination

- Simoncelli, D., M. Dusi, F. Gringoli, and S. Niccolini, [Stream-monitoring with blockmon: convergence of network measurements and data analytics platforms](#), SIGCOMM Comput. Commun. Rev., 2013



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)



Plane

Building an Intelligent Measurement Plane for the Internet

My account

Log out

Home > SOFTWARE > DATI

DATI

View

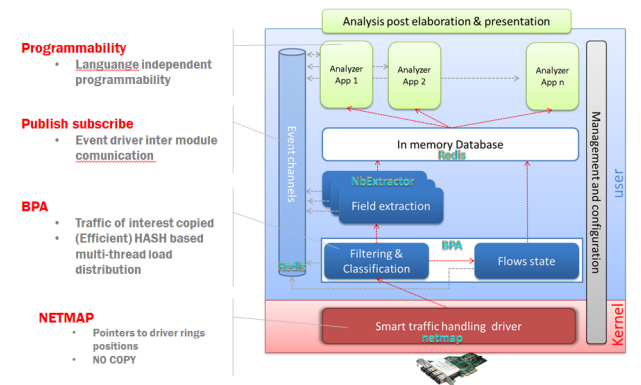
Edit

Public page *DATI* has been updated.

Description

DATI is a flexible, high performance passive monitoring platform. Build on FreeBSD, it leverages on **NetMAP** fast and safe network driver to access network devices, reaching up to 14.88 Mpps with a CPU running at less than 1 GHz. The goal of the platform is to give developers an high level view of traffic of interest without the complexity of high speed traffic capture and classification details.

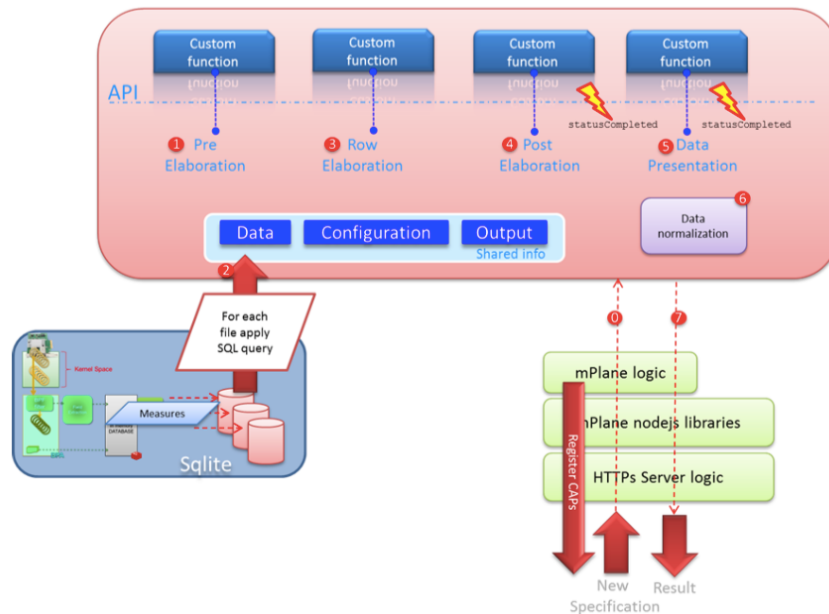
The system can recognize traffic flows from static BPF signatures and collect state information (volume, packets number and session duration) as well as application-level data based on an XML description of the protocol stack.



On top of this framework is possible to build any analysis application, called *analyzer App*, that can elaborate traffic information extracted by DATI at its own convenience. An *analyzer App* can be written in any language with the only requirement to have a **REDIS** library to interface with the in-memory database and pub/sub facility.

Integration into an mPlane environment

For the DATI probe to interoperate with the mPlane platform, a dedicated proxy has been developed leveraging on the mPlane nodes **library** written by Telecom Italia.



The previous figure shows the proxy architecture, where the intermediate Sqlite files produced by the DATI software can be programmatically elaborated to produce needed measures. The proxy API is written so that all intermediate logical functions (Pre-elaboration, ROW-elaboration, Post-elaboration and presentation) can be freely customized per-measure.

New features supported by the mPlane project

- Dedicated proxy to enable interoperation with the mPlane framework.
- Within the mPlane project some specific measures have been defined, enabling the platform to extract the following measures:
 - TCP RTT delay
 - Radius informations
 - Routing protocols informations (OSPF, BGP, LDP).

References

Links to sources, binaries

- The mPlane nodeJS reference library can be downloaded from [here](#).
- The DATI proxy software can be downloaded [here](#).

File:

 [DATI proxy architecture](#)



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



Privacy



Plane

Building an Intelligent Measurement Plane for the

[My account](#)[Log out](#)[Home](#) > [SOFTWARE](#) > [ECN-Spider](#)

ECN-Spider

[View](#)[Edit](#)

Developer

ETH Zurich; contributors University of Liege (Scamper component) and collaborating institution University of Auckland (QoF testing).

Description

ECN-Spider is an active measurement tool, built in a modular manner using mPlane components and an associated client, to test Explicit Congestion Notification (ECN) connectivity failures and readiness for ECN negotiation. ECN-Spider takes as input a list of target IP addresses to test. For each unique address, the tool simultaneously opens two connections, one with ECN negotiation enabled and one without. ECN Spider utilize Linux's system-level configuration of ECN negotiation, instead of packet injection, performing the following step for each measurement:

1. disable ECN and open a socket to the target,
2. enable ECN and open a second socket to the target, and
3. optionally perform HTTP requests via both sockets or close immediately

Then, it reports the per-address connection status. Integrated into ECN Spider is the **QoF flow meter**, which observes the traffic generated by this test and logs TCP and IP ECN flags for each connection attempt. For each test, the tool outputs the address tested and the flags observed on the first and subsequent packets, from which connectivity, ECN negotiation, and ECN signaling failure detection can be derived.

The ECN Spider component performs the actual test. Other components find target addresses for testing:

1. HTTPResolver selects IP addresses based on a list of website domains (e.g., the Alexa top million list) to test ECN readiness of webserver targets. It resolves these to at most one IPv4 and one IPv6 address per site using a given DNS server. Duplicate IP addresses are eliminated taking the highest-ranked website for each address.
2. BtDhtResolver utilizes the BitTorrent Distributed Hash Table (DHT) protocol to detect hosts that are currently available to open a TCP connection by searching for a (random) torrent that leads to forwards to potential seeds.

The tool also includes an mPlane client to coordinate the components to run either local or distributed runs of the measurements. The tool is presently being generalized to support mass measurement of path- and feature-dependent connectivity and feature functionality issues; hence the full toolchain is referred to as **pathspider**, of which ECN Spider is a single feature.

Usage Scenarios

The tool is designed either to harvest a set of IP addresses in order to test general ECN connectivity safety for a sample of the Internet (see measurement results on webserver testing at <http://ecn.ethz.ch/>) or to take ECN connectivity measurements from specific set of targets from a given probe, in order to test ECN-related connectivity failure to eliminate ECN as a potential cause of a reported problem. We note this second scenario is increasingly operationally relevant, due to the pending ECN on-by-default behavior for Apple Mac OS and iOS devices, an June 2015 announcement based in part on mPlane work with ECN safety.

Metrics and Capabilities

Given a set of target IPv4 or IPv6 addresses, ECN Spider returns connectivity with ECN negotiation attempted and without, as well as TCP and IP ECN codepoint information in order to diagnose ECN signaling issues. The core IPv4 capability is as follows:

```
{ "capability": "measure",
  "parameters": { "destination.ip4": "[*]",
    "destination.port", "[*]" },
  "results": { "source.port",
    "destination.ip4",
    "destination.port",
    "connectivity.ip",
    "ecnspider.ecnstate",
    "ecnspider.initflags.fwd",
```

```
"ecns spider.synflags.fwd",
"ecns spider.unionflags.fwd",
"ecns spider.initflags.rev",
"ecns spider.synflags.rev",
"ecns spider.unionflags.rev",
"ecns spider.ttl.rev.min" }
}
```

The ecns spider. elements are included in a custom registry inheriting from the core registry, included with the component.

Probe Execution Environment

ECN Spider runs on any reasonably recent Linux machine capable of running the Python 3 mPlane SDK. It requires QoF (<https://github.com/britram/qof>) and its prerequisites (libglib-2.0, librtrace, libfixbuf) to be installed. It requires the post-0.9.0 version of the SDK (including multiple value support) in the sdk-multival branch to be installed on client and component side. See below for installation instructions.

Quick start

First create a Python 3.4 virtual environment:

```
$ virtualenv -p python3.4 venv
$ source venv/bin/activate
```

Install tornado

```
(venv) $ pip install tornado
```

Installing the sdk-multival branch of mPlane using git and pip:

```
(venv) $ git clone https://github.com/fp7mplane/protocol-ri.git
(venv) $ cd protocol-ri
(venv) $ git checkout sdk-multival
(venv) $ cd ..
(venv) $ git clone https://github.com/britram/pathtools.git

(venv) $ pip install -v -e protocol-ri
```

And finally to install pathspider type: (note: dependencies numpy and pandas need some time to install):

```
(venv) $ pip install -v -e pathtools
```

Configuration

pathspider operates in three modes:

- service : Just run mPlane components, acting as a measurement probe.
- client : A client implementation analyzing results from multiple probes.
- standalone : A standalone implementation where the measurements and analysis are performed on the same computer.

Each operating mode has its own configuration file. Either service.conf, client.conf or standalone.conf. standalone.conf basically includes all configuration options from the client and service mode.

Client Configuration

Adjust URLs to point to your mPlane probes:

```
[probes]
nyc = http://path-nyc.corvid.ch:18888/
ams = http://path-ams.corvid.ch:18888/
sin = http://path-sin.corvid.ch:18888/
sfo = http://path-sfo.corvid.ch:18888/
lon = http://path-lon.corvid.ch:18888/

[main]
use_tracebox = false
resolver = http://path-ams.corvid.ch:18888/
```

Service Configuration

You will probably want to change interface_uri to the network interface the traffic flows.

```
[module_ecnspider]
module = pathspider.ecnspider2
worker_count = 200
connection_timeout = 4
interface_uri = ring:eth0
qof_port = 54739
enable_ipv6 = true

[module_btdhtresolver]
module = pathspider.btdhtresolver
enable_ipv6 = true

# other optional arguments:
# ip4addr = 0.0.0.0   # bind ecnspider to this IPv4 address
# ip6addr = ::       # bind ecnspider to this IPv6 address
# port4 = 9881       # bind address collector to this IPv4 address
# port6 = 9882       # bind address collector to this IPv6 address

# other optional arguments:
# ip4addr = 0.0.0.0   # bind ecnspider to this IPv4 address
# ip6addr = ::       # bind ecnspider to this IPv6 address

[module_scamper]
module = pathspider.scamper.scamper
ip4addr = 1.2.3.4
ip6addr = ::1

#[module_webresolver]
#module = pathspider.webresolver
```

Examples

To run the examples, change to the pathspider directory. (The configuration files have to be in the same directory or they have to be explicitly specified by `--config FILE`.)

```
cd pathtools/pathspider
```

Print all available options with `pathspider -h`

Running a standalone measurement using BitTorrent DHT as address source:

```
pathspider --mode standalone --resolver-btdht --count 1000
```

Official Version

ECN Spider is available from GitHub, <https://github.com/britram/pathtools>; see the quick start instructions above. The probe does not appear in D2.2, as it has been developed and integrated into mPlane since its publication, having started as a separate project at [ETH Zurich](#).

References

[ETH's ECN observatory page](#) is <http://ecn.ethz.ch/>. ECN Spider has been used to generate the results in the mPlane paper "Enabling Internet-Wide Deployment of Explicit Congestion Notification" at PAM 2015, cited in the [recent announcement](#) that Apple will enable ECN by default on the client side in developer seeds of Mac OS X and iOS.



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)

[Home](#) > [SOFTWARE](#) > [Fastping](#)

Fastping

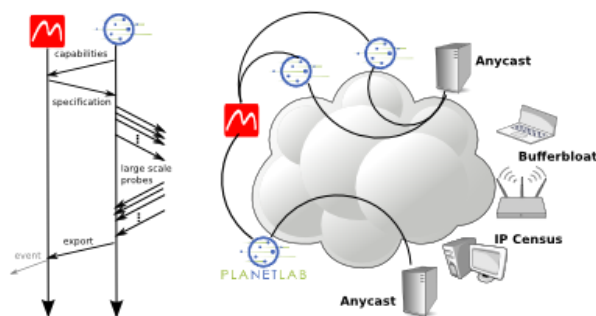
[View](#)
[Edit](#)

Description:

Fastping, developed by [ENST](#), is a fast ICMP scanner for [TopHat/TDMI](#), a dedicated measurement infrastructure running over PlanetLab. If you landed to this page following a <http://ow.ly/wPjH6> URL found within an ICMP packet, this means that you have been a target of an academic experiment with the fastping software, that this page introduces.

"Fast" in fastping means that 50k hosts can be probed in about 5 seconds (a more detailed performance analysis is reported in the [fastping documentation](#)). Scalability of a single probe is obtained in user-space (as opposite to the [zmap](#) software that requires root privileges), with a non-blocking multi-thread design (that allows to significantly exceed nMap Scripting Engine performance, but of course not as much as [zmap](#)).

Additionally, leveraging the [TopHat/TDMI](#) infrastructure, fastping couples the ability to scan a large number of hosts during small time windows, to the availability of a large number of spatially disperse probes -- up to 1000 PlanetLab/OneLab nodes, of which typically around 300 are available at any time.



