



mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Data Collection, Deployments and Assessment Results

Author(s):

TI (ed.)	F. Invernizzi, M. Ullio, M. D Ambrosio
FW	A. E. Kahveci, E. Kowallik, G. P. Mattellini, C. Meregalli, A. Sannino, M. Scarpino
POLITO	S. Traverso
FUB	E. Tego, F. Matera
ALBLF	Z. B. Houdi
EURECOM	M. Milanese
NEC	M. Ahmed
TID	I. Leontiadis, M. Varvello, L. Baltrunas
NETVISOR	B. Szabo, A. Bakay, L. Nemeth
FTW	P. Casas

Document Number:	D5.6
Revision:	1.0
Revision Date:	31 December 2015
Deliverable Type:	RTD
Due Date of Delivery:	31 December 2015
Actual Date of Delivery:	31 December 2015
Nature of the Deliverable:	(R)eport
Dissemination Level:	Public

Abstract:

This document is a deliverable of the mPlane Integration work package, WP5. It reports the results of the assessment and evaluation activities performed for each of the use cases defined in WP1. Links to web pages are preferred to simplify the sharing of information, and to avoid repetition of information.

Keywords: mPlane, integration, probe, supervisor, repository, reasoner, use case, evaluation, assessment, tests, tasks, verification, plans

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
Introduction.....	6
1 Results of Use Cases evaluation activity.....	7
1.1 Estimating content and service popularity for network optimization	7
1.1.1 Use case description and objective	7
1.1.2 Use case preliminary tests results	8
1.1.3 Use case assessment tests	10
1.2 Passive content curation	12
1.2.1 Use case description and objective	12
1.2.2 Use case preliminary tests results	12
1.2.3 Use case assessment tests	14
1.3 Active measurements for multimedia content delivery	16
1.3.1 Use case description and objective	16
1.3.2 Use case preliminary test results	18
1.3.3 Use case assessment test results	18
1.4 Quality of Experience for web browsing	20
1.4.1 Use case description and objective	20
1.4.2 Use case preliminary tests results	20
1.4.3 Use case assessment tests	21
1.5 Mobile network performance issue cause analysis.....	22
1.5.1 Use case description and objective	22
1.5.2 Use case assessment tests	24
1.6 Anomaly detection and root cause analysis in large-scale networks	25
1.6.1 Use case description and objective	25
1.6.2 Use case preliminary tests results	25
1.6.3 Use case assessment tests	28
1.7 Verification and Certification of Service Level Agreements	32
1.7.1 Use case description and objective	32
1.7.2 Use case preliminary tests results	34
1.7.3 Use case assessment tests	36
2 Experiments and data collection.....	38
2.1 Estimating content and service popularity for network optimization	38
2.1.1 Data and experiments.....	38
2.1.2 Evaluation activity and results	39
2.2 Web Content Promotion and Curation	40
2.2.1 Data and experiments.....	40
2.2.2 Evaluation activity results	41
2.3 Active measurements for multimedia content delivery	43
2.3.1 Data and experiments.....	43
2.3.2 Deployed architecture	43
2.3.3 Evaluation activity results	44
2.4 Quality of Experience for web browsing	48
2.4.1 Data and experiments.....	48

2.4.2	Deployed architecture	48
2.4.3	Evaluation activity results	49
2.5	Mobile network performance issue cause analysis	51
2.5.1	Data and experiments	51
2.5.2	Deployed architecture	51
2.5.3	Evaluation activity results	51
2.6	Anomaly detection and root cause analysis in large-scale networks	56
2.6.1	Data and experiments	56
2.6.2	Evaluation activity results	56
2.7	Verification and certification of service-level agreements	61
2.7.1	Data and experiments	61
2.7.2	Evaluation activity results	62
2.8	Publicly available datasets	64

Introduction

This deliverable reports the outcome of assessment and evaluation activities, including tests and experiments performed on each of the use cases defined in WP1.

This document is organized in two main parts:

- **Results of Use Cases evaluation activity**
- **Experiments and collected data analysis**

For a better understanding of the activities and tests performed, general information regarding the use cases is taken from Deliverable 5.5. In particular for each use case a short description and objectives are reported before showing final tests outcome. Links to previous documents and to web pages are preferred to simplify the sharing of information, and to avoid repetition of information.

Tests have been performed in partner premises and replicated in the FastWeb testplant when possible. In most cases, the actual deployment is up and running in real time, with a simple GUI that have been developed to showcase the results. All use cases have been successfully demonstrated during the final mPlane Workshop, held in Heidelberg, whose results are collected in D6.3 and D 7.6 in details. The same demonstration will be replicated during the final review meeting.

1 Results of Use Cases evaluation activity

This chapter contains the outcomes from the test performed on each use case in order to demonstrate the functionalities and applicability of mPlane architecture and protocol. For each use case, we briefly summarize its objectives, then report preliminary tests that have been conducted to demonstrate the interoperability of different components. Finally, assessment tests are described to summarize the results that have been collected in operational networks or in instrumented testbed.

Test results are summarized in tables, for both brevity and clarity.

1.1 Estimating content and service popularity for network optimization

1.1.1 Use case description and objective

The final goal of this use case aims at optimizing the QoE of the user and the load of the network by extracting the expected-to-be popular contents and identifying optimal objects to cache in a given portion of the network. To this end, we leverage the mPlane model to collect HTTP requests to pieces of content (e.g., videos) generated by users in the network. We then used such information to predict the content future popularity and select the best candidates for a proactive cache replacement strategy.

Targets to demonstrate

The demo of this use case shows how mPlane eases the deployment of an HTTP request collection system for Content Popularity Estimation. Currently this use case is being deployed in the campus network of Politecnico di Torino, and will be soon deployed in Fastweb premises too. Differently from what stated in past deliverables, we do not focus on YouTube traffic anymore, as HTTP transactions used to deliver videos are now fully encrypted and we cannot access the IDs of the videos being delivered. Hence, we employ Tstat probe continuously logs HTTP requests from network traffic, and use this data to feed the analysis modules for Content-URLs extraction developed for the Content Curation use case. Hence, for each observed URL, we store its popularity timeseries in a MongoDB repository. The analysis module then processes such data and estimates the future popularity of observed URLs.

