



mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Demonstrator Implementation Release and User Guidelines

Author(s):	Author names
A-LBELL	
ENST (ed.)	D. Rossi, D. Cicalese
FW	A. E. Kahveci, A. Sannino
POLITO	S. Traverso, M. Mellia
FUB	E. Tego, F. Matera
ALBLF	Z. Ben Houdi
EURECOM	M. Milanesio
NEC	M. Ahmed
TI	O. Jabr, F. Invernizzi
TID	I. Leontidas
NETvisor	J. Bartok Nagy, A. Bakay, G. Rozsa
FTW	A. D'Alconzo, P. Casas
ULg	
SSB	S. Pentassuglia
FHA	M. Faath
ETH	B. Trammell

Document Number: D6.2
Revision: 0.1
Revision Date: 30 Oct 2015
Deliverable Type: DEM
Due Date of Delivery: 30 Oct 2015
Actual Date of Delivery: 30 Oct 2015
Nature of the Deliverable: (R)eport
Dissemination Level: Public

Abstract:

This deliverable provides information to setup and demonstrate the different use cases. Its aim is not to faithfully reproduce all the fully-fledged mPlane demonstrations (shown at conferences, YouTube videos, etc.) but to provide a quick yet complete bootstrap guide to start replicating such demos. Specifically, for each use case the deliverable provides guidelines in required hardware, mPlane software and their configuration, as well as instructions to setup and interact with the mPlane software.

Contents

1	Executive summary.....	4
2	Generic Guidelines.....	6
2.1	Common instructions.....	7
2.2	Reference Implementation Instructions.....	9
2.3	Chain of certificates instructions	12
3	Use-case Guidelines.....	14
3.1	Estimating content and service popularity for network optimization	15
3.2	Passive content curation	18
3.3	Active measurements for multimedia content delivery.....	21
3.4	Quality of Experience for web browsing	24
3.5	Mobile network performance issue cause analysis	28
3.6	Anomaly detection and root cause analysis in large-scale networks	31
3.7	Verification and Certification of Service Level Agreements.....	35
3.8	Path transparency measurements	41

1 Executive summary

The main objective of WP6 is to show the capabilities of the mPlane platform based on the demonstration of a selected subset of functionality with respect to the use cases defined in WP1.

The main objective of Deliverable D6.2 is to empower users of mPlane technology with a set of easily reproducible instructions to install and replicate the mPlane use cases, subject to availability of needed equipment at the user premises. With respect to D5.5, a private deliverable that provides detailed instructions to replicate fully-fledged experiments in the Fastweb testplant, D6.2 represents a public counterpart that aims at providing generic instructions for bootstrapping such experiments.

D6.2 does not introduce per se new software components. Rather, guidelines in D6.2 make reference (and use) of mPlane components that already have extensively described in a series of software deliverable, namely D1.4 (mPlane protocol Reference Implementation, Supervisor and communication interfaces), D2.3 (programmable probes), D3.4 (repositories and big-data processing frameworks), and D4.4 (reasoners and analysis modules). While the aim of D1.4, D2.3, D3.4 and D4.4 is to extensively describes each individual component, the aim of D6.2 is to provide a simple syntactical glue among them. To facilitate replication of one or more use-cases for the practitioners, guideline description is reported in a templated fashion, so that workflow of each use-cases follows the same steps.

The full workflow include both steps that are common to all use cases, as well as specific instructions. Chapter 2 of this deliverable presents a reference demonstration environment (Sec.2.1), consisting of the minimal hardware and software requirements common to all use cases. This includes, e.g., steps to setup the reference implementation (Sec.2.2) as well as to setup a chain of certificates (Sec.2.3).

Chapter 3 then describes, for each use case, the demonstration steps to follow. Details about each use-case specific requirement, configurations and instructions, are reported in a templated fashion as follows:

- Hardware list (UC-specific equipment)
- Software list (in terms of mPlane components, repositories and reasoners needed by the UC)
- Software dependency list (in terms of additional third-party software that is used by, but not a main product of mPlane)
- Software installation (pointing to the relevant pages in the [mPlane website](#), or [GitHub](#))
- Software configuration (shall any mPlane component require specific per-UC configuration, it will be listed here)
- Step-by-step walkthrough (providing, for the sake of the example, a simple and easily repeatable sequence of commands, expected output and snapshots of the running software)

Following instructions in Chapters 2-- 3, users should be able to run demonstrations similar to those described in D5.5 and publicly demonstrated at several venues.

Notice that while this software deliverable collects information available from the mPlane website as well as GitHub, it is not meant to be directly used in PDF format¹: the primary primary vehicle of mPlane software and guidelines dissemination are meant to be the **mPlane website** and **GitHub**. D6.2 is thus a handy and informative collection of guidelines for all use-cases. Practitioners willing to follow these instructions step-by-step for one (or several) use-case(s) are thus invited to point their browser to <http://www.ict-mplane.eu/public/demonstration-guidelines> for the online version of instructions collected in D6.2.

¹Since Webpages are rendered as PDFs, hyperlinks are however lost in this process

2 Generic Guidelines

We start by reporting, common instructions:

- for the generic use case (Sec.2.1)
- to setup the reference implementation (Sec.2.2)
- to setup the chain of certificates (Sec.2.3)

<https://www.ict-mplane.eu/public/demonstration-guidelines>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

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

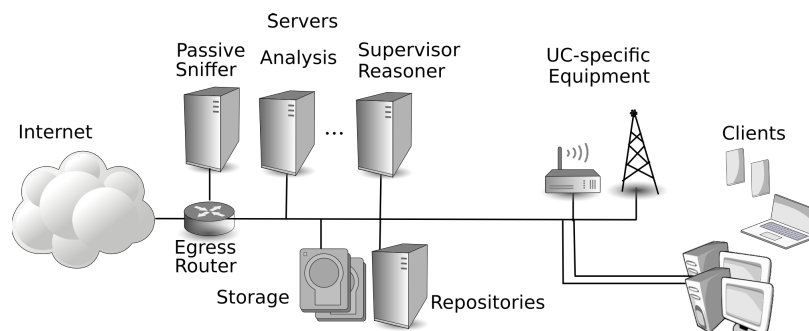
[Home](#) > [Demonstration guidelines](#)

Demonstration guidelines

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

[Clone content](#)

The main objective of **WP6** is to show the capabilities of the mPlane platform based on the demonstration of a selected subset of functionality with respect to the use cases defined in **WP1**. The main objective of this webpage is to empower users of mPlane technology with a set of easily reproducible instructions to install and replicate the mPlane use cases, subject to availability of needed equipment at the user premises.