These temporal and spatial scalability properties can be leveraged for instance, to understand Internet properties such as (but not limited to):

- anycast detection (when all probes target the same target during the same window, but change target over time)
- bufferbloat evaluation (when each probe tracks disjoint targets, but keep the same target continuously over time)
- Internet census (when each probe tracks disjoint targets, changing target over time)

Fastping already perform a fair amount of statistical pre-processing, providing output at different granularities. Namely, from the most coarse to the most fine-grained:

- (txt) experiment summary
- (cvs) per-probe statistics summary (e.g., CDF of relevant metrics as RTT delay, RTT variation, or TTL variation)
- (cvs) per-probe per-host statistics (e.g., quantiles of relevant metrics as RTT delay, RTT variation, or TTL variation)
- (cvs) raw measurement

The above results can be stored locally at a probe (useful for testing/local use), or uploaded to an repository via FTP (useful to centralize data collection from a PlanetLab experiment).

Quick start:

Fastping is a python script and

- can be used as a standalone shell tool (its usage is thus relatively simply explained by the manpage)
- can be used as a TopHat/TDMI component (out of scope of this page)
- can be queried as a mPlane probe (i.e., receive and parse mPlane specification)

Example of use of the tool as a standalone shell tool and as mPlane probe can be found in the software package (HOWTO.shell)

and HOWTO.mplane in the main directory respectively) as well as in the [fastping documentation](#)

New features supported by the mPlane project

Fastping was entirely developed during the mPlane project. So thanks, mPlane!

mPlane proxy interface

The mPlane Fastping interface offers the ability to specify the set of target IP address ranges.

- (simplest form) each probe can be queried individually, i.e., specifying the target for each probe in mPlane terms, and will upload the results on the specified server
- (work in progress) measurement specification can be addressed to a dedicated mPlane supervisor, handling the TopHat/TDMI probes, that will dispatch measurement specification to all Fastping probes that registered to the supervisor

mPlane custom registry

To harness the full power of fastping, a number of parameters have clearly been defined in the mPlane registry. While most of the parameters are trivial, one is worth discussing at length.

To achieve efficient operation, it is imperative for fastping to receive compact specification of target addresses ranges. However, the current mPlane registry only support a **destination.ipv4** type specifying a **single** target address -- which clearly clashes with the ability of probing at large scale.

To circumvent with this current mPlane limit, the fastmPlane implementation employs a clever trick that is both compatible with the current mPlane specifications, and that will possibly be entirely supported by future releases -- since the necessity to specify list of primitive types will be possible shared among multiple components, and thus supported by mPlane.

The fastping custom registry specify thus a **destinations.ipv4range** type whereby the **s** implies that a **plurality** of IPv4 ranges, expressed as **A.B.C.D/N** are expected. Notice that since a single address can be expressed as a /32 range, this effectively means that it is possible to practically "mix" addresses and ranges, while maintaining plurality of a single type

Official version

- Frozen release for D2.2

svn co <https://svn.ict-mplane.eu/svn/public/software/fastping-d22>

- Release for D23 available through the official mPlane GitHub
<https://github.com/fp7mplane/components/tree/master/fastping>



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)


[Home](#) > [SOFTWARE](#) > [Firelog](#)

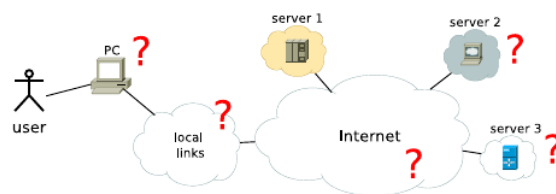
Firelog

[View](#)[Edit](#)

Description:

Firelog is a hybrid probe capable of performing passive and active measurements in the web browsing domain in order to identify a poor quality of experience (i.e., a high page load time) and to diagnose its root causes.

The official Firelog page is available at <http://firelog.eurecom.fr/mplane>.



Firelog consists of different modules: (1) an instrumented headless browser, (2) a network packet analyzer and (3) an active measurements monitor.

Such components independently capture events and metrics and store them in a local data store. An additional orchestration process accepts as input one or more "seed URLs" to inspect, uses the above modules to collect data, and finally proceed with their mapping to a unique identifier space, to build a database of correlated events and metrics that can be used as a training set for an application-specific data analysis.

Firelog is based on the PhantomJs headless browser for two reasons: (1) our interest is not Web usability but rather page downloading speed, and (2) for not imposing an overhead in the browser's activity.

We instrumented the browser for injecting in each HTTP GET message a random identifier for each object in the requested Web page, which is used to establish a link between measurements taken at different layers of the stack.

Similarly, the packet analyzer collects the transport-level measurements and associates each requested Web object with the corresponding TCP flow, by exploiting the injected identifier.

Finally, the active measurements monitor takes care of performing network-level measurements against selected destinations, corresponding to servers involved in building up the Web page under examination.

All the data are processed to diagnose the current web browsing session, providing details on the root cause for a (possible) high page load time.

Metrics and capabilities

Metrics

The workflow of the probe is composed by three distinct phases, as follows.

Phase 1. In the first phase, the probe browses a given URL. Each fetched object is associated with a unique identifier, exploited by the packet analyzer to associate the corresponding TCP flow to the object. All the browser level measurements and the passive measurements are taken in this phase.

Core metrics for Phase 1 are:

- **session start:** timestamp of the starting of the current web browsing session
- **full load time:** the page load time for the current session
- For all objects in a web page:
 - **http time:** time elapsed between a HTTP GET for an object and the 1^{st} byte of data received for that object
 - **tcp time:** TCP handshake timing to the server providing the object
 - **netw bytes:** size of the object
 - **rcv time:** time elapsed for receiving the complete object

Phase 2. In the second phase, the active measurements take place. All the collected IP addresses are processed, sending ICMP messages towards all the destinations and performing a Traceroute against the *primary* IP address (i.e., the IP resolved by the DNS). By doing this, we collect path information for all the resources contained in the browsed Web page.

Core metrics for Phase 2 are:

- For the **primary** IP address:
 - **ping**: RTT to destination
 - **traceroute**: complete path towards the destination
- For **all** the collected IP addresses (e.g., secondary servers for particular resources):
 - **ping**: RTT to destination

Phase 3. All the data relative to the same Web browsing session are packed together and stored in a CSV file and sent to the DISC repository for further analysis. A preliminary analysis of the data collected so far is returned on termination.

Core metrics (among all the others, derived from previous phases) for Phase 3 are:

- **page size**: sum of the sizes of all the objects
- number of secondary servers

Capabilities

Firelog offer the capability to run a local diagnosis algorithm towards a specific web site.

- **firelog-diagnose**: enables the diagnosis of a browsing session towards a given web site.

```
capability: measure
label : firelog-diagnose
link : /
token : 4e9b281073bd9334ca8f45704a90a1bc
when : now + 5m
parameters ( 1):
  destination.url: *
metadata ( 3):
  System_type: firelog
  System_version: 0.1
  System_ID: firelog-Proxy
results ( 1):
  firelog.diagnose
```

Probe execution environment

Firelog is written in Python3, and it is coupled with a Javascript script for exploiting the headless browser toolkit and two C scripts for managing the embedded Tstat version.

It has been compiled and tested under Unix/Linux environments, specifically under:

- Ubuntu 14.10 64bit
- LinuxMint 16 Petra 64bit.

The Python3 environment is needed also to run the proxy derived from the reference implementation RI/SDK.

Installation of standalone probe

We recommend the user to refer to the latest version available on GitHub:

```
git clone https://github.com/marcomilanesio/qoe-headless-probe.git
```

Usage of standalone probe

The README file contains information on how to set up the environment for running the standalone version of the probe.

Requirements: All the additional software can be retrieved from <http://firelog.eurecom.fr/mplane/software>:

- phantomJS headless browser toolkit
- apache flume > 1.5.2
- custom Tstat 2.4

First, configure and compile Tstat on your machine, following the instruction at <http://tstat.tlc.polito.it/index.shtml>

```
$ cd eur-tstat-2.4
$ ./autogen.sh
$ ./configure.sh
```

```
$ make
$ cd ..
```

DO NOT run "make install".

You need to compile with sudo privileges the C programs in the script/ folder:

```
$ cd script
$ sudo gcc -o start.out start.c
$ sudo gcc -o stop.out stop.c
$ sudo chmod 4755 *.out
```

You need a set up a configuration file for Tstat, specifying the interface to sniff (see <http://tstat.tlc.polito.it/index.shtml> for details).

Then, you have to modify accordingly the parameters in the file **conf/firelog.conf**.

Once all the steps are done, you run it by typing:

```
$ ./phantomprobe.py -h
```

Integration into an mPlane environment

Firelog is able to communicate with other mPlane components, thanks to the interface that makes it mPlane compliant. Please refer to the following GitHub repository for the latest version of the interface:

```
git clone https://github.com/fp7mplane/components.git COMPONENTS_DIR
```

Once Firelog is properly installed, get the latest version of the mPlane protocol RI at the following GitHub repository:

```
git clone https://github.com/fp7mplane/protocol-ri PROTOCOL_RI_DIR
```

and add the Firelog components to protocol RI.

The following instructions assume you are in the [COMPONENTS_DIR] folder.

```
$ cd firelog
$ ./install.sh [--flume]
```

The **install.sh** will take care of downloading and installing the additional software needed, and also to configure Tstat. The option **--flume** if set, will take care of installing also a flume agent for the Firelog probe, to be used with a HDFS DISC repository.

- Check the parameters in the file **conf/firelog.conf** (e.g., directory of the installed Tstat and PhantomJS, username and so on)
- Check the parameters in the file **conf/firelog-tstat.conf** (e.g., source IP address/mask to sniff from)
- Set the parameters in the file **conf/firelog-flume.conf** (e.g., flume sink address)

Finally, set the environment variable MPLANE_RI to point to [PROTOCOL_RI_DIR]:

```
$ export MPLANE_RI=[PROTOCOL_RI_DIR]
```

New features supported by the mPlane project

Firelog has been completely developed within the mPlane project, and it's still under constant development by Eurecom.

Changes since D2.2

- full support to export to HDFS (csv or json)
- bug fixing: The platform is constantly maintained and bug fixes are pushed to the public git repository;
- mPlane proxy. Python based proxy for the mPlane RI SDK to enable its usage in mPlane Reference Implementation's distributed measurement

Links to sources, binaries

- official mPlane website: <https://www.ict-mplane.eu/public/firelog>
- standalone version: <https://github.com/marcomilanesio/qoe-headless-probe.git>
- interface to mPlane components: <https://github.com/fp7mplane/components.git>

- The official Firelog web page: <http://firelog.eurecom.fr/mplane>.

Official version(s)

- May 15th, 2014: frozen release for D2.2
 - Firelog prototype presenting all the functionalities [[tarball](#)] (no mPlane proxy, but standalone)
 - Firelog mPlane interface [[github](#)]
- July 10, 2015: for D2.3
 - standalone version: <https://github.com/marcomilanesio/qoe-headless-probe.git>
 - interface to mPlane components: <https://github.com/fp7mplane/components.git>

File:

 [901firelog.tar.gz](#)



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)

Home > SOFTWARE > GLIMPSE

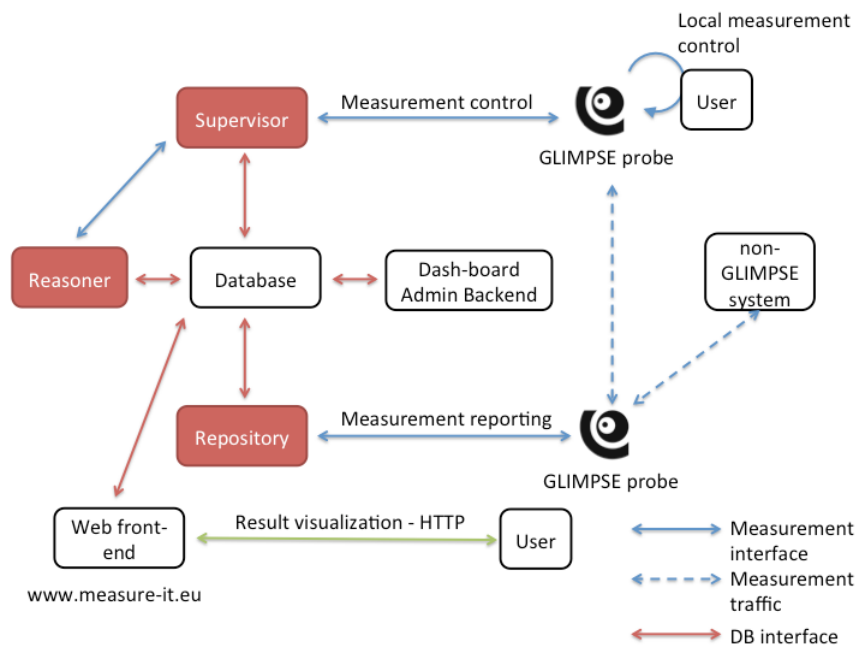
GLIMPSE

View

Edit

Description

GLIMPSE is an active end host-based network measurement tool. It is written in C++ with Qt and is cross-platform by design. The main focus of GLIMPSE is troubleshooting.



The key features are

- Plugin system for measurements
- On-demand troubleshooting if the user experiences problems or bad performance
- Scheduled measurements and measurement campaigns
- User-defined measurements
- Measurements against server hosts as well as other end-users

Usage scenarios

There are two different usage scenarios for the GLIMPSE probe at the moment:

1. Server systems where the GLIMPSE probe runs continuously as a service in the background (**console version**). These deployments provide fixed measurement points to execute measurements from but also to provide measurement end points for other probes (e.g. BTC measurements).
2. End user systems at home and on mobile devices (**GUI version**). We expect these devices to execute measurements for troubleshooting. The user can trigger the execution of measurements once he/she realizes something is wrong with his/her network connection. The GLIMPSE backend and/or the probe have to perform reasoning tasks to locate the problem.