Through the supervisor the demo will show how the reasoner orchestrates the deployment of this use case. First, the reasoner starts the exporting of the data from the probe to the repository. Second, it enables the importing of data and extraction of URLs at the repository. Finally, it queries the analysis modules to get the list of URLs (and the corresponding content) that are predicted to be popular in a given period of time.

Component list and versions

Component name	Role	Software
Tstat	passive probe	link
mPlane interface for Tstat	probe interface	link
Repository	repository and mPlane interface	link
Analysis module	analysis module and mPlane interface	link
Reasoner	reasoner	link

```

mPlane client shell (rev 20.1.2015, sdk branch)
Type help or ? to list commands. ^D to exit.

[implane] ok
Receipt repository-caching_performance-0 (token d9f48c031ec348c38b7487f5e99f6414): now + 1s
receipt: measure
  label      : repository-caching_performance-0
  token      : d9f48c031ec348c38b7487f5e99f6414
  when       : now + 1s
  registry   : http://tstat.polito.it/tstat-registry.json
  parameters ( 3):
    cache.bins: 100
    cache.algo: future
    cache.size: 1000
  File System
  metadata   ( 3):
    System ID: repository-Proxy
    System version: 0.1
    System_type: repository
  results    ( 2):
    time
    tstat.cache.videos

result: measure
  label      : repository-caching_performance-0
  token      : d9f48c031ec348c38b7487f5e99f6414
  when       : 2015-12-16 13:00:34.826785 ... 2015-12-16 13:00:39.190744
  registry   : http://tstat.polito.it/tstat-registry.json
  parameters ( 3):
    cache.bins: 100
    cache.algo: future
    cache.size: 1000
  metadata   ( 3):
    System ID: repository-Proxy
    System version: 0.1
    System_type: repository
  resultvalues( 1):
    result 0:
      time: 2015-12-16 13:00:39.190546
      tstat.cache.videos: 3237, 3631, 4319, 1841, 885, 3726, 2735, 1379, 225, 3743, 3455, 798, 38
75, 1474, 2897, 398, 2151, 3438, 3727, 793, 4193, 4209, 1075, 4314, 1286, 2301, 4280, 2838, 3200, 3771, 2340, 2173, 2
484, 775, 1372, 2778, 2148, 1800, 3885, 2278, 3888, 2220, 2678, 2464, 4170, 77, 1813, 1186, 1833, 2930, 2603, 514, 87
1, 2775, 2171, 4079, 3476, 2570, 1201, 2943, 1983, 1912, 1354, 1570, 2347, 2923, 1864, 650, 2536, 355, 350, 1793, 342
6, 2044, 2979, 1258, 3808, 1564, 4002, 2973, 2562, 1387, 1241, 3456, 997, 1627, 2739, 2089, 4344, 1112, 3992, 1923, 2
851, 3368, 2391, 542, 2774, 1819, 966, 1089, 4383, 401, 3128, 1202, 167, 3828, 1155, 4046, 113, 1674, 1099, 3858, 427
6, 1640, 484, 4040, 4444, 4345, 3567, 1712, 1859, 133, 320, 2103, 4137, 2995, 1047, 4018, 1754, 65, 4068, 1064, 746,
3690, 457, 2837, 812, 2438, 1581, 3611, 1990, 3036, 2316, 4096, 4207, 1582, 3452, 2312, 191, 2609, 47, 4290, 1638, 34
31, 2922, 126, 4377, 4425, 2367, 3488, 4489, 3911, 3865, 869, 4388, 2461, 2267, 4198, 559, 743, 3667, 2443, 3785, 136
7, 3183, 3198, 1654, 224, 2993, 2588, 145, 4224, 4409, 3095, 583, 4275, 1238, 2693, 3228, 2328, 1736, 1388, 686, 2782
, 385, 1499, 2449, 1525, 1799, 4482, 81, 1675, 1140, 2680, 4158, 1683, 2407, 4232, 3482, 1384, 1409, 383, 1933, 2492,
683, 2165, 2532, 2533, 1932, 3996, 108, 2512, 2551, 1687, 863, 621, 2441, 1839, 3189, 1045, 4263, 310, 880, 1756, 93
1, 1037, 87, 1333, 805, 4250, 2147, 1827, 708, 2878, 1995, 1082, 889, 3795, 1005, 107, 3152, 2199, 3513, 3986, 3569, 16
3307, 2733, 2583, 1062, 829, 2386, 1718, 30, 3392, 4205, 509, 2761, 2622, 2672, 2439, 2501, 4056, 517, 1648, 4256, 16
19, 3489, 3253, 2343, 1603, 2633, 2675, 3670, 3675, 733, 347, 3005, 1229, 665, 1329, 1909, 2079, 3201, 4317, 632, 158
, 1464, 4397, 254, 2791, 2201, 1044, 3114, 1445, 1007, 1250, 1955, 3111, 1601, 1259, 623, 633, 1717, 4184, 3546, 3188
, 2315, 1430, 1821, 1280, 173, 1897, 185, 4176, 677, 2475, 747, 2429, 1795, 3462, 952, 1732, 4443, 4337, 2889, 1366,
1116, 2285

```

Figure 1: Reasoner Output.