From a high-level view point, all use cases (UC) can be described using a simplified reference demonstration environment illustrated in the figure above, that consists in the minimal hardware and software requirements common to all use cases. For instance, a single mPlane Supervisor can be used for all UCs, and similarly the same Analysis infrastructure (e.g., hadoop cluster) can be shared across UCs. All use cases will need you to go through these preliminary steps:

- [installing the reference implementation](#)
- [configuring the chain of certificates](#)

At the same time, all UCs have different requirements in terms of specific equipment and mPlane components, so that a per-UC description is required to assist their setup. Specifically, use-case specific requirements, configurations and instructions, are detailed in dedicated subpages per UC, according to the following template:

- Hardware list** (UC-specific equipment)
- Software list** (in terms of mPlane components, repositories and reasoners needed by the UC)
- Software dependency list** (in terms of additional third-party software that is used by, but not a main product of mPlane)
- Software installation** (pointing to the relevant pages in the mPlane website, or GitHub)
- Software configuration** (shall any mPlane component require specific per-UC configuration, it will be listed here)
- Step-by-step walkthrough** (providing, for the sake of the example, a simple and easily repeatable sequence of commands, expected output and snapshots of the running software)

You can access per-UC instructions by the Demonstration guideline menu, or through the link below:

- [Guidelines for **Estimating content and service popularity for network optimization**](#)
- [Guidelines for **Passive content curation**](#)
- [Guidelines for **Active measurements for multimedia content delivery**](#)
- [Guidelines for **Quality of Experience for web browsing.**](#)
- [Guidelines for **Mobile network performance issue cause analysis**](#)
- [Guidelines for **Anomaly detection and root cause analysis in large-scale networks**](#)

<https://www.ict-mplane.eu/public/demonstration-guidelines>

- Guidelines for **Verification and Certification of Service Level Agreement**

Following these instructions, users should be able to run demonstrations similar to those shown the **EuCNC exhibition**, the **mPlane Final workshop** and other venues.



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

<http://www.ict-mplane.eu/public/mplane-software-development-kit-sdk-python-3>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

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

mPlane Software Development Kit (SDK) for Python 3

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.

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 `mpsup`. 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.

<http://www.ict-mplane.eu/public/mplane-software-development-kit-sdk-python-3>

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.

Common Problems

1. If an error like this: `AttributeError: 'module' object has no attribute 'url'` shows up, try to update the `urllib3` library: `sudo pip3 install --upgrade urllib3`
2. If you get something like: `ssl.CertificateError: hostname '127.0.0.1' doesn't match 'Supervisor-1.SSB.mplane.org'`, try the following steps:
 - add the following entry in `/etc/hosts`: `127.0.0.1 Supervisor-1.SSB.mplane.org`
 - replace all "127.0.0.1" with "Supervisor-1.SSB.mplane.org" in the conf files

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

<http://www.ict-mplane.eu/public/mplane-software-development-kit-sdk-python-3>

- 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 erregards 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`.

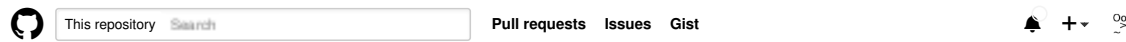


The information available on this website is property of the contributing authors from the mPlane Consortium (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission. The information in this website is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.



[Privacy](#)

<https://github.com/fp7mplane/protocol-ri/blob/master/PKI/README.md>



You don't have any verified emails. We recommend [verifying](#) at least one email.

Email verification helps our support team verify ownership if you lose account access and allows you to receive all the notifications you ask for.

fp7mplane / protocol-ri

Watch 30 Star 5 Fork 5

Branch: master **protocol-ri / PKI / README.md**

stepenta Renamed HOWTO to README in PKI folder cf71f75 3 days ago
1 contributor

43 lines (27 sloc) 1.87 KB

Raw Blame History

Simple instructions for Certificate Generation Scripts

This folder contains three main .sh scripts:

- create-component-cert.sh
- create-client-cert.sh
- create-supervisor-cert.sh

Configuration files for each of those scripts can be found in the the ./etc/ subfolder

Additionally, in case you don't care about compatibility with the PKI provided in this repository, we provide a further script:

- create-ca.sh

Its use is however not encouraged; see [Generating a new CA](#) below for more info.

Generating a certificate

To generate a certificate (e.g. a component certificate), follow these steps:

1. open the corresponding configuration file (./etc/component.conf)
2. modify the SAN field (e.g. replace DNS:Supervisor-1.SSB.mplane.org with DNS:Supervisor-1.Polito.mplane.org)
3. modify the fields in the [component_dn] section (you will be prompted for these fields while running the script, so you can also modify them later)
4. run create-component-cert.sh and follow the instructions:
 - enter filename of your certificate
 - enter PEM passphrase (passphrase to open your encrypted certificate)
 - enter the Distinguished Name
 - enter the root-ca passphrase (mPlan3_CA)
 - re-enter PEM passphrase
5. Certificate created in PKI/ca/certs/

Generating a new CA:

IMPORTANT: You can create your own CA and generate certificates dependent from that CA, but these will not be compatible with certificates provided in this repository. If you want to keep compatibility, use the provided CA (follow the steps in [Generating a certificate](#))

To generate a new CA, run create-ca.sh and follow the instructions:

- enter PEM passphrase (passphrase to open your encrypted certificate)

<https://github.com/fp7mplane/protocol-ri/blob/master/PKI/README.md>

- re-enter PEM passphrase

The certificate will be created in PKI/ca/root-ca/, and the directory structure for the PKI will also be created in PKI/ca/



3 Use-case Guidelines

We now report specific per use-case instructions for

- **Estimating content and service popularity for network optimization (Sec.3.1)**
- **Passive content curation (Sec.3.2)**
- **Active measurements for multimedia content delivery (Sec.3.3)**
- **Quality of Experience for web browsing (Sec.3.4)**
- **Mobile network performance issue cause analysis (Sec.3.5)**
- **Anomaly detection and root cause analysis in large-scale networks (Sec. 3.6)**
- **Verification and Certification of Service Level Agreements (Sec. 3.7)**
- **Path transparency measurements (Sec. 3.8)**

<http://www.ict-mplane.eu/public/guidelines-estimating-content-and-service-popularity-network-optimization>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Estimating content and service popularity for network optimization](#)

Guidelines for Estimating content and service popularity for network optimization

Requirements

This page details the requirements that are specific to the Content Popularity Estimation in addition to those expressed for the Reference demonstration environment ([link](#)).

Hardware list

- **Probe:** One dedicated machine running Tstat as passive traffic probe
- **Repository:** One dedicated machine for the repository running MondoDB and the analysis modules for the Content Popularity Estimation.

Software list

- mPlane protocol reference implementation ([GitHub repository](#))
- mPlane framework for the use-case ([GitHub repository](#))

Probes

- Tstat - Passive network traffic collection and analysis ([mPlane page](#), [GitHub repository](#))

Repositories

- MongoDB ([official page](#))

Reasoner

- Reasoner for the Content Popularity Estimation ([mPlane page](#))

Software dependencies

- Probe: Linux OS and Python3
- Repository: Linux OS, Python3 and MongoDB 3.0 or higher

Software installation

- Download and install Tstat; follow the instructions at [GitHub repository](#)
- Download and install MongoDB; follow the instructions at its [official page](#)
- Download and install the Tstat mPlane proxy interface; follow the instructions at [GitHub repository](#)
- Download and install the mPlane proxy for the repository, **tstatrepository.py** (customised for this specific use-case) and the reasoner, **cachereasoner**, from the [GitHub repository for this demo](#).

Run the software

- Run Tstat (check the [mPlane page](#) for details):

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

- Run MondoDB (check its [official page](#) for details):

```
sudo /etc/init.d/mongodb start
```

Run the mPlane software

Here we assume that [PROTOCOL_RI_DIR] is the folder where the GitHub repository of the mPlane protocol reference implementation has been cloned.

The code for the use case is available on GitHub under [mPlane demo material](#).

<http://www.ict-mplane.eu/public/guidelines-estimating-content-and-service-popularity-network-optimization>

Reasoner

cachereasoner is the Python script for the reasoner. It periodically runs the request which returns the list of contents to cache in a given server.