Metrics and Capabilities

Metrics

The measurement tools included in GLIMPSE are:

- **Ping**: Measures the round-trip-time from the probe to a given destination (by IPv4, IPv6, or hostname). There are multiple parameters available for this measurement to modify the ping (like ports, interval-time, or timeout). The ping-type parameter determines which protocol is used for the measurement: "UDP", "TCP", or "System Ping (ICMP)". The later uses the systems ping-tool and parses the output (screen scraping).

- **Traceroute**: Measures the path from the probe to a given destination (by IPv4, IPv6, or hostname) and returns also the round-trip-time to each hop on the path. This measurement uses the previously described ping measurement to retrieve this information. Note that TCP pings are not available for this measurement on Linux due to operating system limitations. There are multiple parameters available for this measurement similar to the ping options.
- **HTTP download speed**: Measures the speed of a file download from an HTTP-server. It is possible to execute this measurement in multiple parallel threads and get results for each thread separately.
- **Bulk transfer capacity (BTC)**: Measures the bulk transfer capacity from this probe to another probe using either TCP or UDP. The used port and initial data size can be set through options. This measurement needs another GLIMPSE-probe running at the destination.
- **DNS lookup**: This measurement returns all available DNS records for a given hostname. The used DNS server can be set by a parameter.
- **Reverse DNS lookup**: Determines the hostname for a given IP by doing a reverse DNS lookup.
- **Trains of packet pairs**: Determines the sending and receiving speed for the probe by using trains of packet pairs. This measurement is considered experimental.
- **UPnP discovery**: This measurement returns all devices found by a UPnP discovery using the MiniUPnP library. The result format is a dump of the raw data returned by this library.

Capabilities

The JSON capabilities for GLIMPSE can be found on the following pages. Note that no temporal scope is given in this listing, the following are supported:

- singleton measurements: 'now', '2015-07-02 12:00:00'
- finite repeated measurements with an (implicit) inner scope of now: 'repeat now + 60s / 10s', 'repeat 2015-07-02 12:00:00 ... 2015-07-02 12:10:00 / 1m'

The GLIMPSE-registry defines the parameter and result columns needed which are not specified by the mPlane-core-registry and can be found at http://www.measure-it.eu/static/glimpse_registry.json. Details for the parameters can be obtained from there and from the mPlane core registry.

```
{
  "capability": "measure",
  "label": "glimpse-ping",
  "parameters": {
    "destination.port": "*",
    "destination.url": "*",
    "glimpse.ping.interval.ms": "*",
    "glimpse.ping.payload": "*",
    "glimpse.ping.timeout.ms": "*",
    "glimpse.ping.type": "*",
    "packets.ip": "*",
    "source.port": "*",
    "ttl": "*"
  },
  "registry": "https://www.measure-it.net/static/glimpse_registry.json",
  "results": [
    "rtt.ms.min",
    "rtt.ms.max",
    "rtt.ms.avg",
    "rtt.ms.stdev",
    "packets.ip.sent",
    "packets.ip.received"
  ],
  "version": 0,
},
{
  "capability": "measure",
  "label": "glimpse-httpdownload",
  "parameters": {
    "destination.url": "*",
    "glimpse.http.thread.count": "*"
  },
  "registry": "https://www.measure-it.net/static/glimpse_registry.json",
  "results": [
    "glimpse.http.bandwidth.imputed.bps.avg",
    "glimpse.http.thread.count",
    "untainted"
  ]
}
```

```

    ],
    "version": 0,
  },
  {
    "capability": "measure",
    "label": "glimpse-dnslookup",
    "parameters": {
      "destination.url": "*"
    },
  },
  "registry": "https://www.measure-it.net/static/glimpse_registry.json",
  "results": [
    "glimpse.dns.record-name",
    "ttl",
    "glimpse.dns.record-value",
    "glimpse.dns.record-type"
  ],
  "version": 0,
},
{
  "capability": "measure",
  "label": "glimpse-traceroute",
  "parameters": {
    "destination.port": "*",
    "destination.url": "*",
    "glimpse.ping.interval.ms": "*",
    "glimpse.ping.payload": "*",
    "glimpse.ping.timeout.ms": "*",
    "packets.ip": "*",
    "source.port": "*"
  },
  "registry": "https://www.measure-it.net/static/glimpse_registry.json",
  "results": [
    "glimpse.traceroute.start-time",
    "glimpse.traceroute.hop-ip",
    "ttl",
    "rtt.ms.min",
    "rtt.ms.max",
    "rtt.ms.avg",
    "rtt.ms.stdev",
    "packets.ip"
  ],
  "version": 0,
},
{
  "capability": "measure",
  "label": "glimpse-reversednslookup",
  "parameters": {
    "destination.ip4": "*"
  },
  "registry": "https://www.measure-it.net/static/glimpse_registry.json",
  "results": [
    "glimpse.dns.record-name",
    "glimpse.dns.record-value"
  ],
  "version": 0,
}
}

```

Probe execution environment

GLIMPSE can be executed on every environment supporting Qt5 (<http://www.qt.io/>). This includes:

- Linux (tested on Ubuntu, Debian, and Arch Linux)
- Embedded Linux
- Microsoft Windows

- Mac OSX
- Android (version 2.3 and higher)
- iOS
- WinRT

Note that not all measurement tools might be available on all platforms at the moment due to technical limitations of certain platforms. At the moment the development version is mainly tested on Linux and Android.

Quick start

There are two GLIMPSE-releases available: the **standard release** of GLIMPSE found at <http://www.measure-it.net> and the Github repository (https://github.com/HSA-net/glimpse_client) automatically connects to the GLIMPSE infrastructure and does multiple things which are out of scope of mPlane (for example user- and device-registration). To use this version follow the instructions under "**Installation of standalone probe (standard release)**".

The **infrastructure-less release** of GLIMPSE was stripped of all additional server-communication and can connect to any mPlane compliant supervisor. Infrastructure-less, since the GLIMPSE live infrastructure is not involved, but any mPlane-compliant infrastructure components can be used in this release. Using this version, no registration is required and it allows to do arbitrary testing with GLIMPSE. This release enables a user to set up a GLIMPSE probe, connect it to the mPlane reference implementation supervisor and interact with it through the mPlane command line interface by sending specifications and receiving results. To use this release follow the instructions under "**Integration into an mPlane environment (infrastructure-less release)**".

The following restrictions apply to the infrastructure-less release at the moment (this might change in the future):

- Only the console version is available
- The measurements which need another probe as end-point are disabled
- Temporal scopes with a crontab are implemented but disabled until further tested

Both releases of GLIMPSE (standard and infrastructure-less) can be installed with the help of pre-build binary packages or by compiling it yourself.

Installation of standalone probe (standard release)

Pre-build packages can be found at <http://distributor.measure-it.net/packages/>, note that the Windows installer was not fully tested yet. We recommend using the ubuntu-trusty packages for which the detailed installation process is described here:

- Add the package repository to your system: `echo "deb http://distributor.measure-it.net/packages/ ubuntu-trusty/" | sudo tee /etc/apt/sources.list.d/glimpse.list`
- Install GLIMPSE (console version): `sudo apt-get update && sudo apt-get install glimpse-console`
- To update GLIMPSE regularly (which we highly recommend): `sudo apt-get install cron-apt && echo "install glimpse-console -y --force-yes --allow-unauthenticated" | sudo tee /etc/cron-apt/action.d/2-glimpse`

The GLIMPSE console client will start automatically after a reboot (initial setup as described in the next section has to be completed). To change this you can disable the service glimpse-console.

If you want to use the GUI version install glimpse-gui and stop the glimpse-console background service before using it:

- `sudo apt-get install glimpse-gui`
- `sudo service glimpse-console stop`
- For automatic updates: `sudo apt-get install cron-apt && echo "install glimpse-gui -y --force-yes --allow-unauthenticated" | sudo tee /etc/cron-apt/action.d/2-glimpse`

To **compile the software** you should download and install Qt and QtCreator from [here](#). At the time of this writing, GLIMPSE was tested with Qt 5.2.1. The requirements for the different platforms are:

- Linux: libwnck (Arch & Gentoo) libwnck-dev (Debian & Ubuntu based) libwnck-devel (for RPM), openssl
- Android: Android SDK and NDK
- Windows: openssl, WinPcap

Clone and initialize the GLIMPSE Github repository:

- `git clone git@github.com:HSA-net/glimpse_client.git`
- `cd glimpse_client && git submodule init && git submodule update`

Run QtCreator, open the client.pro from the source-code, add your Qt configuration and select the "mobile" (= GUI version) or "console" project near the "Play" button. Hit "build" to compile GLIMPSE.

Usage of standalone probe

This section is divided into the execution of the application itself including the setup on the first run and how to run measurements within GLIMPSE.

Execution and first time setup

After the installation you need to register a GLIMPSE-account or login into an existing one. The GUI version does this on startup with a wizard, for the console version execute the following commands (the program will prompt for the password to register with or to login with and exist afterwards):

- Registration: `sudo -H -u glimpse glimpse-console --register <mail address>`
- Login: `sudo -H -u glimpse glimpse-console --login <mail address>`

Note: if you have compiled the software yourself you don't need to use sudo and you can just execute `glimpse-console` directly.

To start the GLIMPSE console version (prebuild):

- In the foreground: `sudo -H -u glimpse glimpse-console`
- In the background: `sudo service glimpse-console start`

Note: if you have compiled the software yourself you don't have the start up service.

To start the GLIMPSE GUI version (prebuild):

- `glimpse-gui`

To start the GLIMPSE console or GUI version from within the Qt Creator:

- Hit the play button after selecting the "mobile" or "console" project.
- For the console version you still have to register/login with command line arguments like described before, you can set them in the project-settings.

Measurements

GLIMPSE will receive specifications from our servers and execute them in the background. You can execute specifications yourself in the GUI version with the help of the Toolbox and view results on the Results page (the later is still a working in progress). The console version does not display results or allow the execution of own measurements, but the results from remotely scheduled measurements can be viewed in the project directory (`~/local/share/HS-Augsburg/mPlaneClient/results/`).

Integration into an mPlane environment (infrastructure-less release)

Apply the same installation-steps as described for the [pre-build console version](#) above but use "http://distributor.measure-it.net/packages_mplane/" as package repository. You don't need to do the first time setup for this version.

To start GLIMPSE and connect it to an mPlane compliant supervisor use

- `glimpse-console --supervisor <address of supervisor>`

This pushes the available capabilities to the supervisor and pulls new specifications regularly. Here are the steps to use the reference implementation with GLIMPSE

- start `mpspec` with the config set to component-initiated workflow and using http for communicating with the component (`listen-spec-link`)
- start `mpcli` to connect to the supervisor (`listen-cap-link`)
- start `glimpse-console` with the address and port of the supervisor (using http and the address of `listen-spec-link`)
- get the capabilities in the `mpcli` (address of `listen-cap-link`)
- use `showcap`, `runcap`, `listmeas` and `showmeas` to show and execute capabilities and see the results once available

Official versions

- May 15th, 2014 - frozen release for D2.2 [[tarball](#)] [[svn](#)]
- June 15th, 2015 - frozen release for D2.3 [[github tag standard release](#)] [[github tag infrastructure-less release](#)]

New features supported by the mPlane project

GLIMPSE was completely developed as part of the mPlane project.

Changes since D2.2

The mPlane proxy interface was replaced by an infrastructure-less release version which can connect to any mPlane compliant

2015. 07. 14.

GLIMPSE | Building an Intelligent Measurement Plane for the Internet

supervisor. This makes it possible for other partners to use GLIMPSE with their own mPlane infrastructure.

The DNS and reverse DNS lookup measurements were added.

The GUI version now displays results in a table view instead of just dumping JSON files.

References

[GLIMPSE project page](#)

Links to sources, binaries

[GLIMPSE Github project \(standard release\)](#)

[GLIMPSE Github project \(infrastructure-less release\)](#)

[GLIMPSE binary packages \(standard release\)](#)

[GLIMPSE binary packages \(infrastructure-less release\)](#)

File:

 [914glimpse.tar.bz2](#)

 [953license.txt](#)



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)


[Home](#) > [SOFTWARE](#) > [MobiProbe](#)

Mobile Probe (Android)

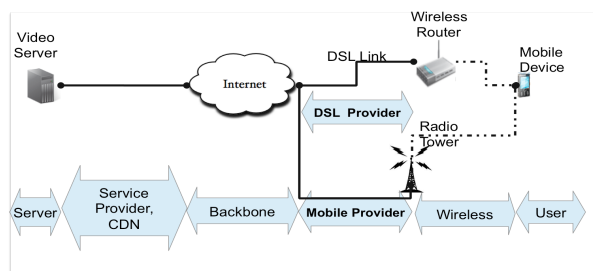
[View](#)[Edit](#)

Description:

The Mobile Probe is a probe designed to detect video streaming QoE issues on mobile devices and perform root-cause analysis to expose the fault that caused the said issue. This is achieved by collecting multiple performance metrics from different layers, including network, hardware, link and OS, which are subsequently aggregated and sent to a remote server.

In more detail, the probe periodically launches a YouTube video from a list of videos with variable quality and length, while it logs network, hardware, link and OS measurements in the background. Measuring the performance of each layer allows not only the detection of QoE issues but also the identification of the root cause of the problem as it may originate from multiple points along the path (see figure below). When a number of measurements become available, they are aggregated and sent to a database on a remote server.