1.1.2 Use case preliminary tests results

Table 1 lists the initial tests that verify integration and deployment, and bootstrap the use case. Fig. 1, Fig. 2 and Fig. 3 illustrate some concrete outputs from the supervisor, reasoner and repository.

```
New Repository: 130.192.9.63:9800
Added <Service for <capability: measure (repository-collect_rrd) when past ... future token 0ff2cf28 schema ebc67c60
p/m/r 1/3/3>>
Added <Service for <capability: measure (repository-collect_streaming) when now ... future token 1ee4c5eb schema 107d
9ee7 p/m/r 1/3/0>>
Added <Service for <capability: measure (repository-collect_log) when past ... future token 3e12a916 schema 107d9ee7
p/m/r 1/3/0>>
Added <Service for <capability: measure (repository-caching_performance) when now ... future token 009a430e schema 5b
58f95 p/m/r 3/3/2>>
Listening...
Ibs ['local', 'polito-database']

Capability registration outcome:
callback: Ok
repository-collect_log: Ok
repository-collect_streaming: Ok
repository-collect_rrd: Ok
repository-caching_performance: Ok

Checking for Specifications...
Connection address: ('130.192.3.4', 44413)
----->scartato regex: http://www.msn.com/it-it/?ocid=wispr
----->scartato regex: http://www.msn.com/it-it/?ocid=wispr
Service for <capability: measure (repository-caching_performance) when now ... future token 009a430e schema 5ba58f95
p/m/r 3/3/2>> matches <specification: measure (repository-caching_performance-656) when now + 1s token 6bad42b1 sche
ma 5ba58f95 p(v)/m/r 3(3)/3/2>
Will interrupt <Job for <specification: measure (repository-caching_performance-656) when now + 1s token 6bad42b1 sch
ema 5ba58f95 p(v)/m/r 3(3)/3/2>> after 1.0 sec
Scheduling <Job for <specification: measure (repository-caching_performance-656) when now + 1s token 6bad42b1 schema
5ba58f95 p(v)/m/r 3(3)/3/2>> immediately
repository-caching_performance Enabled

Returning <receipt: (repository-caching_performance-656)6bad42b1094d5ebe0819a9223b5310e6>
repository-caching_performance-656 1000 future 100 AHHHH
100 OK
specification repository-caching_performance-656: start = 2015-12-16 13:00:34.826785 end = 2015-12-16 13:00:39.190744
repository-caching_performance Disabled
specification repository-caching_performance-656: start = 2015-12-16 13:00:34.826785, end = 2015-12-16 13:00:40.19331
```

Figure 2: Repository Output.

```
Returning <receipt: (repository-caching_performance-0)ff9d8f9d57fbd8fb4979ab353b9c158>
Specification repository-caching_performance-655 successfully pulled by org.mplane.polito.components.contpop-repo
Received exception for repository-caching_performance from org.mplane.polito.components.contpop-repo
Service for <capability: measure (repository-caching_performance) when now ... future token 009a430e schema 5ba58f95
p/m/r 3/3/2>> matches <specification: measure (repository-caching_performance-0) when now + 1s token d9f48c03 schema
5ba58f95 p(v)/m/r 3(3)/3/2>
Scheduling <Job for <specification: measure (repository-caching_performance-0) when now + 1s token d9f48c03 schema 5b
a58f95 p(v)/m/r 3(3)/3/2>> immediately
Returning <receipt: (repository-caching_performance-0)d9f48c031ec348c38b7487f5e99f6414>
Specification repository-caching_performance-656 successfully pulled by org.mplane.polito.components.contpop-repo
Received result for repository-caching_performance from org.mplane.polito.components.contpop-repo
```

Figure 3: Supervisor Output.

Table 1: Use case preliminary tests.

Test #1: Run Tstat			
Description	Command to execute	Command to check	Results
Run Tstat	<code>sudo ./tstat/tstat -l -i DEVNAME -s OUTPUTDIR</code>	Not Needed	Tstat is generating data in the OUTPUTDIR.
Test #2: Run MongoDB			
Description	Command to execute	Command to check	Results
Run MongoDB	<code>mongod</code>	<code>mongostat, mongotop</code>	MongoDB is up and running.
Test #3: Run the supervisor			
Description	Command to execute	Command to check	Results
Run supervisor	<code>./scripts/mpsup --config supervisor.conf</code>	Not Needed	The supervisor is listening on the correct port (e.g., the prompt must return "Listener http component running on 8890").
Test #4: Run the Tstat proxy			
Description	Command to execute	Command to check	Results
Check probe	<code>./scripts/mpcom --config tstat.conf</code>	Not needed	All Tstat proxy's capabilities are activated, e.g., the prompt must list the capabilities (tstat-log_http_complete, tstat-exporter_streaming, tstat-log_rrds, tstat-exporter_rrd, tstat-exporter_log).
Check supervisor		mplane listcap	Check that all above Tstat proxy's capabilities have been registered.
Test #5: Run the Tstat repository proxy			
Description	Command to execute	Command to check	Results
Check repository	<code>./scripts/mpcom --config tstatrepository.conf</code>	Not needed	All Tstat repository proxy's capabilities are activated, e.g., the prompt must list the capabilities (repository-collect_rrd, repository-collect_streaming, repository-collect_log, repository-caching_performance).
Check supervisor		mplane listcap	All Tstat proxy capabilities are registered.
Test #6: Run the reasoner			
Description	Command to execute	Command to check	Results
Run reasoner	<code>python3 reasoner --config reasoner.conf</code>	Not needed	All specifications are correctly sent.
Check supervisor		mplane listmeas	Check that all specifications are correctly received and delivered to proxies.
Check Tstat proxy		mplane showmeas tstat- exporter_streaming-0	The streaming indirect export is active and parameters are correct (URL, path, log type and log length in minutes).
Check queryable analysis module		mplane showmeas repository- caching_performance	The analysis module is queried and the list of candidate contents to cache are properly returned.

1.1.3 Use case assessment tests

Table 2 shows the test that verifies if the use case is on track to reach its objective.

Table 2: Use case assessment tests.

Test #1: Check the analysis module for the popularity estimation			
Description	Command to execute	Command to check	Results
Open an mPlane client and check the popularity analysis module is active by generating a specification manually	<code>./scripts/mpcli --config client.conf and run mplane runcap repository-caching_performance</code>	<code> mplane showmeas</code>	The analysis module correctly returns the list of the URLs expected to be popular in the specified period.

Test #1 can be viewed in Fig. 1.