Supervisor

The Python code for the supervisor is the one provided in the mPlane protocol RI repository, i.e., **scripts/mpsup**. It receives the requests from the reasoner and forwards them to the repository and analysis module component.

Repository and Analysis Module

tstatrepository.py integrates the capabilities of communicating with Tstat and the interface to the query the analysis module.

HowTo

1. Set the parameters in the files **supervisor.conf**, **tstatrepository.conf**, **reasoner.conf** (e.g., path to certificates, supervisor address, client port and address, and roles)
2. Set the following parameters in the files **cacheController.py** and **tstatrepository.py** to connect to the Analysis module that estimates the content popularity of contents

```
_controller_address = Content Estimation analysis module address
_controller_port = Content Estimation analysis module port
```

3. Set the environment variable MPLANE_RI to point to [PROTOCOL_RI_DIR]

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

4. Run the mPlane components:

- o Supervisor

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

- o Tstat proxy:

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

- o Run the Repository proxy:

```
$ ./scripts/mpcom --config ./mplane/components/tstat/conf/tstatrepository.conf
```

- o Run the reasoner (make sure you have set the MPLANE_RI variable)

```
$ python3 cachereasoner --config reasoner.conf
```

Step-by-step walkthrough

Warmup: activating the DB collection of content request timeseries

Once started, the mPlane proxy interface for Tstat should start exporting data (HTTP logs) to the repository interface. This will filter the HTTP requests based on a targeted kind of content (e.g., YouTube videos). Filtered content will be used to build request timeseries and which will be stored in the MongoDB database.

Observe: the content expected to be popular in the future and the accuracy of the prediction

The reasoner will periodically query the analysis modules to obtain the list of pieces of content which represent the best candidates to be cached in a CDN system based on their expected popularity.

This use-case shall run for a long period of time, and the demo will mainly demonstrate the feasibility of the approach and its viability using the mPlane protocol. For the sake of showcasing, the accuracy of the content popularity estimation environment will rely on the analysis of historical data we have collected in the past and pre-imported in the use-case repository.

<http://www.ict-mplane.eu/public/guidelines-estimating-content-and-service-popularity-network-optimization>



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

<http://www.ict-mplane.eu/public/guidelines-passive-content-curation-0>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Passive content curation](#)

Guidelines for Passive content curation

Requirements

This page details the requirements that are specific to the Passive Content Curation in addition to those expressed for the Reference demonstration environment ([link](#)).

Hardware list

- **Probe:** One dedicated machine running Tstat as passive traffic probe
- **Repository:** One dedicated machine for the repository importer, the analysis modules for the Passive Content Curation, and the website used to present the captured URLs.

Software list

- mPlane protocol reference implementation ([GitHub repository](#))
- mPlane framework for the use-case ([GitHub repository](#))

Probe

- Tstat - Passive network traffic collection and analysis ([mPlane page](#), [GitHub repository](#))

Repository and analysis modules

- WeBrowse - an online HTTP request processing module which extracts URLs clicked by users ([GitHub repository](#))

Reasoner

- Reasoner for the Passive Content Curation ([mPlane page](#))

Software dependencies

- Probe: Linux OS and Python3
- Repository: Linux OS, Python3, pymysql and schedule Python modules

Supplemental software dependencies

This software is needed for to enable the presentation module (a website) for the URLs captured by WeBrowse:

- Apache 2.4.7 or higher (possibly configured in "worker" mode, which requires fast-cgi and php5-fpm packages)
- MySQL 14.14 or higher
- PHP 5.5.9 or higher
- Urllib2 for Python3

Software installation

- Download and install Tstat; follow the instructions at [GitHub repository](#)
- Download and install the mPlane proxy for the repository, **tstatrepository.py** (customised for this specific use-case), and the reasoner, **reasoner**, from the [GitHub repository for this demo](#)

Run the software

- Run Tstat (check the [mPlane page](#) for details):

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

Run the mPlane software

Here we assume that [PROTOCOL_RI_DIR] is the folder where the GitHub repository of the mPlane protocol reference implementation has been cloned.

<http://www.ict-mplane.eu/public/guidelines-passive-content-curation-0>

The code for the use case is available on GitHub under [mPlane demo material](#).

Supervisor

The Python code for the supervisor is the one provided in the mPlane protocol RI repository, i.e., **scripts/mpsup**. It receives the specification from the reasoner and forwards them to the other mPlane components.

Repository and Analysis Module

tstatrepository.py integrates the capabilities of communicating with Tstat and the interface to the query the analysis module.

Reasoner

reasoner is the Python script for the reasoner. It starts the Passive Content Curation demo by activating the streaming exporter of HTTP logs generated by Tstat, and periodically queries the tstatrepository to obtain the list of the most popular content observed in a given period.

HowTo

1. Set the parameters in the files **supervisor.conf**, **tstatrepository.conf**, **reasoner.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. Run the mPlane components:

- Supervisor

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

- Tstat proxy:

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

- Run the Repository proxy:

```
$ ./scripts/mpcom --config ./mplane/components/tstat/conf/tstatrepository.conf
```

- Run the reasoner (make sure you have set the MPLANE_RI variable)

```
$ python3 reasoner --config reasoner.conf
```

Step-by-step walkthrough

Warmup: online extraction of user clicks from network traffic

Once started, the mPlane proxy interface for Tstat should start exporting data (HTTP logs) to the repository interface. This will filter the HTTP requests to extract those corresponding to actual clicks generated by users. The analysis modules built on top of the streaming importer will also classify the URLs depending on the kind of content the point to (e.g., videos or news).

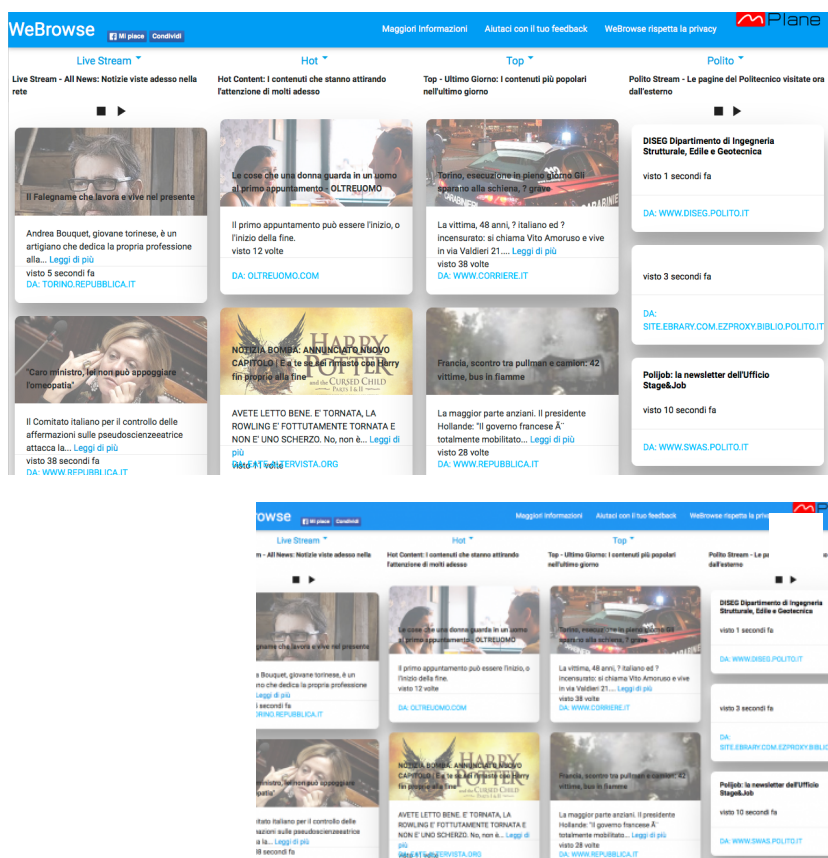
Observe: URLs observed in the network presented in a WeBrowse

The demo will demonstrate the feasibility of the passive content curation approach and its viability using the mPlane protocol. For the sake of showcasing, the URLs extracted by the WeBrowse modules will be presented in a website, similar to the one available in Polito deployment, <http://webbrowse.polito.it> (see the picture below). Notice that popular content exhibit as expected a strong locality bias, with italian newspaperes (having no english version) consistently showing up in the interface.