For the network measurements the application uses a tstat binary pre-compiled for Android devices which is packed inside the installation file. All of the remaining metrics are collected with functionality directly implemented in the application.



Usage Scenarios:

The Mobile Probe can be run on Android devices using either WiFi or cellular connectivity. Moreover, it is agnostic to video characteristics such as quality, codec, resolution, size and duration, while it is compatible with content delivered over HTTP or HTTPS.

Metrics and Capabilities:

- **Hardware:** CPU usage, free memory, received/transmitted/dropped bytes from the network interface
- **Location:** location information from GPS and WiFi
- **Wireless:** Received Signal Strength, connectivity state, wireless technology used, cell tower information
- **System:** Playback state, re-buffering events, re-buffering duration, load time, HTTP requests, video decoder state
- **Network:** statistics per tcp flow as provided by tstat

Probe execution environment:

The Mobile Probe requires a modern Android device with root access in order to allow tstat to capture live traffic on the network interface.

Prerequisites:

It is required to have the official YouTube application installed, which is available from Google Play Store. The separate installation of the pcap library is no longer required as it comes bundled with the installation file.

Installation:

To install the probe the following steps are required:

- Download the application on the device
- Install the application*

- When the application starts it will ask for superuser permission which you need to grant
- When prompted to complete action using Browser or YouTube, select YouTube and Always

*If installation fails you need to enable 'Unknown sources' that is found in Settings->Security

General Usage:

The probe's functionality is automated for the most part and little interaction with the user is required. When launched, the application will randomly select and launch a video from a predefined list and repeat the process when the playback is finished.

The user can select three options the application's menu:

- Restart Monitoring
- Stop Monitoring
- Exit

The function of each option is self-explanatory.

Integration into an mPlane environment:

The data from the probes are collected in the remote server in a Mongo database as repository (the **repository** is part of D3.4). The proxy Interface is written in python and it sits on top of MongoDB. It exports the collections of mongoDB as capabilities using the Reference Implementation. Afterwards, the mPlane clients can pull the requested data.

New features:

- Python based proxy compatible with the new Reference Implementation.
- Separate installation of the pcap library is no longer required as it comes bundled with the application.
- Received signal strength is now supported for 3G/LTE connections as well.
- Byte counts on the network interface allows to calculate new features such as the interface utilization.

Repository Description:

The repository consists of a MongoDB database and a node.js interface. The Mobile Probe sends HTTP requests from the device to the server with the aggregated measurements. The node.js script is in turn responsible for receiving, parsing and storing the incoming data to the database. Each measurement is inserted with fields such as timestamp, the device's unique ID and IP address. In this way, the repository can handle concurrent reports from multiple devices. As can be seen in the provided sample below, a report contains the required performance metrics from the different layers that the probe is monitoring.

A detailed description of the data stored in the repository can be found here: <https://www.ict-mplane.eu/public/mongo-db>

mPlane proxy interface

The proxy can be found here: <https://github.com/fp7mplane/components/blob/master/mongoDB-mobileProbe/>

The proxy Interface is written in python and it sits on top of MongoDB. It exports the collections of mongoDB as capabilities using the Reference Implementation. Afterwards, the mPlane clients can pull the requested data.

The mongoDB proxy can be used by any component that has data stored in a mongoDB database. For details on how to install and run the probe visit the github help file:

<https://github.com/fp7mplane/components/blob/master/mongoDB-mobileProbe/README.md>

Official version

- July 6th, 2015, frozen release for D2.3: [[tar.gz](#)]

File:

 [1130mobileprobe.tar.gz](#)

2015. 07. 17.

Mobile Probe (Android) | Building an Intelligent Measurement Plane for the Internet



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



Privacy

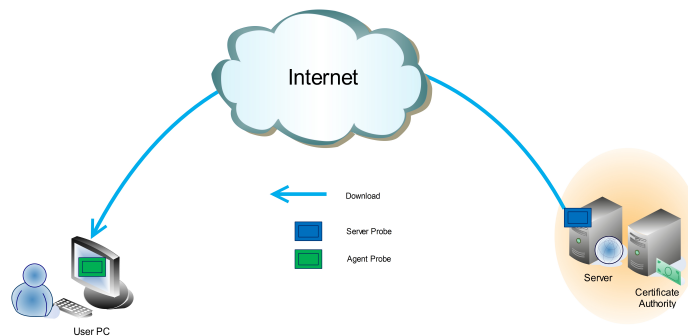

[Home](#) > [SOFTWARE](#) > [MSLACert](#)

mSLACert Active probe

[View](#)
[Edit](#)

Description:

The main component of the probe are the server and the client. The probe can do RTT, TCP and UDP measures to verify the SLA. mSLACert uses ICMP to measure the two way delay between server and client. The server sends 10 pings and calculates the RTT of the connection between server and client as the mean of the 10 samples.



mSLACert measures TCP and UDP throughput using IPERF [<http://iperf.fr>], a tool that does active measurements, it supports customization of various parameters like timing, buffers and protocols. Iperf can measure the maximum achievable bandwidth on a network and it reports the bandwidth, time, jitter loss and other parameters [<http://iperf.fr>]. The TCP throughput it is measured as the min, mean and the maximum bps between Server and client. The same is for UDP throughput, the probe measures the min, mean and the maximum bps in the transfer between server and client. In the UDP measure the probe also measures the mean throughput that the client has received also the mean jitter and percentage of datagram loss.

Metrics and Capabilities

Metrics

mSLACert offers a few measurement metrics, we can identify the following metrics:

time: duration of measurement

start: start time of a measure that may have a non-zero duration

end: end time of a measure that may have a non-zero duration

source.ip4: source IPv4 address from which an active measurement was taken

destination.ip4: the destination IPv4 address of the target of an active measurement

Capabilities

mSLACert has 8 capabilities, for the measurement of RTT, TCP and UDP throughput:

ping-average-ip4: This capability enables the probe to measure the RTT and to report back the min, mean and maximum value of RTT.

ping-detail-ip4: This capability enables the probe to measure the RTT and to report back the value of the RTT for every sample/ second.

tcpsla-average-ip4: This capability enables the probe to measure the TCP throughput and to report back the min, mean and maximum value of throughput.

tcpsla-detail-ip4: This capability enables the probe to measure the TCP throughput and to report back the value of the TCP

throughput for every sample/ second.

udpsla-average-ip4: This capability enables the probe to measure the UDP throughput and to report back the min, mean and maximum value of throughput, additionally it will report back the mean value of throughput that the client has received the percentage of datagram loss and jitter, in this case it cannot retrieve this information it will display -1 for this measurements. The UDP test will last 2 seconds less than the assigned time, to give time to the server to get the report from the client.

udpsla-detail-ip4: This capability enables the probe to measure the UDP throughput and to report back the value of the UDP throughput for every sample/ second, additionally it will report back the mean value of throughput that the client has received the percentage of datagram loss and jitter, in this case it cannot retrieve this information it will display -1 for this measurements. The UDP test will last 2 seconds less than the assigned time, to give time to the server to get the report from the client.

msla-average-ip4: This capability will enable the probe to launch RTT, TCP and UDP test one after another. It will report back the min, mean and maximum value of RTT, TCP and UDP throughput, additionally it will report back the mean value of UDP throughput that the client has received the percentage of UDP datagram loss and jitter. This capability will do three consecutive tests, and the time assigned to this capability will be divided by three, so every test will have 1/3 of the total time.

msla-detail-ip4: This capability will enable the probe to launch RTT, TCP and UDP test one after another. It will report back the value of RTT, TCP and UDP throughput for every sample/ second, additionally it will report back the mean value of UDP throughput that the client has received the percentage of datagram loss and jitter. This capability will do three consecutive tests, and the time assigned to this capability will be divided by three, so every test will have 1/3 of the total time.

msla-AGENT-Probe-ip4: The presence of this capability shows that the client is available.

Capabilities

```
[mplane] showcap ping-average-ip4
capability: measure
  label    : ping-average-ip4
  link     : /
  token    : a74fabd15ef8bbaaf68eb106a6c1c54e
  when     : now ... future / 1s
  parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
  results ( 4):
    delay.twoway.icmp.us.min
    delay.twoway.icmp.us.mean
    delay.twoway.icmp.us.max
    delay.twoway.icmp.count
None
```

```
[mplane] showcap ping-detail-ip4
capability: measure
  label    : ping-detail-ip4
  link     : /
  token    : 75cd8c844dce30a09c579d9fc89caa3d
  when     : now ... future / 1s
  parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
  results ( 2):
    time
    delay.twoway.icmp.us
None
```

```
[mplane] showcap tcpsla-average-ip4
capability: measure
  label    : tcpsla-average-ip4
  link     : /
  token    : 915d4e930eb023e3f422f56753593947
  when     : now ... future / 1s
```

```

parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
results ( 4):
    mSLA.tcpBandwidth.download.ipperf.min
    mSLA.tcpBandwidth.download.ipperf.mean
    mSLA.tcpBandwidth.download.ipperf.max
    mSLA.tcpBandwidth.download.ipperf.timecountseconds

```

None

```

|mplane| showcap tcpsla-detail-ip4
capability: measure
label    : tcpsla-detail-ip4
link     : /
token    : daa89cba4700ad4c665ae814a1de8ca8
when     : now ... future / 1s
parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
results ( 2):
    time
    mSLA.tcpBandwidth.download.ipperf

```

None

```

|mplane| showcap udpsla-average-ip4
capability: measure
label    : udpsla-average-ip4
link     : /
token    : 397ec41d3c1f81adeb7b7602fec6e632
when     : now ... future / 1s
parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
results ( 7):
    mSLA.udpCapacity.download.ipperf.min
    mSLA.udpCapacity.download.ipperf.mean
    mSLA.udpCapacity.download.ipperf.max
    mSLA.udpCapacity.download.ipperf.jitter
    mSLA.udpCapacity.download.ipperf.error
    mSLA-Bandwidth_mean_Mbps-Correct-UDP
    mSLA.udpCapacity.download.ipperf.timecountseconds

```

None

```

|mplane| showcap udpsla-detail-ip4
capability: measure
label    : udpsla-detail-ip4
link     : /
token    : 5c8c07f06c41c0fd144adfaa9c5a4ff7
when     : now ... future / 1s
parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
results ( 5):
    time
    mSLA.udpCapacity.download.ipperf
    mSLA.udpCapacity.download.ipperf.jitter
    mSLA.udpCapacity.download.ipperf.error
    mSLA-Bandwidth_mean_Mbps-Correct-UDP

```

None

```

|mplane| showcap msla-average-ip4
capability: measure
label    : msla-average-ip4
link     : /
token    : 824f3777947371bee6192e222123f53f

```

```

when      : now ... future / 1s
parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
results   (15):
    delay.twoway.icmp.us.min
    delay.twoway.icmp.us.mean
    delay.twoway.icmp.us.max
    delay.twoway.icmp.count
    mSLA.tcpBandwidth.download.iperf.min
    mSLA.tcpBandwidth.download.iperf.mean
    mSLA.tcpBandwidth.download.iperf.max
    mSLA.tcpBandwidth.download.iperf.timecountseconds
    mSLA.udpCapacity.download.iperf.min
    mSLA.udpCapacity.download.iperf.mean
    mSLA.udpCapacity.download.iperf.max
    mSLA.udpCapacity.download.iperf.timecountseconds
    mSLA.udpCapacity.download.iperf.jitter
    mSLA.udpCapacity.download.iperf.error
    mSLA-Bandwidth_mean_Mbps-Correct-UDP
None

```

```

[mplane] showcap msla-detail-ip4
capability: measure
label      : msla-detail-ip4
link       : /
token      : ea19803847ef4189974870e557d6eab1
when       : now ... future / 1s
parameters ( 2):
    source.ip4: 1.2.3.4
    destination.ip4: *
results    ( 7):
    time
    delay.twoway.icmp.us
    mSLA.tcpBandwidth.download.iperf
    mSLA.udpCapacity.download.iperf
    mSLA.udpCapacity.download.iperf.jitter
    mSLA.udpCapacity.download.iperf.error
    mSLA-Bandwidth_mean_Mbps-Correct-UDP
None

```

Probe execution environment

mSLACert runs on Linux (tested on Ubuntu 12.04 and 14.04), it is written in python. It requires Iperf to be installed on both server and client machines.

Installation of standalone probe

mSLACert is integrated with mPlane supervisor, the frozen version now runs only with mplane protocol.

The probe is really simple to install, just needs to have its requirement full filled:

Install, Yalm, Iperf and Tornado

1. `sudo apt-get install iperf`
2. `sudo apt-get install python3-yaml`
3. `sudo apt-get install python3-tornado`
4. Download mSLACert files [<https://github.com/etego/msla>]
5. Configure the configuration file for the certificates `./conf/component-certs.conf` and `./conf/supervisor-certs.conf`, also `./conf/client-certs.conf`, if you will be using the client.

There are needed two PCs, one for the server and one for the client, with linux installed on them.