1.2 Passive content curation

1.2.1 Use case description and objective

This use case shows how the mPlane architecture can provide a content curation service. Content curation is the act of helping users in finding relevant content in the web. It appeared as an answer to the overwhelming amount of content produced today on the Web. In this use case, we invented passive crowd sourcing for content curation; an approach that uses the crowd of users to discover relevant content without the need of user engagement. To do so, we assume that a click for a web page is a good measure of interest (since users often know what they are about to visit). We then leverage the collective clicks to infer relevant content. In this case, we use mPlane probes to export HTTP logs to repositories where intelligent analysis modules run in order to detect relevant web pages to recommend to users.

Targets to demonstrate

The demo of this use case will show first the feasibility of this use case, and second how mPlane can ease its deployment. The use case is already continuously running since several months in the campus network of Polito. The demo shows how the reasoner orchestrates the deployment of this use case including: (1) launching the data export from the probe to the repo, (2) data import by the repo and (3) the launch of the different analysis modules that continuously run to promote interesting web content to show to the users.

Component list and versions

Component name	Role	Software
Tstat	passive probe	link
mPlane interface for Tstat	probe interface	link
WeBrowse repository	repository and mPlane interface	link
WeBrowse online modules	analysis module and mPlane interface	soon in link
WeBrowse popularity module	analysis module and mPlane interface	link
WeBrowse reasoner	reasoner	link

1.2.2 Use case preliminary tests results

Table 3, taken from deliverable D5.5 lists the tests that verify integration and deployment, and bootstrap the use case. Fig 4, Fig 5 and Fig 6 illustrate some concrete outputs from the supervisor, reasoner and repository.

Figure 4: Reasoner Output.

Figure 5: Repository Output.

Figure 6: Supervisor Output.

Table 3: Use case preliminary tests.

Test #1: Run Tstat			
Description	Command to execute	Command to check	Results
Run Tstat	<code>sudo ./tstat/tstat -l -i DEVNAME -s OUTPUTDIR</code>	Not Needed	Tstat generate logs in OUTPUTDIR.
Test #2: Run the supervisor			
Description	Command to execute	Command to check	Results
Run supervisor	<code>./scripts/mpsup --config supervisor.conf</code>	Not Needed	The supervisor listens on the correct port.
Test #3: Run the Tstat proxy			
Description	Command to execute	Command to check	Results
Check probe	<code>./scripts/mpcom --config tstat.conf</code>	Not needed	Tstat proxy's capabilities are activated, the prompt lists the capabilities (tstat-log_http_complete, tstat-exporter_streaming, tstat-log_rrds, tstat-exporter_rrd, tstat-exporter_log).
Check supervisor		mplane listcap	The above Tstat proxy's capabilities are registered.
Test #4: Run the Tstat repository proxy			
Description	Command to execute	Command to check	Results
Check repository	<code>./scripts/mpcom --config tstatrepository.conf</code>	Not needed	Tstat repository proxy's capabilities are activated and the capabilities are listed (repository-collect_rrd, repository-collect_streaming, repository-collect_log, repository-top_popular_urls).
Check supervisor		mplane listcap	All Tstat proxy capabilities are registered.
Test #5: Run WeBrowse reasoner			
Description	Command to execute	Command to check	Results
Run reasoner	<code>python3 reasoner --config reasoner.conf</code>	Not needed	All specifications are sent.
Check supervisor		mplane listmeas	All specifications are received.
Check Tstat proxy		mplane showmeas tstat-exporter_streaming-0	The streaming indirect export is active with the correct parameters (URL, path, log type and log length).
Check queryable analysis module		mplane showmeas repository-top_popular_urls	The analysis module is queried and the list of popular URLs is properly returned.

1.2.3 Use case assessment tests

Table 4 lists the tests that verify if the use case is on track to reach its objective.

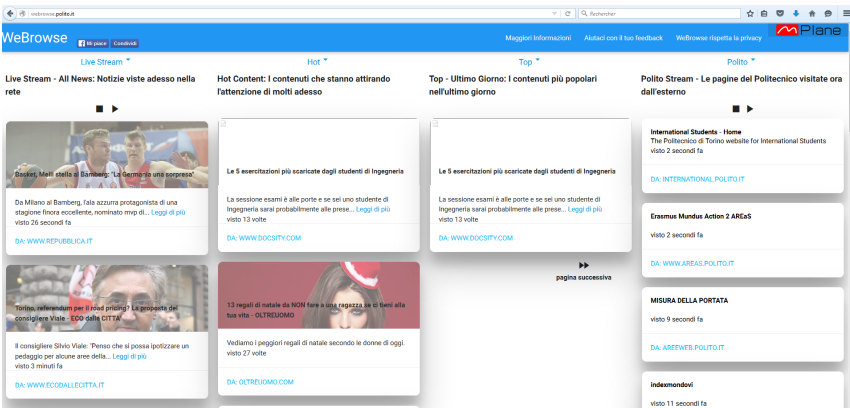


Figure 7: WeBrowse new website interface.

Table 4: Use case assessment tests.

Test #1: Check WeBrowse website			
Description	Command to execute	Command to check	Results
Check the WeBrowse website	Visits a few news webpages from a machine in the monitored network	Content must appear in WeBrowse website with a recent insertion time (seconds)	The live stream of URLs promoted by WeBrowse analysis module is active and dynamic.
Test #2: Check the popularity analysis module			
Description	Command to execute	Command to check	Results
Open an mPlane client and check the popularity analysis module is active by generating a specification manually	<code>./scripts/mpcli --config client.conf and run mplane runcap repository-top_popular_urls</code>	<code> mplane showmeas</code>	The analysis module must return the list of the most popular webpages for the specified period.

Test1 can be viewed in Fig. 4. Fig. 7 illustrates the new graphical web interface of the WeBrowse site.

1.3 Active measurements for multimedia content delivery

1.3.1 Use case description and objective

This use case demo presents mPlane components applied for a classic root-cause analysis function, i.e. the monitoring of content delivered over HTTP from a set of central servers, to a number of endpoints, most typically residential subscriber sites. The scheme is quite typical in today's internet considering the large proportion of video content consumed by mobile and stationary users.