Trigger: visit webpages to have them presented in WeBrowse

During the demo we will emulate the behaviour of a user in the network by generating automatic visits to a list of news webpages. These will hence appear in the front-page of WeBrowse.

<http://www.ict-mplane.eu/public/guidelines-passive-content-curation-0>



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

<http://www.ict-mplane.eu/public/guidelines-active-measurements-multimedia-content-delivery>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Active measurements for multimedia content delivery](#)

Guidelines for Active measurements for multimedia content delivery

Requirements

This page details the requirements that are specific to the Active Measurements for Multimedia Content Delivery use case, in addition to those expressed for the Reference demonstration environment ([link](#)).

Hardware list

- **Probe:** dedicated machine to run the OTT-probe and GLIMPSE probe. It can also host the EZRepo and the RC1 reasoner needed for this demo.
- **CDN server:** (preferably more than one) to deliver multimedia content
- **Miniprobe:** (hardware based OTT probes, optional) to actively monitor multimedia delivery. The miniprobes used in the use case demonstration are NETvisor's proprietary hardware appliances called MiniProbes. We suggest to use of the models M-180 or above (in the official UC demo we use M-180 and M-195 models).
- **Impairment device:** to emulate streaming errors like jittering, packet delays, noise.

Software list

- mPlane protocol reference implementation ([GitHub repository](#))
- mPlane framework for the use-case ([GitHub repository](#))

Probes

- **OTT-probe** ([mPlane page](#), [GitHub repository](#))
- **GLIMPSE probe** ([mPlane page](#), [GLIMPSE official project page](#))

Repositories

- **EZRepo** ([mPlane page](#), [GitHub repository](#))

Reasoner

- **RC1 reasoner** ([mPlane page](#), [GitHub repository](#))

Software dependencies

- Probes, EZRepo, RC1 reasoner: Linux OS and Python 3.x

Software installation

- Download and install OTT-probe; follow the instructions at [GitHub repository](#)
- Download and install GLIMPSE; follow the instructions at [GLIMPSE official project page](#)
- Download and install EZRepo; follow the instructions at [GitHub repository](#)
- Download and install RC1 reasoner; follow the instructions at [GitHub repository](#)

Software configuration

- The demo environment specific configuration parameters like certificates, client listening links, etc should be set in the corresponding configuration files in the "conf/" directory (for the core components like supervisor) and in the "conf/mmcd/" directory (for the probes used in the demo like probes, EZrepo and RC1 reasoner).
- Probes must be configured to connect to the Supervisor and to send indirect measurement data to the EZRepo instance as well.

<http://www.ict-mplane.eu/public/guidelines-active-measurements-multimedia-content-delivery>

- Both content servers should have 4 VoD titles each (i.e. servers are alternative sources for same content).
- The topology of the network needs to be uploaded in the Reasoner (as a JSON file).
- The initial set of routine measurements needs to be configured in the Supervisor (as a JSON file).

Demonstration environment

- CDN servers for content hosting and streaming
- Impairment devices to emulate errors (packet delays, jitter, etc) in video streaming

Step-by-step walkthrough

Warmup: starting the monitoring infrastructure

1. Install and start up the components: the Supervisor (with GUI), the EZ_Repo and the RC1 Reasoner, possibly on a single machine. In any case, the Supervisor and the Repository need to have public IP addresses. Launch commands from separate windows, under the PYTHONPATH (~/protocol-ri) directory:

```
$ mplane/svgui --config conf/svgui.conf
```

```
$ scripts/mpcom --config conf/mmcd/ezrepo.conf
```

```
$ scripts/mpcom --config conf/mmcd/rc1.conf
```

In the supervisor's terminal window we expect to see the intro and the |mplane| prompt. By issuing "listcap" command we can check if repository and reasoner has been registered and connected to the supervisor. The GUI shall be accessible via the <supervisor>:<gui_port> address (default <gui_port> is 9892).

2. Deploy mPlane OTT probes, GLIMPSE probes and Pinger probes to multiple subscriber locations. Probes are implemented in Python and install packages will be created for Linux, Mac and Windows. In the following command we use a unified probe, with GLIMPSE, OTT-probe and Pinger installed.

```
$ scripts/mpcom --config conf/mmcd/common_probe.conf
```

We should see if the probes are up and running by issuing the "listcap" command from the prompt.

3. Probes are also available as hardware devices, deployed on the Miniprobe platform (TODO reference).

Trigger & observe: "Houston, we've had a problem here"

- Error scenario #1: remove a piece of content from both servers (to emulate "upstream/ingress error").

```
$ mmcd/errgen_rmcontent.sh
```

Reasoner shall correctly identify "upstream error".

- Error scenario #2: Shutdown one of the content server (keeping the machine running).

```
$ mmcd/errgen_serverdown.sh
```

Reasoner should correctly identify "CDN server X error".

- Error scenario #3: Configure a bandwidth limitation of about 500 kbps on one of the customer access lines.

```
$ mmcd/errgen_limitbandwidth.sh
```

Reasoner should correctly identify "Inadequate CPE bandwidth for Customer Y".



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.



<http://www.ict-mplane.eu/public/guidelines-active-measurements-multimedia-content-delivery>

Privacy

<http://www.ict-mplane.eu/public/guidelines-quality-experience-web-browsing-0>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Quality of Experience for web browsing](#)

Guidelines for Quality of Experience for web browsing

Requirements

This page details UC-specific requirements in addition to those detailed expressed for the Reference demonstration environment.

Hardware list

- A Linux PC (for running the **probe** and the **reasoner**)
- A set of servers (at least 3) for running the repository.

Software list

- A **OpenStack** - **Sahara** cluster
- The Firelog **probe** and the Web QoE **reasoner**

Components

- Firelog **probe**

Repositories

- **OpenStack** - **Sahara** cluster, with the configuration steps provided here

Reasoner

- The Web QoE **reasoner**.

Software dependency

In order to run the use case the following software is needed.

- Software:
 - Python ($\geq 3.3.x$)
 - java runtime environment (≥ 1.7) (JAVA_HOME set)
 - **Apache Flume**
- Linux packages
 - for the probe:
 - dh-autoreconf
 - python-numpy
 - sqlite3
 - for the repository:
 - python-psycpg2

Root access is needed to compile and run Tstat at the probe side.