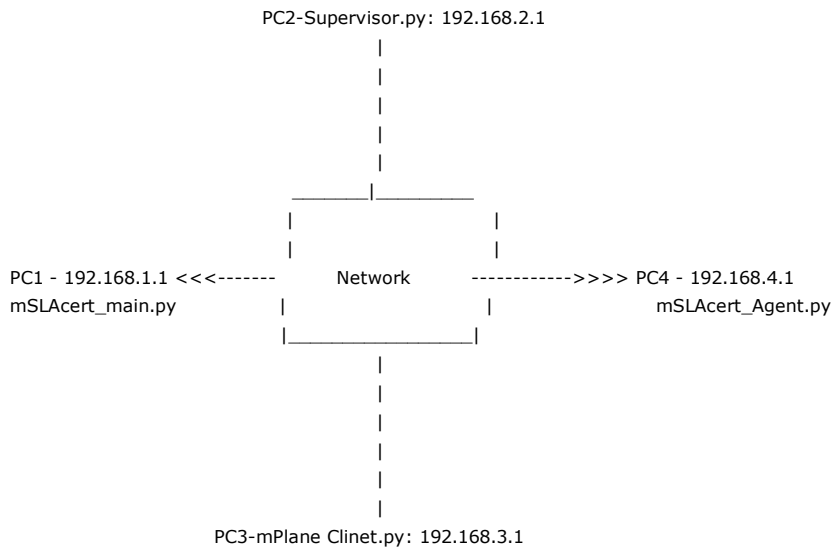
Usage of standalone probe

mSLACert runs only as an integrated probe with mPlane protocol, in cannot run as a standalone probe. For all the instruction on how to run it please see HOWTO-MSLACert.md on [<https://github.com/etego/msla>] or [<https://github.com/fp7mplane/components/tree/master/mSLACert%20-%20Probe>].

Integration into an MPlane environment

The probe is fully integrated with the mPlane environment, if you want a ready to test environment with the probe you can download [<https://github.com/etego/msla>]. To configure it on your own, here is a short guide on how to configure the .conf files of mPlane RI.

For the sake of simplicity we suppose the sequent scenario:



In base of you network configuration you have to change the sequent files, for the Ip and certificates:

```

./conf/client.conf
[TLS]
cert = PKI/ca/certs/"client-certificate".crt
key = PKI/ca/certs/"plaintext certificate.key"
ca-chain = PKI/ca/root-ca/root-ca.crt

[client]
# leave registry_uri blank to use the default registry.json in the mplane/ folder
registry_uri =
# http://ict-mplane.eu/registry/demo
# workflow may be 'component-initiated' or 'client-initiated'
workflow = component-initiated
# for component-initiated:
listen-host = "IP of the machine where is launched the client" (exmp:192.168.3.1)
listen-port = 8891
listen-spec-link = https://127.0.0.1:8891/
registration-path = register/capability
specification-path = show/specification
result-path = register/result
# for client-initiated:
capability-url = "IP supervisor":8890/ (exmp:192.168.2.1:8890/)

./conf/component*.conf

[TLS]
cert = PKI/ca/certs/"Components-certificate".crt
key = PKI/ca/certs/"plaintext certificate".key
ca-chain = PKI/ca/root-ca/root-ca.crt
  
```

```

[Roles]
org.mplane.FUB.Clients.CI-Client_FUB = guest,admin
"add also the roles for all the other components, client, supervisor ect"

[Authorizations]
msla-AGENT-Probe-ip4 = guest,admin
"add the capability of your probe"

[module_mSLA_main]
module = mplane.components."name of python file"
ip4addr = 1.2.3.4

[component]
scheduler_max_results = 20
# leave registry_uri blank to use the default registry.json in the mplane/ folder
registry_uri =
# http://ict-mplane.eu/registry/demo
# workflow may be 'component-initiated' or 'client-initiated'
workflow = component-initiated
# for component-initiated
client_host = "IP of the supervisor" (exmp: 192.168.2.1)
client_port = 8889
registration_path = register/capability
specification_path = show/specification
result_path = register/result
# for client-initiated
listen-port = 8888
listen-cap-link = https://127.0.0.1:8888/

./conf/supervisor.conf

[TLS]
cert= PKI/ca/certs/CI-Supervisor-FUB.crt
key= PKI/ca/certs/CI-Supervisor-FUB-plaintext.key
ca-chain = PKI/ca/root-ca/root-ca.crt

[Roles]
org.mplane.FUB.Components.mSLACert_server = guest,admin
org.mplane.FUB.Agent.mSLACert_Agent = guest,admin
org.mplane.FUB.Supervisors.CI_Supervisor_FUB = admin
Supervisor-1.FUB.mplane.org = admin
org.mplane.FUB.Clients.CI-Client_FUB = guest,admin

[Authorizations]
ping-average-ip4 = guest,admin
ping-detail-ip4 = guest,admin
tcpsla-average-ip4 = guest,admin
tcpsla-detail-ip4 = guest,admin
udpsla-average-ip4 = guest,admin
udpsla-detail-ip4 = guest,admin
msla-average-ip4 = guest,admin
msla-detail-ip4 = guest,admin
msla-AGENT-Probe-ip4 = guest,admin

[client]
# workflow may be 'component-initiated' or 'client-initiated'
workflow = component-initiated
# for component-initiated:
listen-host = "IP of the machine where is launched the supervisor" (exmp: 192.168.2.1)
listen-port = 8889
listen-spec-link =
# https://127.0.0.1:8889/
registration-path = register/capability
specification-path = show/specification
result-path = register/result
# for client-initiated:
component-urls: "IP of component 1":8888/, "IP of component 2":8888/ (exmp:

```

```
192.168.1.1:8888/,192.168.4.1:8888/)
```

```
[component]
scheduler_max_results = 20
# leave registry_uri blank to use the default registry.json in the mplane/ folder
registry_uri =
# http://ict-mplane.eu/registry/demo
# workflow may be 'component-initiated' or 'client-initiated'
workflow = component-initiated
# for component-initiated:
client_host = "IP of the machine where is launched the client" (exmp: 192.168.2.1)
client_port = 8891 / 9911
registration_path = register/capability
specification_path = show/specification
result_path = register/result
# for client-initiated:
listen-port = 8890
listen-cap-link =
# https://127.0.0.1:8890/
```

New features supported by the mPlane project

mSLACert was entirely developed within mPlane project.

Changes since D2.2

There have been small changes since 2.2:

1. The agent registers to the supervisor
2. Added msla-*-ip4 capabilities.

References


mPlane proxy interface: <https://github.com/etego/msla>

Iperf: <https://github.com/esnet/iperf>

mPlane repo: <https://github.com/fp7mplane/components/tree/master/mSLACert%20-%20Probe>

File:

 [1092mslacertv3.0.1.zip](#)

 [1093howto-mslacert.md.pdf](#)



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)



Plane

Building an Intelligent Measurement Plane for the

[My account](#)[Log out](#)[Home](#)

OTT Probe

[View](#)[Edit](#)

OTT probe

Developers, contributors

Developer: [NETVISOR](#), H-1119 Budapest, Petzvál József utca 56., Hungary

Contributing partner(s): [FW](#), Via Caracciolo 51, 20155 Milano – Italy

Description

OTT Probe is an active probe to verify access "over-the-top" media content, i.e. various forms of live and recorded video transport over the Internet. It is designed to run from typical customer premises, like residential and business locations with broadband Internet access.

OTT Probe can access content offered in various popular adaptive streaming formats like Apple HLS, Microsoft Smooth Streaming and MPEG-DASH. Also OTT probe will access content in major proprietary formats like Youtube videos.

When accessing content OTT Probe basically emulates the behaviour of a standard media player application, i.e. it downloads content progressively, in synch with the relative presentation time. However, compared to media players the probe uses much less resources (no decoding in most cases), and measures a number of quality parameters and attributes on the download process, which are offered as mPlane capabilities to its clients.

OTT Probe provides various strategies to access content which is available in multiple qualities for adaptive download: parallel download of all qualities, sequential shuffling between qualities, or the truly adaptive download.

The protocols used by the probe are listed below.

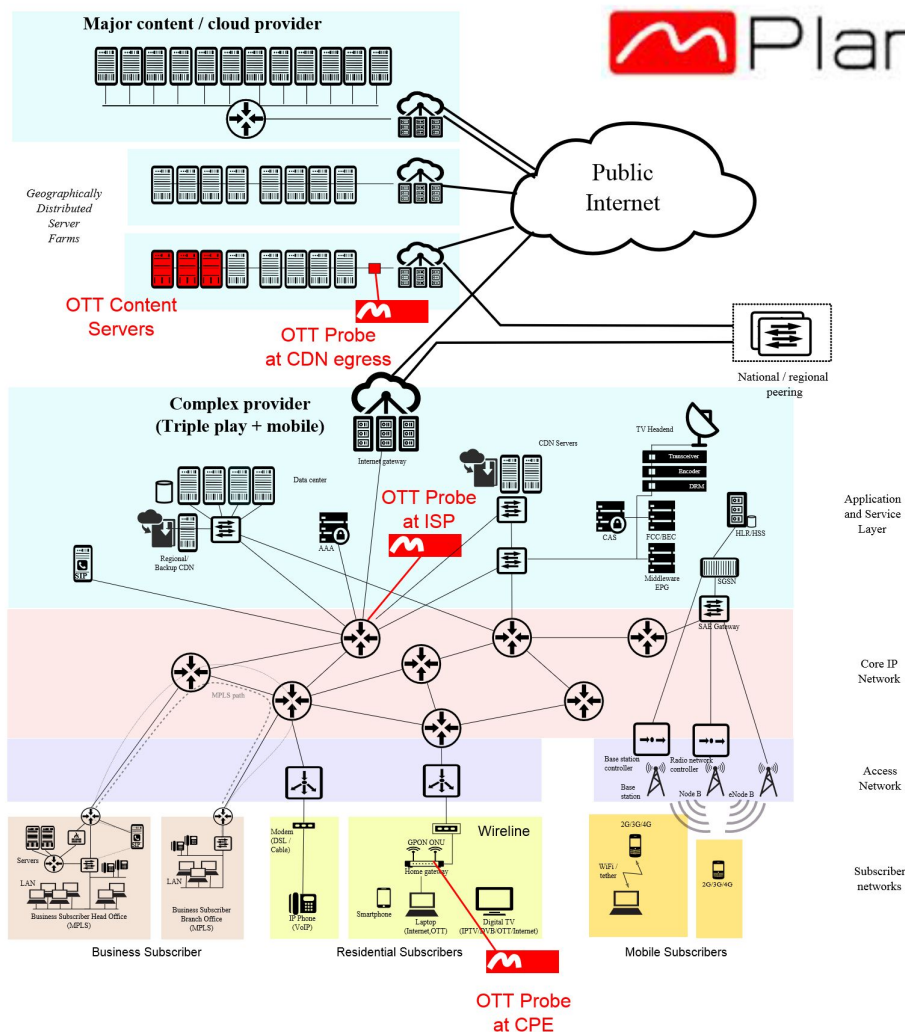
Transport protocols

- HTTP and HTTPS (ports depending on content server configuration)

Streaming formats over transport

- HLS (Apple), MSS (Microsoft), MPEG-DASH, Youtube/MP4 (Q2 2015)

Overview of Typical Usage Topologies for the OTT Probe



Usage scenarios

OTT Probes are to be used in either of two typical locations

- At some subscriber premises, where the probe downloads only one or a few streams at a time. This scenario is used for "End user experience" testing, where access issues or bottlenecks are a primary reason for non-perfect results.
- In some provider's premises, where a probe of larger capacity downloads dozens or hundreds of streams in parallel. If the provider is the service provider, the probes are simply used to verify that the components of the content are available and downloadable from the server. If the provider is an ISP, the focus may be on checking the availability of the content CDN from the ISP network.

Metrics and Capabilities

Metrics

The OTT probe offers a single capability, which measures access to some content specified by *source.url* parameter. The other parameter, *source.ip4* is used to define the source address OTT probe measures from. The maximum frequency of downloads is 10s, i.e. this is the minimum length of a measurement cycle that produces results.

Return values:

- time*: time of measurement

- *bandwidth.nominal.kbps*: the highest bandwidth of the best quality accessed during the period
- *http.code.max*: the maximum value of the http response codes
- *http.redirectcount.max*: the maximum value of the redirect counts measured during the period
- *qos.manifest*: a 0-100 value representing the relative quality of the manifest downloads and processing
- *qos.content*: a 0-100 value representing the relative quality of the content download
- *qos.aggregate*: a 0-100 value representing the average of relative quality of the manifest downloads and processing, and the content download, respectively
- *qos.level*: overall service level, based on the above metrics

Capabilities

```
{
  "capability": "measure",
  "label": "ott-download",
  "parameters": {
    "content.url": "*",
    "source.ip4": "192.168.96.1"
  },
  "registry": "http://ict-mplane.eu/registry/core",
  "results": [
    "time",
    "bandwidth.nominal.kbps",
    "http.code.max",
    "http.redirectcount.max",
    "qos.manifest",
    "qos.content",
    "qos.aggregate",
    "qos.level"
  ],
  "token": "5915d71a91e46b7070298bc23e813b80",
  "version": 1,
  "when": "now ... future / 10s"
}
```

Probe execution environment

OTT probes require a Python3 environment (to run the proxy derived from RI/SDK), and a binary component (aka standalone probe) that actually executes the downloads.

The binary is compiled for x86 Linux machines (with glibc), and for MiniProbes (MIPS32/uclibc). Should compile on other architectures as well. libcurl, libpthread and boost/program_options are required for compilation and execution.

Quick start

Installation of standalone probe

Prerequisites

Required libraries for the C++ module:

- libcurl.so.4
- boost_program_options
- libpthread
- libz
- libssl
- libreactor - published by NETvisor Ltd.
- probe-ott - published by NETvisor Ltd.

The published modules can be downloaded from <ftp://ftp.netvisor.hu/outgoing/mplane/mplane-ottmodule.tar.gz>. To speed up testing there is no need to compile it from source as it is already available for various platforms:

- ar71xx (tested on OpenWRT 12.04)
- x86_64 (tested on Ubuntu 14.04)

- o i386 (tested on CentOS release 6.5)

If there is any problem (or a new platform is requested) please contact tufa@netvisor.hu.