The technical objective of this demo is to present a set of components that communicate with each other entirely over mPlane-standardized protocols. Probes, the repository and the reasoner all register with the supervisor, and almost all of the communication between those run through these registered connections (with the exception for indirect export of probe measurements to the repository, which is again an mPlane-suggested technique).

Besides the clean architecture this solution provides the capability to monitor the analysis and the resolution process from a single point, i.e. the Supervisor. In order to simplify user interaction the Supervisor GUI can be used, as demonstrated in this UC.

The Probes (OTT Probe, GLIMPSE and Pinger) and the Repository (EZ-Repo) have already been described in WP2 and WP3 documents, so this demonstration clearly focuses on the 'RC1' Reasoner, which is a central piece that queries and controls all other components and produces the final diagnosis messages. This reasoner is a rule-based reasoner which operates based on a priori topological information. It is capable of receiving external 'problem triggers', but more interestingly it also periodically queries the repository for typical 'problem patterns'. Whenever some problem occurs in either way, the reasoner is capable to start up additional measurements on probes to execute new, on-demand tests in order to be able to exclude or confirm some possible causes. E.g. whenever a residential probe cannot access some content, other probes are also directed to access the same or similar content (same content on another server, or other content on this server). At the same time, the original probe will also download other content from other machines. After these results are available the RC1 reasoner has much more data to diagnose whether the problem is at the client side, at the server, or maybe the content itself is not available.

To make the reasoning visible, the Reasoner is also integrated with the Dashboard capabilities of the SV GUI. The Dashboard allows any client to present data (measurements, states, distributions, etc.), on a configurable GUI surface, which includes various charts, dynamic tables, maps, etc.

This use case demonstration presents a "time-lapsed" view of a longer term evaluation currently in progress. During the demo, various errors are introduced with unusual frequency, and to cope with that the cycle times within the components have also been sped up 12 times (i.e. to 5 secs from 1 minute). The diagnosed events are thus happening quite fast during the demo, as shown by the time-line chart below (figure 8, taken from the Dashboard).

Targets to demonstrate

The main features presented during the demonstrations are as follows:

1. Use case proof-of-concept: the demo demonstrates that mPlane probes can indeed monitor service availability, and that they will indicate any disruptions in the service (as far as the affected clients, servers and content is included in the monitoring scope).

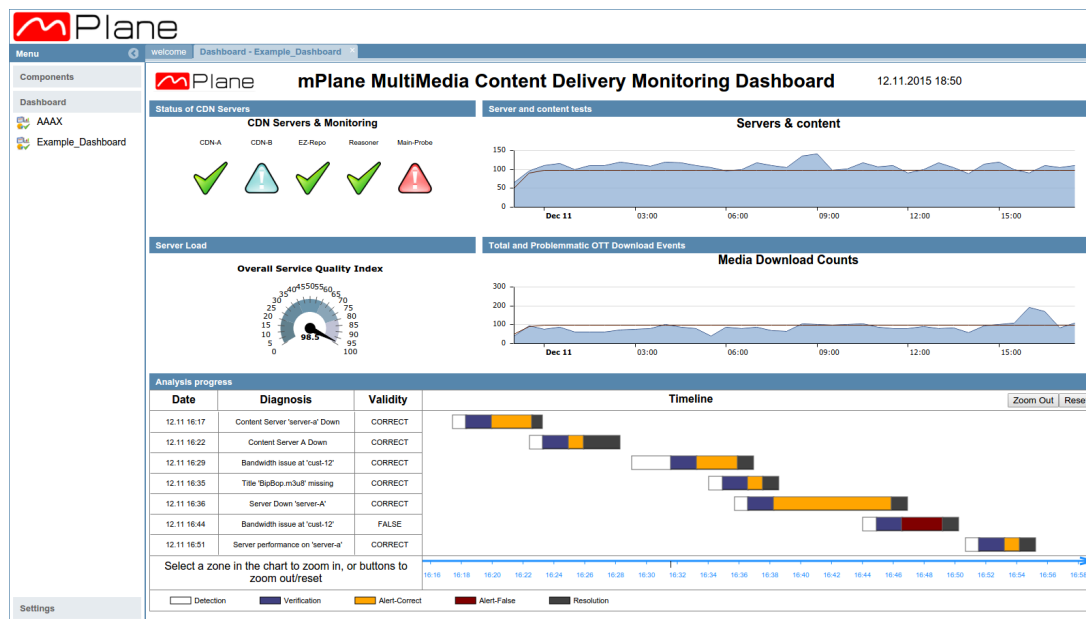


Figure 8: Dashboard visualization of multimedia content delivery diagnosis.

2. Root Cause Analysis: the intelligence built into the RC1 reasoner is capable of producing diagnoses with >90% accuracy, even with the current rudimentary ruleset, and with the relatively small number of probes available to the system.
3. Visualization: the capabilities presented on the Supervisor GUI and on the integrated Dashboard are efficient, generic-purpose visualization techniques to demonstrate the internal operation of the Root-cause analysis. The Dashboard itself is configurable enough to provide visual interface to a wide range of network/systems management applications.