Software installation

- **Probe:**
 - Follow the instructions provided on the **probe home page**
 - For Ubuntu Linux a **install.sh** is provided. On others Linux systems follow the instruction in the "usage of standalone probe" subsection on the **probe home page**

<http://www.ict-mplane.eu/public/guidelines-quality-experience-web-browsing-0>

Repository:

- Pre-requisites: A OpenStack-Sahara cluster
 - Navigate to the OpenStack web interface
 - Go to: Compute/Access & Security: Create Key Pair
 - Go to: Data Processing/Plugins: verify if Apache Spark plugin is installed, if not install it
 - Go to: Compute/Images:
 - Create Image using [this image](#)
 - Check the "Public" checkbox
 - Go to: Data Processing/Image Registry
 - User name: ubuntu
 - Plugin Spark 1.5 -> add plugin tags
 - Go to: Data Processing/Node Group Template
 - Create template master (master + namenode)
 - Create template worker (slave + datanode)
 - Go to: Data Processing/Cluster Template
 - Create cluster
 - Assign 1 master + 3 slaves
 - Launch cluster (use generated authentication)
 - On each cluster machine:
 - `sudo apt-get install libpq-dev`
 - download the DB for retrieving data from HDFS from [here](#), unpack.
 - All the software is now ready to be configured.
- #### Reasoner:
- The [web qoe reasoner](#) is based on the mpcli script, so it will run as any other component in the mPlane framework.

Software configuration

Components

- Change directory to [PROTOCOL_RI_DIR]/mplane/components/phantomprobe
- edit conf/firelog.conf: add username, modify paths of the local tstat, flume and phantomjs binaries
- edit conf/flume.conf for the sink ip/port
- edit conf/firelog-tstat.conf specifying ip/subnet to sniff from (e.g., 192.168.13.0 / 255.255.255.0)

Repositories

- On **all** nodes edit /etc/hadoop/conf/hdfs-site.xml


```
<property>
<name>dfs.datanode.data.dir.perm</name>
<value>755</value>
</property>
```
- `sudo service hadoop-hdfs-datanode (hadoop-hdfs-namenode) restart`
- On the **master node**:
 - Edit `dinodb/config/stado.config`

```
xdb.nodecount (number of worker nodes)
xdb.node.k.dbhost (k being the sequence number)
xdb.node.k.dbport
```

<http://www.ict-mplane.eu/public/guidelines-quality-experience-web-browsing-0>

○ **On all nodes:**

- Edit metastore.conf
 - metastore.hdfs.namenode -> namenode of HDFS (ipaddr:port) # Isot -i (default 50070)
 - metastore.hdfs.datanode -> datanode of HDFS (separated by ',')
 - metastore.hdfs.dir -> the path of datanodes' data directory (e.g., /dfs/dn/current, which MUST have read permission)
 - metastore.datanode.port: 8888
 - postgresraw.path -> the path of DiNoDB node
 - postgresraw.num: 1
- add Add \$DiNoDBnode/bin to PATH
- cd \$dinodbnode; bin/pg_ctl start -D datadir1
- on all worker nodes: cd \$metastore; nohup python dinodbnode.py &
- on master node: cd \$stado/bin; ./gs-server.sh
- on master node: Use gs-createdb.sh or createtable.sh to create the schema

Reasoner

- Make sure that params in webqoe/extract.py point to the master node on the repository

Demonstration environment

- Browsing session with no impairments
- Browsing session with impairments

Step-by-step walkthrough

Warmup - Setting up the QoE Use case: first browsing session

- Download and install the probe
- Register the probe to the supervisor
- Execute:

```
runcap firelog-diagnose
```

```
when: now + 1s
```

```
destination.url = www-selected-url
```

- when done:

```
show-meas firelog-diagnose-0
```

- No error should be reported.

Trigger:

- In order to raise errors, impairments should be put on the path between the probe and the web server. The easier way is to use tools like **netem** on a proxy machine (e.g., the gateway)

Observe:

- Re-running the same measurements in presence of impairments will highlight the root cause identified by the diagnosis algorithm on the probe side
- Executing the reasoner:

```
export PYTHONPATH=.
```

```
python3 mplane/components/qoe_reasoner.py --config conf/firelog-reasoner.conf --url www-selected-url
```

will cause the diagnosis algorithm to be run on the data available on the repository, so to provide further details on the selected web site behaviour.

<http://www.ict-mplane.eu/public/guidelines-quality-experience-web-browsing-0>



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

<http://www.ict-mplane.eu/public/guidelines-mobile-network-performance-issue-cause-analysis>



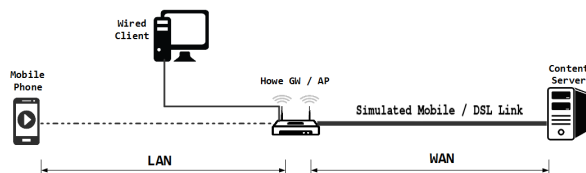
Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Mobile network performance issue cause analysis](#)

Guidelines for Mobile network performance issue cause analysis

Requirements



Hardware list

Mobile Phones

The mobile probe runs on *rooted* Android devices. It has been tested on Android 4.2.2 or later devices. Also the unmodified YouTube application may be used if we want to perform root cause analysis on YouTube videos.

Access Point/Router

The phones connect to the internet through a wireless access point, this should be a linux-based access-point where root access is possible. In our setup we use Netgear WNDR3800 running openWRT OS.

Media/Content Server

Our approach can work with both YouTube and custom video servers. In case of a customised video server a linux-based machine that supports Apache/2.4.7 or higher is required.

Repository

A linux-based machine that supports running mongoDB and the mPlane reference implementation is adequate. Enough storage should be provided to store the measurements (depending on the demonstration size). The Repository can be co-hosted with the media-server.

Reasoner

A linux-based machine that supports running the mPlane reference implementation and Java (Weka) is adequate. The Reasoner can be co-hosted with the Repository and/or the media-server.

Visualization

A linux-based machine that supports a web-server is adequate. The visualization machine can be co-hosted with the other servers above.

Software list

Components

- Mobile Probe (<https://www.ict-mplane.eu/public/mobile-probe-android>)
- Router/Server probe ([router_probe.tar.gz](#), [server_probe.tar.gz](#))

Repositories

- Mongo-db (<https://www.ict-mplane.eu/public/mongo-db>)

Reasoner

- Mobile Network RCA Reasoner (<https://www.ict-mplane.eu/public/mobile-network-rca-reasoner>)

Software dependency

- Mobile Probe: rooted Android device.
- Router Probe: AR71xx based router running OpenWRT with ip tool and Python 2.7 installed. If a router with different architecture is used, compatible Tsat and Python binaries should be provided.

<http://www.ict-mplane.eu/public/guidelines-mobile-network-performance-issue-cause-analysis>

- Server Probe: Linux OS with ip tool and Python 2.7 with pycurl library installed.
- All probes should have connectivity with the Repository.
- Video Server: Apache/2.4.7 or newer. All videos from <https://goo.gl/IMOlOZ> should be previously downloaded in the server's public directory.
- Repository: MongoDB and node.js with modules "express", "zlib", "util", "events" installed.
- Reasoner: The dependencies for the fault simulation component can be found in <https://www.ict-mplane.eu/public/mobile-network-rca-reasoner>. The machine learning component requires Java 1.7 or later and Weka 3.6.13.