Installation

Copy probe-ott to your PATH (/usr/bin) and add libreactor to LD_LIBRARY_PATH. The easiest way to check that all the libraries are installed is to run the object file:

```
$ probe-ott
the option '--slot' is required but missing
```

If it fails with the aforementioned error, the measurement module is configured well.

Usage of standalone probe

```
probe-ott --slot <n> --url <url> [ --buffLen <n> --httpTimeout <n>
--protocol < HLS | IIS | MPEG-DASH > --downloadMode < 0 | 1 | 2 > --qualityLevel < n >
```

where

- o *slot* - the slot used to send the measurements to
- o *url* - manifest to download
- o *buffLen* - buffer length in seconds
- o *httpTimeout* - transaction timeout for the full HTTP transaction in millisecs
- o *protocol* - protocol to use. Could be HLS, IIS or MPEG-DASH. If not defined, it tries to guess
- o *downloadMode* - possible values are 0 (default): for adaptive download, 1: for fixed level download, 2: parallel download
- o *qualityLevel* - when *downloadMode* is 1, *qualityLevel* defines the download level

Integration into an MPlane environment

The probe implements the Component Initiated mPlane workflow. The proxy is developed from the mPlane RI and supports the mPlane HTTP API with or without https. It is tested with the mPlane RI standard supervisor (from the "master" branch).

In the following guide we use the following legend:

<ott-probe_installdir> the directory where the OTT-probe binary (probe-ott) has been installed (eg. /usr/bin)

<protocol-ri_dir> the directory where the GitHub repository of the mPlane protocol reference implementation has been cloned to (eg./home/<mplane_user>/protocol-ri)

<components_dir> the directory where the GitHub repository of the components (this repository) has been cloned to (eg./home/<mplane_user>/components).

Put <ott-probe_installdir> into the PATH variable, eg.

```
export PATH=$PATH:<ott-probe_installdir>
```

Put <protocol-ri_dir> into the PYTHONPATH variable, eg.

```
export PYTHONPATH=$PYTHONPATH:<protocol-ri_dir>
```

Copy the ott-probe interface stuff (<components_dir>\ott-probe folder) into <protocol-ri_dir>/mplane/components. This directory should contains the followings:

- o *ott-registry.json* The registry.json file containing the needed extensions for the OTT-probe (core registry is included within). It can be also downloaded from <http://tufaweb.netvisor.hu/mplane/ott-probe/ott-registry.json>. This is an extended version of the core JSON file, with all the needed definitions for ott-probe.
- o *ott.py* The Python interface
- o *supervisor.conf,client.conf, component.conf* The config files for running OTT in the component framework
- o *ott-capabilities* The capabilities file, does not needed for installation
- o *README.md* The installation guide.

Adjust the parameters in the *.conf files if needed (e.g. path to certificates, supervisor address, client port and address, roles, etc).

In a terminal window start supervisor:

```
root@h139-40:~/protocol-ri# test2@h139-40:~/protocol-ri$ scripts/mpsup --config mplane/components/o
ListenerHttpComponent running on port 8890
```

In another terminal start the OTT probe as a component and check if communication is established and the probe capabilities are registered with the supervisor:

```
test2@h139-40:~/protocol-ri$ scripts/mpcom --config mplane/components/ott-probe/component.conf
Added <Service for <capability: measure (ott-download) when now ... future / 10s token b77698ec sch
Added <Service for <capability: measure (ping-average-ip4) when now ... future / 1s token a74fabd1
Added <Service for <capability: measure (ping-detail-ip4) when now ... future / 1s token 75cd8c84 s
```

```
Capability registration outcome:
ping-detail-ip4: Ok
ping-average-ip4: Ok
ott-download: Ok
callback: Ok
```

Checking for Specifications...

Now we can start a clientshell in a third window to test the measurement functionalities:

```
test2@h139-40:~/protocol-ri$ scripts/mpcli --config mplane/components/ott-probe/client.conf
ok
mPlane client shell (rev 20.1.2015, sdk branch)
Type help or ? to list commands. ^D to exit.
```

```
|mplane|
```

Now check that capabilities are registered and run a measurement:

```
|mplane| listcap
Capability ott-download (token b77698ecef311f599940612f51ac7e27)
Capability ping-average-ip4 (token a74fabd15ef8bbaaf68eb106a6c1c54e)
Capability ping-detail-ip4 (token 75cd8c844dce30a09c579d9fc89caa3d)
|mplane| runcap ott-download
|when| = now + 30s / 10s
content.url = http://devimages.apple.com/iphone/samples/bipbop/bipbopall.m3u8
source.ip4 = 127.0.0.1
ok
|mplane| listmeas
Receipt ott-download-0 (token b5133e57c985a7194cb51caebf552bc9): now + 30s / 10s
|mplane|
```

During this we should see something similar in the component window, showing the launched application and then the receipt:

```
<Service for <capability: measure (ott-download) when now ... future / 10s token 0a01b50e schema 50
Will interrupt <Job for <specification: measure (ott-download-0) when now + 30s / 10s token 0a4f3ea
Scheduling <Job for <specification: measure (ott-download-0) when now + 30s / 10s token 0a4f3eac sc
```

```
2015-06-10 10:58:33.906406: running probe-ott --slot -1 --mplane 10 --url http://devimages.apple.co
```

After a while we can notice the result notification in the component window as the measurement has been finished:

```
<result: measure (ott-download-0) when 2015-06-10 10:58:34.969026 token 0a4f3eac schema 503f24c6 p/
Result for ott-download-0 successfully returned!
```

We can now check the results in the client window:

```
|mplane| listmeas
Receipt ott-download-0 (token b5133e57c985a7194cb51caebf552bc9): now + 30s / 10s
|mplane| listmeas
Result ott-download-0 (token b5133e57c985a7194cb51caebf552bc9): 2015-06-10 10:58:34.969026
|mplane| showmeas ott-download-0
result: measure
```


2015. 07. 14.

OTT Probe | Building an Intelligent Measurement Plane for the Internet

```
label      : ott-download-0
token      : b5133e57c985a7194cb51caebf552bc9
when       : 2015-06-10 10:58:34.969026
parameters ( 2):
              content.url: http://devimages.apple.com/iphone/samples/bipbop/bipbopal
              source.ip4: 127.0.0.1

resultvalues( 1):
  result 0:
              time: 2015-06-10 10:58:34.969026
              bandwidth.nominal.kbps: 720
              http.code.max: 200
              http.redirectcount.max: 0
              qos.manifest: 100
              qos.content: 100
              qos.aggregate: 100
              qos.level: 100
```

|mplane|

Official version

- June 15th, 2015 - frozen release for D2.3 on [mPlane SVN](#) or in the [tarball file](#).
- Latest version is available on [GitHub](#).

New features supported by the mPlane project

- Python based proxy for the mPlane RI SDK to enable its usage in mPlane Reference Implementation's distributed measurement environment.

Changes since D2.2

- Ported to new reference implementation version (master branch)
- Minor bugfixes

References

Links to sources, binaries

- [Official link on mPlane website](#)
- [Description on compilation and usage](#)
- [Standalone probe binaries](#)
- [Source code for mPlane proxy](#) ('master' branch at 'components' repository)

Links to additional documentation

- [iTVSense \(OTT probe product line\) - Solution Brief](#)
- ["iTVSense Probes Technologies" - Product Brief](#)

Dissemination

- [OTT probe demo video](#)

File:

 [Tarball version for D2.3 2015-06-15](#)

The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not

2015. 07. 14.

OTT Probe | Building an Intelligent Measurement Plane for the Internet



necessarily reflect the view of the European Commission. The information in this website 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.



Privacy



Plane

Building an Intelligent Measurement Plane for the

[My account](#)[Log out](#)[Home](#) > [SOFTWARE](#) > [Scamper](#)**Scamper**[View](#)[Edit](#)**Description:**

scamper is a parallelised packet-prober capable of large-scale Internet measurement using many different measurement techniques. Briefly, scamper obtains a sequence of measurement tasks from the input sources and probes each in parallel as needed to meet a packets-per-second rate specified on the command line. Tasks currently being probed are held centrally by scamper in a set of queues the probe queue if the task is ready to probe, the wait queue if it is waiting for time to elapse, and the done queue if the task has completed and is ready to be written out to disk. Each measurement technique is implemented in a separate module that includes the logic for conducting the measurement as well as the input/output routines for reading and writing measurement results, allowing measurement techniques to be implemented independently of each other. When a new measurement task is instantiated, the task attaches a set of callback routines to itself that \scamper then uses to direct the measurement as events occur, such as when it is time to probe, when a response is received, or when a time-out elapses. Sockets required as part of a measurement are held centrally by scamper in order to share them amongst tasks where possible so that resource requirements are reduced. Finally, scamper centrally maintains a collection of output files where completed measurements are written.

The scamper tools integrated in the mPlane framework are: ping, traceroute (various versions), trancelb and **tracebox**.

Metrics and Capabilities**Metrics**

Scamper's mPlane interface also allows to run the following tools, all tools are available for IPv4 and IPv6 networks:

- **ping**
- **trace** (traceroute)
- **trancelb** (a tool that infers all per-flow load-balanced paths between a source and destination.)
- **tracebox** (a middlebox detection and localization tool)

Capabilities

Scamper's mPlane interface has 7 capabilities, each of them is available for both IPv4 and IPv6 networks. To obtain the name of IPv4 and IPv6 versions of a capability, replace **V** respectively by 4 and 6. Every capability comes with multiple parameters that are documented in the Scamper man pages and in the **registry file**. Moreover, as many parameters are specific and not always needed, default value suggestion are available in the **README file**.

- scamper-ping-detail-ip**V**: This capability aims at measuring the round-time trip between a source and a destination and returns a value for every probe sent/received.
- scamper-ping-average-ip**V**: This capability aims at measuring multiple round-time trips between a source and a destination and returns minimum, maximum and means of observed RTT values.
- scamper-trace-standard-ip**V**: This capability make use of Scamper's traceroute implementation to measure IP-level routes between a source and a destination and RTTs to each intermediate cooperative hop.
- scamper-trancelb-standard-ip**V**: This capability is used to infer all per-flow load-balanced paths between a source and destination.
- scamper-tracebox-standard-ip**V**: This capability runs a simple tracebox probe over TCP between a source and a destination and aims at inferring middlebox modifications. For more detailed information about the tracebox algorithm, please refer to [this page](#).
- scamper-tracebox-specific-ip**V**: This capability runs a tracebox probe that you can precisely specify by setting IP fields value (ECN, DSCP, IPID/IPFLOW), the transport layer (TCP or UDP) and various TCP options/features (MSS, WScale, ECN, MPTCP, ...) and more (refer to scamper man page for an exhaustive list of probe definition possibilities). It returns all inferred middlebox modifications across the measured path. For more detailed information about the tracebox algorithm, please refer to [this page](#).
- scamper-tracebox-specific-quotesize-ip**V**: This capability is similar to scamper-tracebox-specific-ip**V**, but adds an additional result column containing the ICMP quoting size standard used by each hop. This information allows to identify a sub-path on which the modification occurred. For more detailed information about the tracebox algorithm, please refer to [this page](#).