Component list and versions used in the final demo

Component name	Role	Software
<i>OTT-Probe</i>	active probe for video tests	<i>D2.8 (2015-11-29) on GitHub</i>
Pinger probe	simple ICMP tester provided by the SDK	<i>mPlane RI v0.99 on GitHub</i>
<i>EZRepo</i>	generic performance data repository with built-in grading and search services	<i>D2.6 (2015-11-27) on GitHub</i>
<i>RC1 Reasoner</i>	root cause analyzing reasoner	<i>D5.5 (2015-11-28) on GitHub</i>
<i>mPlane supervisor</i>	the generic mPlane supervisor	<i>mPlane RI v0.99 on GitHub</i>
<i>Supervisor GUI</i>	the supervisor GUI	<i>D2.1 (2015-11-28) on GitHub</i>

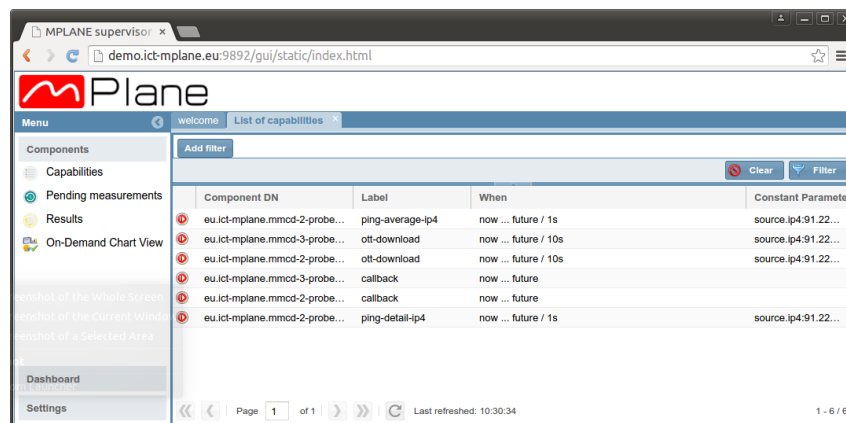


Figure 9: Probes registered in mPlane SV GUI (Test case #2).

1.3.2 Use case preliminary test results

Table 5 lists the initial tests that verify integration and deployment, and bootstrap the use case.

Table 5: Use case preliminary tests.

Test #1: Run the mPlane supervisor			
Description	Command to execute	Command to check	Results
Supervisor	<code>./scripts/svgui --config ./conf/mmcd/svgui.conf</code>	<code> mplane </code> prompt and access the GUI through browser	Supervisor and its GUI is up and running
Test #2: Run the probes			
Description	Command to execute	Command to check	Results
GLIMPSE OTT	<code>./scripts/mpcom --config ./conf/mmcd/common_probe.conf</code>	<code> mplane listcap</code> or check capabilities on GUI	Pinger, GLIMPSE and OTT capabilities are registered to supervisor, as demonstrated on SV GUI
Test #3: Run the repository			
Description	Command to execute	Command to check	Results
EZ Repo is registered	<code>./scripts/mpcom --config ./conf/mmcd/ez_repo.conf</code>	<code> mplane listcap</code>	EZ Repo capabilities registered to supervisor, checked on SV GUI
Test #4: Run the RC1 Reasoner			
Description	Command to execute	Command to check	Results
Register RC1	<code>./scripts/mpcom --config ./conf/mmcd/rc1.conf</code>	<code> mplane listcap</code> or access the GUI through browser	Reasoner and its GUI is available and accessible

1.3.3 Use case assessment test results

Table 6 lists the tests that verify if the use case is on track to reach its objective (see also figure 8 for a flow of some diagnosis test cases).

Table 6: Use case assessment tests.

Test #1 : Diagnosis of content missing from both servers			
Description	Command to execute	Command to check	Results
Rename the „BipBop.m3u8” of content to a temporary name in both servers (to emulate „upstream/ingress error”)	<code>mmcd/errgen_hidecontent.sh</code>	Checked on dashboard	Reasoner identified the error and reported as 'Title XXXXX missing' at 8 of 9 test cases within 252-384 seconds after the problem was emulated. After restoring the system alarms were revoked within 70-95 seconds.
Test #2 : Diagnosis of content server outage			
Description	Command to execute	Command to check	Results
Shut down one of the content server (keeping the machine running)	<code>mmcd/errgen_serverdown.sh</code>	Checked on dashboard	Reasoner identified the issue in all 7 test cases executed within 3 minutes (92 to 171 seconds), and reported 'Server Down server-XXX', after the resolution of the problem the Alarm was terminated in 55-96 seconds
Test #3 : Diagnosis of client-side bandwidth issues			
Description	Command to execute	Command to check	Results
Use netem to configure a general bandwidth limitation of about 500 kbps on one of the customer probes	<code>mmcd/errgen_limitbandwidth.sh</code>	Checked on dashboard	Reasoner identified the issue with 320-585 seconds, and reported 'Bandwidth issue at XXXXX' diagnoses at 11 out of 12 tests. The relative slowness of detection is justified by the test repetition rate configured on the MiniProbes (300 secs)
Test #4 : Diagnosis of server-side performance/network issues			
Description	Command to execute	Command to check	Results
Load Server-A with resource intensive processes so that system load is constantly in the 15-20 range. (Leave Server-B free of such load)	<code>mmcd/errgen_startupsort.sh -n 200</code>	Checked on dashboard	Reasoner identified 'Server performance on server-XXXX' in 6 of 8 tests. Clients reported download issues (mainly response time and refused requests but less on bandwidth), while the other server operated normally. Analysis time took 80-114 seconds, return to normal time was 119-510 seconds.

Note the assessment test results presented here are taken during a testing session between 5-20 December, 2015. Due to the limited timeframe only a simple test cases could be seen on the Heidelberg demo.

1.4 Quality of Experience for web browsing

1.4.1 Use case description and objective

The Web QoE Use Case aims at identifying the root causes of a high page load time in a browsing session. To do so, this use case is run via the Firelog probe, a hybrid probe capable of performing active and passive network measurements over a web browsing session. The collected data will serve as input to the diagnosis algorithm presented in deliverables D4.x to filter out the causes (if any) of a high page load time.

Targets to demonstrate

The use case is composed two live experiments.

1. Normal behaviour. A measurement session is run in absence of network impairments: the result will be that no problem is found.
2. Anomalous behaviour. A measurement session is run in presence of network impairments: the root cause will be identified.

An optional run from the reasoner will then gather data from the repository to show additional information on the browsed web site.

Component list and versions in the tests

Component name	Role	Software Release
Firelog probe	hybrid probe	0.1
WebQoE Reasoner	Reasoner	0.1

1.4.2 Use case preliminary tests results

Table 7 lists the initial tests that verify integration and deployment, and bootstrap the use case.