Software installation

- Install the Mobile Probe following the instructions in <https://www.ict-mplane.eu/public/mobile-probe-android>.
- Install the Router Probe by extracting the router-probe.tar.gz in the router's local file system change to the directory with the extracted files and execute the provided install script (run.sh).
- Install the Server Probe by extracting the server-probe.tar.gz in the server's local file system, change to the directory with the extracted files and execute the provided install script (run.sh).
- The repository installation script ([repository_node.js](#)) needs to be run with the following command node repository_node.js.
- The reasoner script ([reasoner.py](#)) is controlled automatically and does not require installation.
- Install the reasoner as describd here (<https://www.ict-mplane.eu/public/mobile-network-rca-reasoner>)

Software configuration

- The global variable "REPO_URL" in the "parseAndPush.py" scripts which are included in the Mobile and Router Probes tarballs should be changed to point to the Repository URL or IP address.

Repositories

- specific configuration

Reasoner

- specific configuration

Step-by-step walkthrough

Warmup: Launching the probes

- The mobile device should be connected to the router's wireless network and a DSL or direct link to the video server should be available to provide access to the video content.
- Launch the Router/Server Probe as root by running ./run.sh in both devices. After the dependency checks have finished the probes will start to register and send hardware and network metrics to the repository.
- Launch the Mobile Probe and select the "Video Server URL" option from the menu. Enter the URL or the IP address of the video server including the trailing "/". If this is left empty, the default option is "<http://mplane.pdi.tid.es/youtube/>".

Trigger: Simulating a fault

- Launch the fault simulation script as explained in <https://www.ict-mplane.eu/public/mobile-network-rca-reasoner>.
- The Mobile Probe will automatically start a playback of a randomly selected video.
- When the video is finished stop the simulation script.

Observe: Video session QoE and root cause analysis


- The reasoner automatically evaluates the QoE of the new video session and illustrates the video timestamp along with the estimated QoE and the predicted impairment.
- The reasoner's output can be seen with an HTTP GET to "http://<reasoner.IP>/sessions"

File:

<http://www.ict-mplane.eu/public/guidelines-mobile-network-performance-issue-cause-analysis>

 [1269repositroyreasoner.tar.gz](#)

 [1270serverprobe.tar.gz](#)

 [1271routerprobe.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](#)

<http://www.ict-mplane.eu/public/guidelines-anomaly-detection-and-root-cause-analysis-large-scale-networks>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Anomaly detection and root cause analysis in large-scale networks](#)

Guidelines for Anomaly detection and root cause analysis in large-scale networks

Requirements

This page details UC-specific requirements in addition to those expressed for the Reference demonstration environment ([link](#))

Notice that the interaction between the different components listed in the present page, is schematically illustrated in the corresponding use case description webpage ([link](#)). This page assumes knowledge of these interactions, and merely expresses guidelines to setup and start each component.

Hardware list

- One dedicated machine running Tstat as passive traffic probe
- One dedicated machine for the repository running DBStream
- One machine running DidNETPerf (non necessarily dedicated, can be the same machine running the use-case reasoner)

Software list

- mPlane framework for the use-case ([GitHub repository](#)), containing the mPlane Supervisor, the Reasoner (mpAD_Reasoner) and the proxy for ADTool.

Components

- Tstat - Passive network traffic collection and analysis ([mPlane page](#), [GitHub repository](#))
- ADTool - Statistical Anomaly Detection Module, version 2.3 ([mPlane page](#))
- DisNETPerf - Network Proximity Service Module ([GitHub repository](#))
- mPlane Ripe Atlas proxy - proxy to distributed active measurement platform ([GitHub repository](#))

Repositories

- DBStream, including MATH data transfer protocol ([mPlane page](#), [GitHub repository](#))

Reasoner

- mpAD_Reasoner ([mPlane page](#))

Software installation

- Download and install the Tstat; follow instructions at [GitHub repository](#)
- Download and install DBStream; follow instructions at [GitHub repository](#)
- Download and install the ADTool analysis module; follow instructions at the corresponding [mPlane page](#)
- Download and install the mPlane framework for the use-case, which already provides the mPlane proxy to the ADTool at [GitHub repository](#)
- Download and install the mPlane RIPE Atlas proxy following the instructions at [GitHub repository](#)
- Download and install DisNETPerf following the instructions at [GitHub repository](#)

Software configuration

- Run the mPlane Supervisor:

<http://www.ict-mplane.eu/public/guidelines-anomaly-detection-and-root-cause-analysis-large-scale-networks>

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

- Run the mPlane Client:

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

- Run the Tstat proxy:

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

- Run the Repository proxy:

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

Components

- Run the ADTool proxy:

```
./scripts/mpcom --config ./mplane/components/ADTool/conf/adtool.conf
```

- Run the RIPE Atlas proxy:

```
./scripts/mpcom --config ./mplane/components/ripe-atlas/conf/component.conf
```

Repositories

- Run both DBStream and the MATH importer module, **math_repo**:

```
./hydra --config sc_tstat.xml
```

```
./math_repo
```

- Run Tstat and the MATH exporter module, **math_probe**, using the mPlane Client shell:

```
|mplane| runcap tstat-log_tcp_complete-core
```

```
|when| = now + inf
```

```
|mplane| runcap tstat-exporter_log
```

```
repository.url = localhost:3000
```

Reasoner

- no specific configuration

Demonstration environment

- Import in DBStream external data provided by geo-localization services such as **MaxMind** and IP address analysis services such as the one provided by the **Team Cymru** community.

Step-by-step walkthrough

Warmup

The use-case is run by starting the **mpAD_Reasoner** which interacts with all the mPlane components through the mPlane Supervisor, using the mPlane RI protocol, and orchestrates all the tasks needed to automate the detection and diagnosis of anomalies occurring in the distribution of YouTube videos.

- run the mpAD_Reasoner

```
./scripts/mpadtoolreasoner --config ./conf/mpadclient.conf
```

This use-case shall run for several months collecting and analyzing YouTube measurements in the quest for anomalies. However, we cannot ensure the presence of major anomalous events in the YouTube video provisioning during the deployment period. For the sake of showcasing the detection and diagnosis procedure of a major event, we will rely on the analysis historical data where we have detected and investigated a major anomaly and have been pre-imported in the use-case repository.

<http://www.ict-mplane.eu/public/guidelines-anomaly-detection-and-root-cause-analysis-large-scale-networks>

Consequently, the use-case demo consists of two scenarios

Scenario 1: use-case proof-of-concept

This scenario is intended to showcase that the mPlane Anomaly Detection modules can effectively detect anomalous behaviors related to both QoS-based and QoE-based performance metrics, and help in the root cause analysis investigation. The demo is based on historical YouTube traces pre-loaded in DBStream.

Trigger

Once the mpAD_Reasoner is started the ADTool starts analysing the preloaded YouTube traces. After a few analysis rounds ADTool starts flagging anomalies related to YouTube users QoE degradation. Along with the anomaly, the tool returns to the reasoner the list of involved server IP addresses.

Observe

- The reasoner client displays the list of server IP address involved in the anomaly along with the affected traffic features.
- DisNETPerf is used to locate the RIPE Atlas probes closest to the servers involved in the anomaly.
- The reasoner triggers, via the mPlane RIPE Atlas proxy, traceroutes from the identified probes towards the passive vantage point where TStat is deployed.
- The reasoner combines results returned by RIPE Atlas and the historical passive measurements, to generate a report about the flagged YouTube server IP addresses.