Capabilities details

```
|mplane| showcap scamper-ping-average-ip4
capability: measure
  label      : scamper-ping-average-ip4
  link       : /
  token      : 2378eb9c718a2a2038a71f7afc10a73e
  when       : now ... future / 1s
  registry   : http://ict-mplane.eu/registry/core
  parameters ( 9):
    destination.ip4: *
    scamper.ping.tos: 0 ... 255
    scamper.ping.size: 84 ... 140
    scamper.ping.dport: 0 ... 65535
    scamper.ping.sport: 0 ... 65535
    scamper.ping.ttl: 0 ... 255
    scamper.ping.method: tcp-ack, udp-dport, tcp-ack-sport,icmp-echo, udp, icmp-time
    scamper.ping.rr: *
    source.ip4: *
  metadata   ( 3):
    System_type: Scamper
    System_ID: Scamper-Proxy
    System_version: 0.1
  results     ( 4):
    delay.twoway.icmp.us.min
    delay.twoway.icmp.us.mean
    delay.twoway.icmp.us.max
    delay.twoway.icmp.count
```

```
|mplane| showcap scamper-ping-average-ip4
capability: measure
  label      : scamper-ping-average-ip4
  link       : /
  token      : 2378eb9c718a2a2038a71f7afc10a73e
  when       : now ... future / 1s
  registry   : http://ict-mplane.eu/registry/core
  parameters ( 9):
    destination.ip4: *
    scamper.ping.tos: 0 ... 255
    scamper.ping.size: 84 ... 140
    scamper.ping.dport: 0 ... 65535
    scamper.ping.sport: 0 ... 65535
    scamper.ping.ttl: 0 ... 255
    scamper.ping.method: tcp-ack, udp-dport, tcp-ack-sport,icmp-echo, udp, icmp-time
    scamper.ping.rr: *
    source.ip4: *
  metadata   ( 3):
    System_type: Scamper
    System_ID: Scamper-Proxy
    System_version: 0.1
  results     ( 4):
    delay.twoway.icmp.us.min
    delay.twoway.icmp.us.mean
    delay.twoway.icmp.us.max
    delay.twoway.icmp.count
```

```
|mplane| showcap scamper-trace-standard-ip4
capability: measure
  label      : scamper-trace-standard-ip4
  link       : /
  token      : b2855fd8415186bc582fa0ddeba433f2
  when       : now ... future
  registry   : http://ict-mplane.eu/registry/core
  parameters (19):
    scamper.trace.firsthop: 0 ... 255
    scamper.trace.method: UDP, TCP, ICMP,UDP-paris, ICMP-paris, TCP-ACK
    scamper.trace.tos: 0 ... 255
```

```

scamper.trace.gaplimit: 0 ... 255
scamper.trace.loops: 0 ... 255
scamper.trace.confidence: 95.0 ... 99.0
scamper.trace.attempts: 0 ... 255
scamper.trace.T: *
destination.ip4: *
scamper.trace.M: *
scamper.trace.maxttl: 0 ... 255
scamper.trace.gapaction: 1 ... 2
source.ip4: *
scamper.trace.Q: *
scamper.trace.loopaction: 0.0 ... 1.0
scamper.trace.dport: 0.0 ... 65535.0
scamper.trace.sport: 0.0 ... 65535.0
scamper.trace.waitprobe: 0 ... 2550000
scamper.trace.wait: 0 ... 255

metadata (3):
  System_type: Scamper
  System_ID: Scamper-Proxy
  System_version: 0.1

results (3):
  scamper.trace.hop.ip4
  rtt.ms
  rtt.us

[mpplane] showcap scamper-tracelb-standard-ip4
capability: measure
label      : scamper-tracelb-standard-ip4
link       : /
token      : 68d36e39841c7ed875563ff1942b6d77
when       : now ... future
registry   : http://ict-mpplane.eu/registry/core
parameters (13):
  scamper.tracelb.dport: 0 ... 65535
  scamper.tracelb.maxprobec: 1 ... 10000
  scamper.tracelb.method: tcp-sport, udp-dport, tcp-ack-sport,udp-dport, icmp-echo, udp-sport
  scamper.tracelb.attempts: 0 ... 255
  destination.ip4: *
  scamper.tracelb.sport: 0 ... 65535
  scamper.tracelb.firsthop: 0 ... 255
  source.ip4: *
  scamper.tracelb.confidence: 95 ... 99
  scamper.tracelb.gaplimit: 0 ... 255
  scamper.tracelb.waitprobe: 0 ... 2550000
  scamper.tracelb.waittimeout: 0 ... 255
  scamper.tracelb.tos: 0 ... 255

metadata (3):
  System_type: Scamper
  System_ID: Scamper-Proxy
  System_version: 0.1

results (1):
  scamper.tracelb.result

[mpplane] showcap scamper-tracebox-standard-ip4
capability: measure
label      : scamper-tracebox-standard-ip4
link       : /
token      : 67f7f831d6c511b6b7cc31ffd642c3aa
when       : now ... future
registry   : http://ict-mpplane.eu/registry/core
parameters (2):
  destination.ip4: *
  source.ip4: *

metadata (3):
  System_type: Scamper

```

```

      System_ID: Scamper-Proxy
      System_version: 0.1
results  ( 2):
  scamper.tracebox.hop.ip4
  scamper.tracebox.hop.modifications

[implane] showcap scamper-tracebox-specific-ip4
capability: measure
label    : scamper-tracebox-specific-ip4
link     : /
token    : 9684b58ab96251ce8e279257d533289f
when     : now ... future
registry : http://ict-mplane.eu/registry/core
parameters ( 4):
  destination.ip4: *
  scamper.tracebox.probe: *
  source.ip4: *
  scamper.tracebox.dport: 0 ... 65535
metadata ( 3):
  System_type: Scamper
  System_ID: Scamper-Proxy
  System_version: 0.1
results  ( 2):
  scamper.tracebox.hop.ip4
  scamper.tracebox.hop.modifications

[implane] showcap scamper-tracebox-specific-quotesize-ip4
capability: measure
label    : scamper-tracebox-specific-quotesize-ip4
link     : /
token    : 2f733b848adc6d9e604569adc71b9bb0
when     : now ... future
registry : http://ict-mplane.eu/registry/core
parameters ( 4):
  destination.ip4: *
  scamper.tracebox.probe: *
  source.ip4: *
  scamper.tracebox.dport: 0 ... 65535
metadata ( 3):
  System_type: Scamper
  System_ID: Scamper-Proxy
  System_version: 0.1
results  ( 3):
  scamper.tracebox.hop.ip4
  scamper.tracebox.hop.modifications
  scamper.tracebox.hop.icmp.payload.len

```

Probe execution environment

Scamper is coded in C and no additional framework/libraries are needed. It is fully standalone (e.g.: It does not require any external services/applications to operate). It should compile and run under FreeBSD, OpenBSD, NetBSD, Linux (Ubuntu), MacOS X, Solaris, Windows and DragonFly.

Integration into the Monitored Network

The scamper probe will typically be deployed within the clients networks, but can technically be deployed anywhere in the Internet. Its only requirement is to be deployed in an ICMP-friendly network environment (e.g.: In-network firewalls allowing ingress ICMP traffic), and a UDP-friendly network environment for the UDP-based probes.

Usage of standalone probe & Integration into an MPlane environment

[Explained here.](#)

Quick start:

Building scamper from the source is very easy:

```

$ ./configure
$ make

```

```
$ sudo make install  
$ (optional) setuid 4755 /usr/local/bin/scamper
```

New features supported by the mPlane project

Thanks to the support of the mPlane project we extended scamper functionalities with the following features:

- [tracebox](#) support

Changes since D2.2

The [tracebox component](#) has been merged into the Scamper component.

References

- [Scamper source code](#).
- [mPlane proxy interface](#).

File:

 [1137scamper-07072015.tar.gz](#)



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)

Home > SOFTWARE > Tracebox

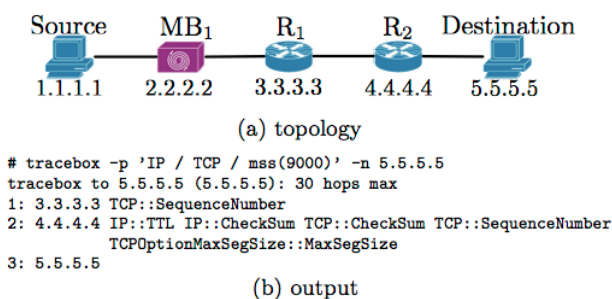
Tracebox

View

Edit

Description:

Tracebox is an open source topology discovery software that has been developed by the [Université de Liège](#) et by the Université catholique de Louvain in Belgium. Tracebox is an extension to the widely used traceroute tool. The objective of tracebox is to detect various types of middlebox interference over almost any path. To do so, tracebox sends IP packets containing TCP segments with different TTL values and analyses the packet encapsulated in the returned ICMP message. Further, as recent routers quote, in the ICMP message, the entire IP packet that they received, tracebox is able to detect any modification performed by upstream middleboxes. In addition, tracebox can often pinpoint the network hop where the middlebox interference occurs.



The figure above ((a) topology) shows a simple network, where MB₁ is a middlebox that changes the TCP sequence number and the MSS size in the TCP MSS option but that does not decrement the TTL. R₁ is an old router while R₂ is a router that is able to quote, in the returned ICMP message, the entire message that is responsible of the problem. The server always answer with a TCP reset. The output of running tracebox between "Source" and "Destination" is given by the below part of the figure ((b) output). The output shows that tracebox is able to effectively detect the middlebox interference but it may occur at a downstream hop. Indeed, as R₁ does not quote the full packet, tracebox can only detect the TCP sequence change when analyzing the reply of R₁. Nevertheless, when receiving the full message quoted from R₂, that contains the complete IP and TCP header, tracebox is able to detect that a TCP option has been changed upstream of R₂. At the second hop, tracebox shows additional modifications on top of the expected ones. The TTL and IP checksum are modified by each router and the TCP checksum modification results from the modification of the header.

Tracebox comes in three flavours:

- The original software, written in C++, allows some LUA embedding to easily extend tracebox capabilities. This is particularly interesting for discovering new types of middleboxes. This version supports two types of output: text (on the standard output) and pcap files.
- tracebox has been ported into **scamper**, an integrated tool for topology discovery. scamper comes with interesting tools in the context of mPlane: ping, traceroute (various implementations), Doubletree, TBit, ... scamper allows for IPv4 and IPv6 network probing. tracebox into scamper supports two types of output: text (on the standard output) and warts files (warts is a binary format for storing packets into the scamper system).
- tracebox has been ported into the Android system in order to support some mPlane use cases. Due to limitations inherent to mobile systems, the Android tracebox is a kind of light version of the original one. The output is sent to a back office where any kind of manipulation is made possible.

Tracebox important features:

tracebox brings two important features.

- **Middleboxes Detection.** tracebox allows one to easily and precisely control probe packets sent (IP header, TCP or UDP header, TCP options, payload, etc). Further, tracebox keeps track of each transmitted packet. This permits to compare the quoted packet sent back, in an ICMP time-exceeded message by an intermediate router, with the original one. By correlating the different modifications, tracebox is able to infer the presence of middleboxes.

- **Middleboxes Location.** using an iterative technique (in the fashion of traceroute) to discover middleboxes also allows tracebox to approximately locate, on the path, where modifications occurred and so the approximate position of middleboxes.

Quick start:

Original Software

The original software is freely available at tracebox.org (with the source code). The software works under Mac OS X, BSD, and Linux distribution. If you are under Mac OS X, the easiest way to install tracebox is to run homebrew (brew install tracebox).

Source can be found at <http://www.github.com/tracebox/tracebox>.

Tracebox requires:

- The development package of libpcap, (lib)lua >= 5.1.
- Automake, autoconf and libtool.

To build Tracebox:

```
$ ./bootstrap.sh
$ make
$ sudo make install
```

There are two possible ways to use tracebox either with the Python scripts (see some samples scripts in /tracebox/examples) or with the default binary. The later only sends one TCP probe and look for changes in the path.

Scamper Port

The current snapshot of scamper's source code is [cvs-20140404](#) and do not contain Tracebox. Scamper should compile and run under FreeBSD, OpenBSD, NetBSD, Linux, MacOS X, Solaris, Windows, and DragonFly. All releases of scamper are licensed under the GPL v2.

Scamper with tracebox can be found at: <https://github.com/fp7mplane/components/tree/master/scamper/source>

To build Scamper:

```
$ ./configure
$ make
$ sudo make install
```

The Scamper Tracebox implementation is the most complete and efficient. It involves different options to modify the text output format (e.g: traceroute-like output vs simplified middlebox locations only output, add the ICMP quoting size standard used by each hop, ...). It also contains other non-tracebox middlebox detection methods that are not interface within mPlane.

Android Port

The tracebox Android port is available at androidtracebox.org (with the source code).

To install tracebox Android:

1. Root your mobile phone (it is impossible to forge packet without the right privileges)
2. Install tracebox Android from the [Play Store](#).

tracebox Android usage is very intuitive. You can either enter yourself a destination to probe or download, from the back office (the source code of the back office is also available), a targets file.

mPlane proxy interface

The current mPlane Tracebox interface is part of the [scamper](#) component and contains 3 capabilities (each one subdivided for IPv4/IPv6):

- traceboxV_standard_capability: a simple tracebox probe over TCP without any options. V is the IP version.
- traceboxV_specific_capability: a tracebox probe that you can precisely specify by choosing IP fields value (ECN, DSCP, IPID/IPFLOW), the transport layer (TCP or UDP) and various TCP options/features (MSS, WScale, ECN, MPTCP, ...).
- traceboxV_specific_quotesize_capability: the same probe description than traceboxV_specific_capability with an additional result column containing the ICMP quoting size standard used by each hop

New features supported by the mPlane project

Thanks to the support of the mPlane project we extended tracebox functionalities with the following features:

- tracebox port into scamper
- tracebox deployment on IPv6 infrastructure
- tracebox port into Android

Changes since D2.2

The tracebox has been merged into the **Scamper component**.

File:

 **mPlane-compatible Scamper sources**



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)



Home > SOFTWARE > Tstat

Tstat

View

Edit

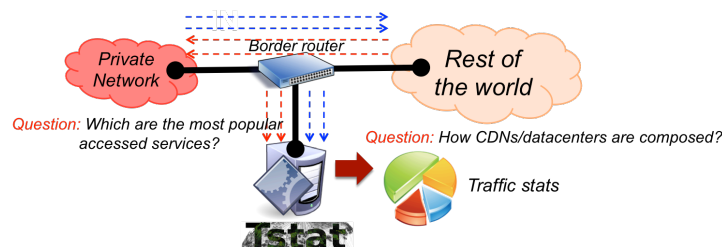
Revisions

Log

[Clone content](#)

Description:

Tstat is an open source passive traffic analyzer developed at **Telecommunication Network Group (TNG)** of **Politecnico di Torino**. It started as a branch of **TCPtrace** with a focus on TCP statistics only, but over the years it evolved in a full fledged monitoring solution offering an extensive set of measurements for IP, TCP and UDP, as well as traffic classification capabilities through a combination of Finite State Machine Deep Packet Inspection (FSM DPI) and Behavioural engines.



As shown in the figure, Tstat can produce real time analysis processing live traffic acquired from both standard PC NICs, using the **libpcap** standard interface, or dedicated solutions like **DPDK** for 10Gbps+ live analysis using off the shelf hardware, or dedicated solutions like **DAG cards**.

Tstat can also process previously recorded packet-level traces of various formats (**libpcap**, snoop, etherpeek, etc.).