Table 7: Use case preliminary tests.

Test #1 : Run the supervisor			
Description Run the supervisor	Command to execute ./scripts/mpsup --config supervisor.conf	Command to check	Results The supervisor is listening on the correct port (e.g.- the prompt must return "Listener http component running on 8890)
Test #2 : Verify OpenStack Installation			
Description Check OpenStack management console	Command to execute Browse to http://[IP MASTER]/horizon	Command to check	Results The OpenStack cluster is up and running
Test #3 : Run the Firelog proxy			
Description Firelog registers its capability to the supervisor	Command to execute ./scripts/mpcom --config ./mplane/components/phantomprobe/conf/component.conf	Command to check mplane listcap	Results firelog-dagnose capability registered
Test #4 : Run the reasoner			
Description The reasoner runs the diagnosis algorithm on top of the repository data	Command to execute ./mplane/components/qoe_reasoner.py --config conf/client.conf --url selected.url	Command to check	Results Report on selected.url available

1.4.3 Use case assessment tests

Table 8 lists the tests that verify if the use case is on track to reach its objective.

Table 8: Use case assessment tests.

Test #1 : Probe measurement			
Description	Command to execute	Command to check	Results
Run the measurement and receive the measurements	mplane runcap firelog-diagnose	mplane showmeas firelog-diagnose-0	mplane listmeas
Test #2 : Find the root cause (with impairments)			
Description	Command to execute	Command to check	Results
Injection of additional RTT (e.g.) on the LAN	mplane runcap firelog-diagnose	mplane showmeas firelog-diagnose-0	e.g. {'result': 'Congestion on LAN/GW', 'details': 'Cusum on T1'}

1.5 Mobile network performance issue cause analysis

1.5.1 Use case description and objective

In this use case we use a combination of probes to perform Video Quality Root cause analysis (RCA) on mobile devices. Probes on the mobile device, the wireless gateway and the content server are used and upload data to the repository. For this purpose, a set of mobile devices (Android phones and tablets), WiFi access points and video servers were set up. An example is shown in Figure 10.

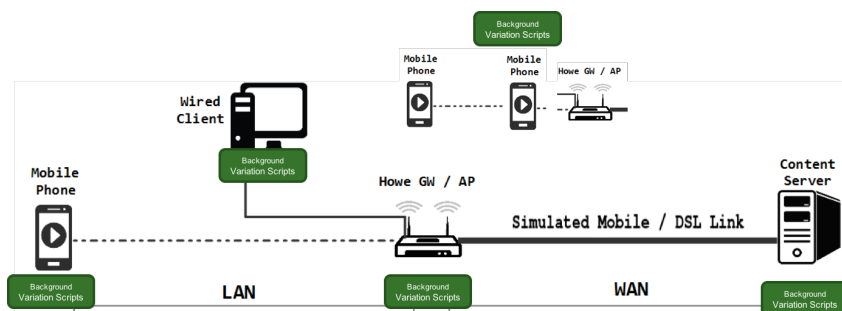


Figure 10: Mobile RCA demo architecture.

Finally, the reasoner uses the available information to analyze the video session and to estimate the quality of experience and the root cause of a problem. The last step is to visualize the result of the machine learning estimation as shown in Figure 11.

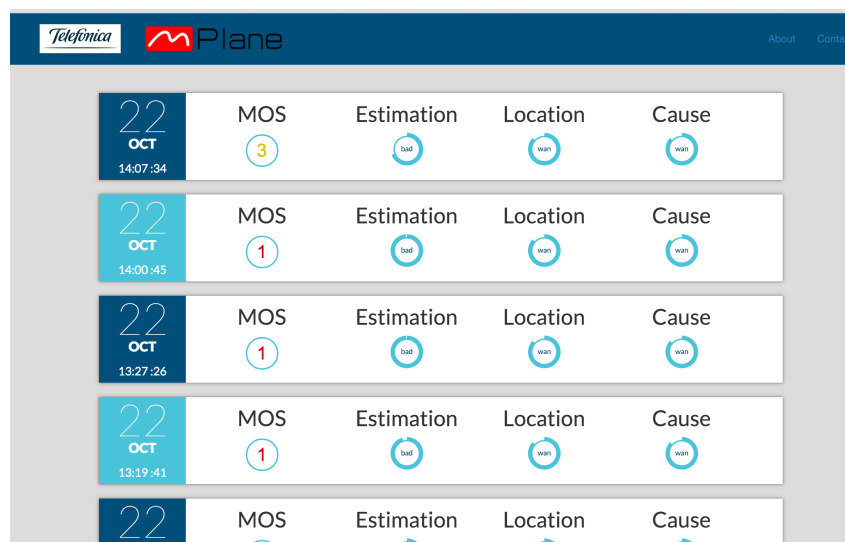


Figure 11: Mobile RCA demo visualization.

Targets to demonstrate

For showcasing the root cause analysis and diagnosis, a number of live experiments will be carried out.

1. "Normal": we will load one or more videos on a mobile device (e.g. a mobile phone) without any induced problems and show that no anomalies are detected.

2. “Problematic”: we will put impairments (i.e. wifi interference) and show how the mPlane probes and the diagnosis algorithm are able to correctly determine the root cause of poor video experience. The results will be given on a web page that will be running either locally or on a remote server.

Component list and versions in the tests

Component name	Role	Software
Probes	hybrid probe	link
Impairments	demo tools	link
mPlane mongo proxy	repository and interface	link
Reasoner	reasoner	link
mPlane framework	supervisor	link

Table 9 lists the initial tests that verify integration and deployment, and bootstrap the use case.

Table 9: Use case preliminary tests.