Scenario 2: real-time correlation of passive and active measurements

This part of the demo is meant to showcase that by using the mPlane reasoner it is possible to orchestrate the live collection of passive and periodic active measurements, and to trigger on the fly further active measurements. DBStream stores and analyzes nearly in real-time all the collected measurements. This part of the demo uses live traffic from Tstat probe at PoliTo/Fastweb (passive), active measurements returned by DistNETPerf (periodic/continuous, based on RIPE Atlas), and the RIPE Atlas mPlane proxy instantiation (reactive, on-demand).

The mpAD_Reasoner instructs ADTool to run on DBStream tables containing live data collected from the Tstat probe. The demo work-flow for Scenario 2 resembles the one of Scenario 1, with the only difference that we have no guarantee that any anomaly is flagged at run-time.

Trigger

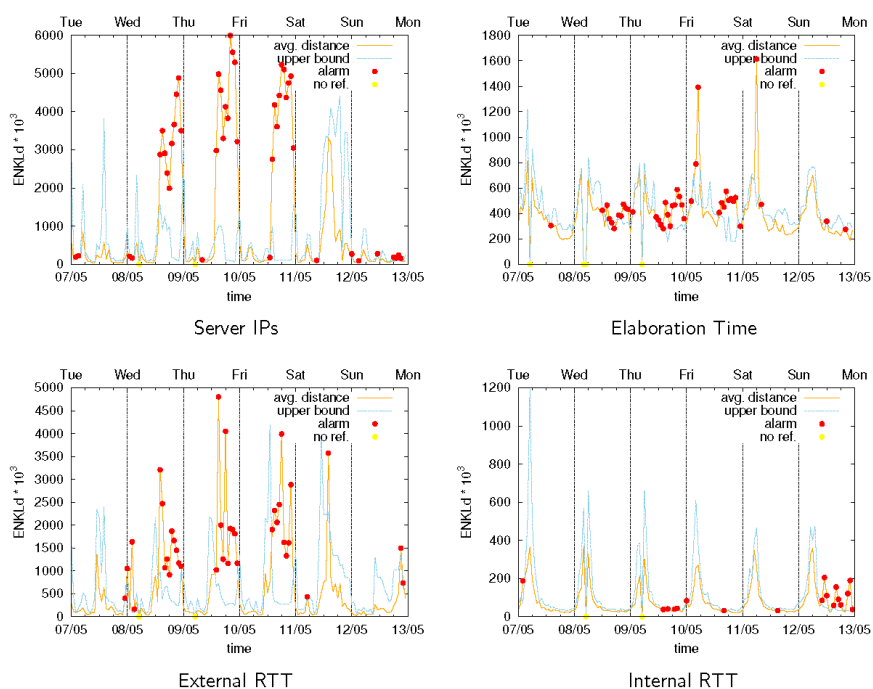
For demo purposes a subset of YouTube servers (e.g., those serving most of the traffic) are selected via the reasoner terminal to be further investigated through the RIPE Atlas active measurements.

Observe

Analysis results are reported in a similar manner of Scenario 1.

The pictures below show the typical output of the ADTool running on a number of traffic features. The detector outputs are then used by the reasoner to generate the anomaly report. For instance, a number of red dots in the picture refers to situations that can trigger alarms for anomalous situations detected by the reasoner.

<http://www.ict-mplane.eu/public/guidelines-anomaly-detection-and-root-cause-analysis-large-scale-networks>



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

<http://www.ict-mplane.eu/public/guidelines-verification-and-certification-service-level-agreement>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

[Home](#) > [Demonstration guidelines](#) > [Verification and Certification of Service Level Agreement](#)

Guidelines for Verification and Certification of Service Level Agreement

Requirements

This page details UC-specific requirements in addition to those detailed expressed for the Reference demonstration environment.

Hardware list

- Two Linux PC (for running the **probe** components server and client, the **reasoner** could be run on either of them.)

Software list

- The mSLAcert **probe** and the SLA **reasoner**.

Components

- mSLAcert **probe**

Repositories

- The data it is stored locally, no repository it is needed.

Reasoner

- The SLA **reasoner**.

Software dependency

In order to run the use case the following software is needed.

- Software:
 - Python (>=3.3.x)
- Linux packages
 - Additional libraries for RI
 - `sudo apt-get install python3-yaml`
 - `sudo apt-get install python3-urllib3`
 - `sudo apt-get install python3-tornado`
 - for the probe:
 - `sudo apt-get install iperf`
 - for the reasoner:
 - `expect - sudo apt-get install expect`
 - `convert - sudo apt-get install imagemagic`
 - `cupsfilter - sudo apt-get install cups`
 - `ps2pdf - sudo apt-get install ghostscript`

Executable permission are needed to run the reasoner.

Software installation

<http://www.ict-mplane.eu/public/guidelines-verification-and-certification-service-level-agreement>

- Probe:

- On a Ubuntu-like PC, follow the instructions provided on the [probe home page](#)

Software configuration

Before running the components and the RI you need to configure the `.conf` files at `./conf/` directory.

mPlane RI:

You need to configure mPlane client as following:

- `./conf/client.conf`

```
[TLS]
cert = PKI/ca/certs/"client-certificate".cert
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://"IP to where the spec are/IP Supervisor":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/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'
```

<http://www.ict-mplane.eu/public/guidelines-verification-and-certification-service-level-agreement>

```

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/

```

You will have also to configure a client that will be used by `./reasoner_msla.sh`

Components

- `./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

```

<http://www.ict-mplane.eu/public/guidelines-verification-and-certification-service-level-agreement>

```
# 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/
```

Repositories

- There are no repositories used for this usecase.

Reasoner

mSLAcert reasoner uses mPlane client, if it is different from the one configured, you need to add an additional link for it at the supervisor.

- *ipaddressdest.in-* each row contains the destination IP that have mSLAcert_Agent enabled.
- *ipsupervisor.in-* on this file is set the IP of the supervisor that the reasoner will use.
- *measnum.in-* on this file is set the default value is "0", please do not change this file.
- *timemeas.in-* on this file, measurement unit is set in seconds, the default value is 40 seconds.

Demonstration environment

To demo this use case, special hardware is needed to add impairments on the network (like modifying RTT, adding congestion, modifying jitter etc).

Step-by-step walkthrough

Before launching the components, the configuration files are needed to be configured as in "Software configuration" part.

To run the CI components (with SSL), from the protocol-ri directory, run:

To run CI mSLAcert server:

```
export PYTHONPATH=.
./scripts/mpcom --config ./conf/component.conf
```

To run CI mSLAcert Agent:

```
export PYTHONPATH=.
./scripts/mpcom --config ./conf/component-agent.conf
```

To run mPlane client:

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

To run mPlane Supervisor:

```
export PYTHONPATH=.
```

<http://www.ict-mplane.eu/public/guidelines-verification-and-certification-service-level-agreement>

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

This will launch the supervisor.

To run mSLAcert reasoner:

```
./reasoner_msla.sh
```

Warmup:

First, you need to make sure that all the component communicate with each other:

From the mPlane client connect to the supervisor and get capabilities:

```
|mplane|getcap https:"ipsupervisor":8890
```

After make sure all the capabilities are presented on the supervisor with:

```
|mplane|listcap
```

Trigger:

To see impact on TCP traffic of different network conditions, the following impairments could be used:

- High RTT ($\geq 100\text{ms}$): this can cause TCP congestion control to not saturate the capacity on high speed link
- High jitter ($\geq 30\%$ RTT): this can trigger false retransmission at TCP layer
- Link congestion: this would show TCP congestion control issues when dealing with not responsive traffic.
- Low SNR: this can create packet losses which are not related to congestion, thus reducing TCP throughput
- High BER: as above, packet lost due to physical impairment would trigger TCP congestion control despite no congestion is present.

Observe:

You can choose the capability you want to run with: "`|mplane| runcap cap-name`" command. To fully test SLA, run tests based on RTT, TCP and UDP measurements:

You could launch the SLA capability as:

```
|mplane|runcap msla-average-ip4
```

```
|when| = now + 40s / 1s
```

```
destination.ip4 = 192.168.1.2
```

```
source.ip4 = 1.2.3.4
```

```
ok
```

And get the results as:

```
|mplane| showmeas msla-average-ip4-0
```

Measurements could be performed and retrieved separately as well:

```
|mplane| runcap ping-average-ip4}
```

```
|mplane| runcap ping-detail-ip4
```

```
|mplane| runcap tcpsla-average-ip4
```

```
|mplane| runcap tcpsla-detail-ip4
```

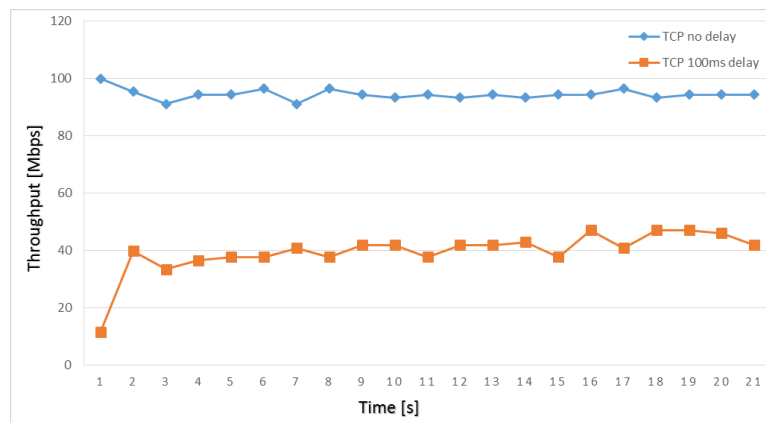
```
|mplane| runcap udpsla-average-ip4
```

<http://www.ict-mplane.eu/public/guidelines-verification-and-certification-service-level-agreement>

```
|mplane| runcap udpsla-detail-ip4
```

```
|mplane| showmeas [label-or-token]
```

In a normal case with no interference, there should not be significant difference between the throughput of TCP and the throughput of UDP on the client side. If there are impairments on the network we would see degradation of TCP throughput, or increase of RTT jitter, etc.. To analyse more in detail the nature of the impairment on the network, other probes/capabilities could be used. For example on high RTT, depending on you congestion algorithm you could see a difference of throughput as seen below:



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

<https://www.ict-mplane.eu/public/guidelines-path-transparency-measurements>



Registration for mPlane Final Workshop, Heidelberg, November 30, 2015

Building an Intelligent Measurement Plane f

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

[Home](#)

Guidelines for Path transparency measurements

[View](#)

[Edit](#)

[Revisions](#)

Public page *Guidelines for Path transparency measurements* has been updated.

[Clone content](#)

Requirements

This page details demonstration-specific requirements.

Introduction

The demonstration described here is a standalone demo, not integrated into the demonstration supervisor/repository, to illustrate the applicability of the mPlane protocol and architecture to Internet transport research. Pathspider consists of a custom mPlane client, which takes the role of client, reasoner, and supervisor all in one, and an mPlane component. It can be used in standalone mode (with client speaking to component locally), in service mode (component starts and waits for client-initiated connection to run measurements), and in client mode (connecting to previously started components). In addition to its own component and measurements Pathspider integrates the Scamper component for following up potential path-dependent connectivity issues by running traceroutes from each vantage point measurements are taken from.

Hardware list

The client/supervisor runs on any reasonably recent commodity PC (including laptops) that can run Python 3 in a UNIX or UNIX-like environment. The service-mode component runs on geographically and topologically separated virtual machines with good network connectivity without any ECN impairment (for this demo, five DigitalOcean VMs, `path-(sfo|nyc|lon|ams|sin).corvid.ch`).

Software list

Pathspider is available from <https://github.com/britram/pathtools>, and should be installed as described below on all service machines as well as on a central client from which the demonstration is run.

Software dependency

As Pathspider is installed as a single software installation including everything necessary for client, service, and standalone mode operation, all dependencies are required regardless of the intended mode of operation. The following software is necessary:

- mPlane SDK (<https://github.com/fp7mplane/protocol-ri>) and its dependencies. Since Pathspider uses multiple value support, the `sdk-multival` branch is as of this writing necessary, though mPlane SDK releases beyond 1.0.0 should include multiple value support.
- QoF (<https://github.com/britram/qof>) and its dependencies. The `develop` branch is necessary for ECN measurement features.
- Scamper and tracebox (<https://github.com/fp7mplane/components/blob/master/scamper/source/scamp...>) and their dependencies.
-

Software installation

To install pathspider for demonstration purposes, first create a Python 3.4 virtual environment:

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

<https://www.ict-mplane.eu/public/guidelines-path-transparency-measurements>

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
```

Software 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; fills the supervisor and reasoner roles.
- standalone : A standalone implementation where the measurements and analysis are performed on the same computer.
Cannot find path-dependent behavior.

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          # num of connection attempts in parallel (threads)
connection_timeout = 4      # timeout for a single connection attempt
interface_uri = ring:eth0   # libtrace uri, interface to listen on
```

<https://www.ict-mplane.eu/public/guidelines-path-transparency-measurements>

```
qof_port = 54739          # port for inter-process communication with QoF.
enable_ipv6 = true        # enable/disable ipv6 capabilities
ip4addr = 0.0.0.0         # bind measurement connections to this IPv4 address
ip6addr = ::              # bind measurement connections to this IPv4 address

[module_btdhtresolver]
module = pathspider.btdhtresolver
enable_ipv6 = true        # enable/disable ipv6 capabilities
ip4addr = 0.0.0.0         # bind resolver to this IPv4 address
ip6addr = ::              # bind resolver to this IPv4 address
port4 = 9881              # bind address collector to this IPv4 address
port6 = 9882              # bind address collector to this IPv6 address

[module_scammer]
module = pathspider.scammer.scammer
ip4addr = 1.2.3.4
ip6addr = ::1

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

Step-by-step walkthrough

Assuming that we have a set of service mode Pathspider instances configured as well as a client mode Pathspider instance configured to connect to them, as above:

- Start pathspider instances on each probe (from the appropriate path / Python venv)

```
pathspider --mode service
```

- Start the pathspider client and client gui on the client side:

```
pathspider --mode client --webui -v -B -c 100
```

- Browse to the pathspider gui (<https://localhost:1234>).
- Select a source for resolution ("IP address list") and enter a list of target IP addresses to test. For a good demonstration, you can use known-path dependent or otherwise interesting IP addresses, as available from previous testing, e.g. those available at <http://ecn.ethz.ch>.
- Start measurements. Targets with results will show up lower in the workflow when they are available; the color coding shows whether ECN is available (green), unavailable (red), or path-dependent (orange) on a given target.
- To examine the routes to a given target, click on it. This will initiate Tracebox measurements to the target. Targets with results will be shown when available. Click on a result to see a graph of paths to the target.



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.