Tstat supports different output formats:

- log files:** text files collecting per-flow stats. Each file is related to a different protocol or application (e.g., TCP/UDP traffic, video streaming, etc.); For each flow, Tstat will produce a detailed set of statistics like bytes/packets sent, received, retransmitted, Round-Trip-Time, application protocol, etc.
- histograms:** a set of histogram, each one related to a specific traffic features (e.g., incoming/outgoing IP bitrate, number of TCP flows, etc.). Statistics are collected over a time window (by default 5 minutes) and saved in separated text files for easy postprocessing.
- RRD:** a set of histograms saved in **RRD** format, which can be easily browsed to produce historical plots using a simple web interface -- or exported to be used by a **Graphite** GUI.
- pcap:** dump packets in pcap files, using advanced filtering, e.g., dump only BitTorrent or Youtube traffic. This is useful to capture packets trace of selected traffic, limiting disk usage.

Quick start:

We recommend the user to refer to the last version available on SVN:

```
svn checkout http://tstat.polito.it/svn/software/tstat/trunk tstat
```

The software works for Linux, BSD, Mac OS, and Android. To compile the software, from the Tstat main software folder

```
./autogen.sh
./configure
make
sudo make install
```

Last command is optional and it will install tstat into /usr/local/bin.

Note: to compile on Android, please refer to <http://tstat.polito.it/svn/software/tstat/trunk/doc/android.txt>

To run Tstat in live mode from a network interface DEVNAME (e.g., eth0) and create logs in the directory OUTDIR, run:

```
sudo ./tstat/tstat -l -i DEVNAME -s OUTDIR
```

For more information, please refer to the official website <http://tstat.polito.it> where complete and detailed **HOWTO** can be found.

New features supported by the mPlane project

Thanks to the support of the mPlane project we extended Tstat functionalities with the following features

- **HTTP module:** it allows to save text log files reporting information about HTTP queries/responses.
- **IP address anonymization:** it allows one to mask local IP address monitored using hashing functions using the **CryptoPan** to preserve prefixes while guaranteeing strong anonymization of addresses.
- **Blockmon integration:** we collaborated with **NEC** to integrate Tstat analysis modules in **Blockmon**. It means that Tstat can run as a Blockmon module, and output can be then processed in a streaming fashion.
- **log_sync:** a client/server application which allows to continuously export from Tstat logs from probes to repositories, offering both a bulk synchronization, and a streaming exporter.
- **improved log configurability:** rather than collecting a monolithic set of stats, Tstat now offers more fine-grained control on which set of features are saved in the logs. Per-flow stats are now grouped in macro classes which can be added or removed at runtime.
- **Improved internal configuration:** all parameters can now be configured easily in a separate file, without the need to recompile Tstat.
- **Improved performance** and optimized memory usage. In particular, now garbage collection operations cannot interfere with packet processing, and create possible packet loss.
- **Integration with high speed DPDK packet capture:** **ENST** and **POLITO** cooperated to build a DPDK-based engine based on Tstat capable of live processing of traffic at 40Gbps with suitable NICs. Check the **instructions** on the Tstat-DPDK branch.
- **Android integration:** thanks to the effort of **TID**, Tstat works also on rooted Android devices.
- **OpenWRT integration:** thanks to the effort of **NETVISOR**, starting from release 37196, the OpenWRT Linux distribution contains Tstat as a "Network Utilities" package.

mPlane proxy interface

Component-initiated (with supervisor)

Tstat mPlane proxy interface is available on [github](#). It allows to expose Tstat as a probe component to the Supervisor, which can then later be used to turn on and off features by setting capabilities, to start exporting results to repositories, etc.

Reminder: the whole mPlane Reference Implementation is built using [Python3](#). All libraries and software below thus refer to Python3 supported tools.

To test the software:

- **STEP1:** running Tstat

From the root folder of the Tstat software obtained doing a svn checkout

```
>>>sudo tstat -l -i eth0 -s tstat_output_logs -T tstat-conf/runtime.conf -R tstat-conf/rrd.conf
```

-l: enables live capture using libpcap

-i interface: specifies the interface to be used to capture traffic

-s dir: puts the results into directory "dir"

-T runtime.conf: configuration file to enable/disable logs at runtime

-R conf: specifies the configuration file for integration with RRDtool.

-r path: path where to create/update the RRDtool database

For more information, please refer to the official [Tstat website](#).

The example above runs Tstat in live capture mode, from the eth0 interface, using the default runtime and RRD configuration, and saving the logs in tstat_output_logs folder.

- **STEP2:** download and configure the Tstat mPlane proxy

```
>>> git clone https://github.com/fp7mplane/protocol-ri.git
>>> cd protocol-ri/mplane/
>>> mv components components.orig (or rm -rf components)
>>> git clone https://github.com/fp7mplane/components/
>>> cd ../
```

Add to the [Authorizations] section of the 'conf/supervisor.conf' file the new supported capabilities listed below:

```
tstat-log_http_complete = guest,admin
tstat-exporter_streaming = guest,admin
tstat-log_rrds = guest,admin
tstat-exporter_rrd = guest,admin
tstat-exporter_log = guest,admin
repository-collect_rrd = guest,admin
repository-collect_streaming = guest,admin
repository-collect_log = guest,admin
```

- **STEP3:** running the standard mPlane supervisor

Open a new terminal, enter 'protocol-ri' folder and execute:

```
>>> export PYTHONPATH=.
>>> ./scripts/mpsup --config ./conf/supervisor.conf
```

- **STEP4:** running the mPlane Tstat proxy

Open a new terminal, and enter 'protocol-ri' folder.

Change the paths inside Tstat's configuration files in ./mplane/components/tstat/conf/tstat.conf . If you run the Tstat as presented in **STEP1**, change the paths to:

```
runtimeconf = PATH_TO_TSTAT/tstat-conf/runtime.conf
tstat_rrd_path = /rrdfiles/
```

```
>>> export PYTHONPATH=.
```

```
>>> ./scripts/mpcom --config ./mplane/components/tstat/conf/tstat.conf
```

```
Added <Service for <capability: measure (tstat-log_tcp_complete-core) when now ... future t
Added <Service for <capability: measure (tstat-log_tcp_complete-end_to_end) when now ... fu
Added <Service for <capability: measure (tstat-log_tcp_complete-tcp_options) when now ... f
Added <Service for <capability: measure (tstat-log_tcp_complete-p2p_stats) when now ... fut
Added <Service for <capability: measure (tstat-log_tcp_complete-layer7) when now ... future
Added <Service for <capability: measure (tstat-log_rrds) when now ... future token f0bab5b4
Added <Service for <capability: measure (tstat-exporter_rrd) when past ... future token 58c
Added <Service for <capability: measure (tstat-log_http_complete) when now ... future token
Added <Service for <capability: measure (tstat-exporter_streaming) when now ... future toke
Added <Service for <capability: measure (tstat-exporter_log) when past ... future token ebl
```

NOTE: For dependencies, you need to install the [psutil](#), [unidecode](#) and the [rrdtool](#) python3 modules first. The easiest way is to use [pip3](#).

In the Supervisor screen you should see the Tstat proxy registering and exposing capabilities.

- **STEP5:** run the mPlane repository proxy

Open a new terminal, create a certificate using the 'create_component_cert.sh' script for repository proxy. Please refer

to [HOWTO.txt](#) for more information.

NOTE: it is recommended to use Repository-Polito as name for the certificate in order to be compatible with *tstatrepository.conf* by default.

Enter protocol-ri folder and execute:

```
>>> export PYTHONPATH=.
>>> ./scripts/mpcom --config ./mplane/components/tstat/conf/tstatrepository.conf

Added <Service for <capability: measure (repository-collect_rrd) when past ... future token
Added <Service for <capability: measure (repository-collect_streaming) when now ... future
Added <Service for <capability: measure (repository-collect_log) when past ... future token
```

NOTE: The repository proxy expected that the [Graphite](#) and [DBStream](#) are running on default setting.

As above, on the Supervisor window the capabilities of the proxies should be visible.

- **STEP6:** run the standard mPlane client, and connect to the Supervisor.

Open a new terminal, enter protocol-ri folder and execute:

```
>>> export PYTHONPATH=.
>>> ./scripts/mpcli --config ./conf/client.conf

ok
mPlane client shell (rev 20.1.2015, sdk branch)
Type help or ? to list commands. ^D to exit.
|mplane| listcap      # list the available capabilities
Capability repository-collect_log (token 0ccb3dc4c3290bbbb63caeb1a9f44a6d)
Capability repository-collect_rrd (token 5628ceb366cf23076ab131f701b6dbd0)
Capability repository-collect_streaming (token 39de2de6bf9e6b1423cb42f258a40683)
Capability tstat-exporter_log (token eble0c4f3b91673a52733948ef0a9c98)
Capability tstat-exporter_rrd (token 58d1109f591c69dba93193bc88688131)
Capability tstat-exporter_streaming (token 2ad7da68b76a91e9ed96c8d90c0df4b3)
Capability tstat-log_http_complete (token d8c8fa10aa833948024fc18477f1d69b)
Capability tstat-log_rrds (token f0bab5b4e6305b9faf5e4bdbdc4f71a5)
Capability tstat-log_tcp_complete-core (token 7ee0a2812d9decc8085f2b204df0af7d)
Capability tstat-log_tcp_complete-end_to_end (token 1ab5668dd6da4000cee08007cee23a73)
Capability tstat-log_tcp_complete-layer7 (token c03c028db352b9a41852cad68b37df4b)
Capability tstat-log_tcp_complete-p2p_stats (token 9963ddd359a679e6b48c0c9d0b282782)
Capability tstat-log_tcp_complete-tcp_options (token 68cc4936ffec4c0aab829796d8a07c74)
|mplane|
```

- **Activating passive measurements**

The Tstat mPlane proxy allows to activate and control all passive measurements offered by Tstat. Check [here](#) for a complete documentation.

For instance, to activate the collection of *Core TCP* set in *log_tcp_complete* for 30 minutes, enter in the client window:

```
|mplane| runcap tstat-log_tcp_complete-core
|when| = now + 30m
```

To activate the RRD collection forever, run:

```
|mplane| runcap tstat-log_rrds
|when| = now + inf
```

NOTE: To reset the scheduling option *when* you need to run:

```
|mplane| unset when
```

o Activating indirect log and RRD exporting

Now that Tstat started logging TCP flows, and filling in RRD data, we can start the Tstat repository proxy to export this data into mPlane compliant repositories. Currently the proxy offers three different indirect exporting approaches:

1. Log bulk exporter
2. Log streaming exporter
3. RRD exporter

1. Activating Log bulk exporter

The Tstat proxy sends log files collected by Tstat to the repository proxy. The log files are then stored in **DBStream**, using an asynchronous indirect export, supported by a custom protocol.

To activate the log indirect export from now to forever, run in the client:

```
|mplane| runcap tstat-exporter_log
|when| = now + inf
repository.url = localhost:3000
ok
```

NOTE: The *repository.url* contains the IP address of repository and the port value associated to *repository_log_port*.

2. Activating log streaming exporter

The exporter enable the streaming of logs collected in real-time by Tstat. It will create a direct and asynchronous channel with the repository, where rows in the selected logs will be piped into. The code contains in *tstatrepository.py* acts as a simple endpoint server which receives the streamed log and redirect them to stdout (or to a file -- just redirected stdout).

For instance, to activate the streaming indirect export of *log_tcp_complete* for 1 day, run at the client:

```
|mplane| runcap tstat-exporter_streaming
|when| = now + 1d
log.folder = /path/to/log/folder/    # this is where the logs are stored
log.time = 60
log.type = log_tcp_complete
repository.url = localhost:9001
ok
```

NOTE: The *repository.url* contains the IP address of the repository and the port value associated to *repository_streaming_port* in *tstatrepository.py*.

3. Activating RRD exporter

The Tstat proxy sends the RRD files collected by the Tstat to the repository proxy. The RRD files are then sent to **Graphite** for storage and graphical presentation using the Graphite (and possibly **Grafana**) GUI.

For instance, to activate the RRD indirect export from now, and for 1 hour, at the client interface just run:

```
|mplane| runcap tstat-exporter_rrd
|when| = now + 1h
repository.url = localhost:9000
ok
```

If the specification execute properly, the data will be available on Graphite web interface

NOTE: The *repository.url* contains the IP address of repository and the port value associated to *repository_rrd_port*.

Official version

- ◻ July 10th, 2015 - frozen release for D2.3
 - ◻ Tstat v3.0 [[tar.gz](#)] -- NEW -- we suggest to use the SVN version. See above.
 - ◻ Tstat-DPDK v0.9 [[tar.gz](#)] -- NEW -- we suggest to use the SVN version. See [this page](#).
 - ◻ Tstat-proxy [[tar.gz](#)] -- NEW -- this archive includes tstat.py, all tstat_exporters (bulk log, streaming and rrd), and the corresponding configuration files. However, we suggest to use the GitHub version. See above.
- ◻ May 15th, 2014 - frozen release for D2.2
 - ◻ Tstat v2.4 [[tar.gz](#)] -- OLD -- kept here for historical reasons.



The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website 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.



[Privacy](#)

2 Discontinued probes and name changes

In this section we give an overview of the probes which has been presented in D2.2 but are not presented in D2.3.

- **PerformanceVisor**
PerformanceVisor is NETvisor's proprietary commercial software. The mPlane proxy interface has been developed originally for deliverable D2.2. Since PerformanceVisor is not used in any of the WP5 demonstration Use Cases, it's mPlane proxy upgrade has been postponed.
- **QoF**
Its functionality has been migrated into *ECN-Spider*.
- **RILAnalyzer**
Its functionality has been migrated into *Mobile-Probe*.
- **probe-local-storage**
Its functionality has been migrated into *EZRepo* (see *D3.4*).
- **youtube-probe**
Its functionality has been migrated into *ott-probe*.