Test #2: Run MongoDB			
Description	Command to execute	Command to check	Results
Run MongoDB	mongod	mongostat, mongotop	MongoDB is up and running. See evidence below
Test #1 : Run the supervisor			
Description	Command to execute	Command to check	Results
Run the supervisor	./scripts/mpsup --config supervisor.conf		The supervisor is listening on the correct port (e.g.- the prompt must return "Listener http component running on 8890)
Test #2 : Run the mongoDB proxy			
Description	Command to execute	Command to check	Results
The mongo DB repository registers its capability to the supervisor	scripts/mpcom --config ./conf/component.conf Instructions here	mobileProbe capabilities registered	
Test #3 : Run the reasoner			
Description	Command to execute	Command to check	Results
The Reasoner runs the diagnosis algorithm on top of the repository data	Instructions here		Perform RCA on all video sessions that have not been classified before and store the results back into the repository

1.5.2 Use case assessment tests

Table 10 lists the tests that verify if the use case is on track to reach its objective.

Table 10: Use case assessment tests.

Test #1 : Measurements Without impairments			
Description	Command to execute	Command to check	Results
Run the measurement and receive the measurements	Launch a video with a mobile device that is instrumented. The probe installs an APP that can load random videos for convenience.	execute <code>runcap mobile-probe-rca-0</code> capability or launch the GUI	After finishing the video, the estimated video quality will be displayed and if there was a problem the route cause will be identified
Test #2 : Find the root cause (with impairments)			
Description	Command to execute	Command to check	Results
Injection of impairments (e.g., low RSSI)	Launch the impairment (<code>./experiment_controller.sh <fault></code>) More details here . Launch a video with a mobile device that is instrumented.	Execute the <code>mobile-probe-rca-0</code> capability or launch the GUI	After finishing the video, the cause should match the impairment

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

1.6.1 Use case description and objective

The main objective of the use case is to detect and diagnose large-scale anomalies in the provisioning of Internet scale services, i.e., services which are provided by omnipresent CDNs and which have a very large number of users worldwide distributed. Given its paramount role in current Internet as the leading service in terms of customers and traffic volume worldwide, the use case focuses on the detection and diagnosis of anomalies in the YouTube video provisioning system, specially targeting the Google CDN.

Targets to demonstrate

The specific targets to demonstrate through this use case are two-fold: (i) firstly, given the complexity of the monitored service (i.e., YouTube), the use case servers as show-case for the monitoring capabilities of the mPlane framework, in particular the usage of the reasoner to orchestrate the collection of passive and active measurements, and the triggering of new measurements on the fly, based on intermediate analysis results. The integration of multiple mPlane components (passive and active probes, repositories, analysis modules) as well as the integration of an external distributed measurement framework such as RIPE Atlas additionally shows the flexibility of mPlane to integrate existing large-scale measurement platforms. (ii) Secondly, the use case deployment shows 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.

Component list and versions

Component name	Role	Software
Tstat	passive probe	link
DBStream	repository	link
DisNETPerf	continuous/periodic active probe	link
RipeAtlas_proxy	on-demand active probe	link
ADTool	analysis module	version 2.3 link
mpAD_Reasoner	reasoner	link
UC specific proxies	probes	link

1.6.2 Use case preliminary tests results

Table 11 lists the initial tests that verify integration and deployment, and bootstrap the use case.

Table 11: Use case preliminary tests.

Test #1: Run the supervisor (or use the Public Supervisor)			
Description	Command to execute	Command to check	Results
Run supervisor	<code>./scripts/mpsup --config supervisor.conf</code>	Not Needed	The supervisor is listening on the correct port
Test #2: Run the Tstat proxy			
Description	Command to execute	Command to check	Results
Tstat registers log_tcp capabilities at the supervisor	<code>./scripts/mpcom --config ./mplane/components /tstat/conf/tstat.conf</code>	<code> mplane listcap</code>	log_tcp_complete capability registered at supervisor
Test #3: Run the repository proxy			
Description	Command to execute	Command to check	Results
DBStream is registered as Tstat repository	<code>./scripts/mpcom --config ./mplane/components /tstat/conf/tstatrepository.conf</code>	<code> mplane listcap</code>	DBStream capability registered to supervisor
Test #4: Run the ADTool proxy			
Description	Command to execute	Command to check	Results
Register ADTool	<code>./scripts/mpcom --config ./mplane/components /ADTool/conf /adtool.conf</code>	<code> mplane listcap</code>	ADTool registered as analysis module
Test #5: Run the RIPE Atlas proxy			
Description	Command to execute	Command to check	Results
Register RIPE Atlas proxy	<code>./scripts/mpcom --config ./mplane/components /ripe-atlas/conf /component.conf</code>	<code> mplane listcap</code>	RIPE Atlas registered as active probe
Test #6: Run DBStream and MATH importer module			
Description	Command to execute	Command to check	Results
Start DBStream and the math_repo module	<code>./hydra --config sc_tstat.xml</code>	Not Needed	DBStream and the data importer start
Test #7: Run Tstat and the MATH exporter module			
Description	Command to execute	Command to check	Results
Run Tstat and the math_probe module via the mPlane Client shell	<code> mplane runcap tstat-log_tcp _complete-core when =now+inf mplane runcap tstat-exporter_log repository.url =localhost:3000</code>	Not Needed	Tstat and data exporter start

In the specific integration tests, we have used the public mPlane Supervisor running at Fastweb premises for demo purposes (at <http://demo.ict-mplane.eu:9892>). The following figures present some snapshots of the aforementioned steps, executed as part of the use case bootstrapping. In particular, figure 12 shows the registration process of the anomaly detection analysis module (test #4) and the registration of the integrated RIPE Atlas capabilities (test #5) performed through the `mpcom` mPlane RI command, and figure 13 shows the corresponding registration results as depicted on the GUI of the public mPlane Supervisor. The verification of capabilities correctly registered at the public Supervisor can also be done from command line, using an instantiation of a simple mPlane client, through the `mpcli` mPlane RI command and the `listcap` functionality, as depicted in figure 14. Next we show the results obtained when running the corresponding Reasoner, which shall orchestrate the complete process.

```
pcasas@comanche-ubuntu: ~/mplane/supervisor_gui/protocol-ri
(mplane_demo)pcasas@comanche-ubuntu:~/mplane/supervisor_gui/protocol-ri$ ./scripts/mpcom --config
./mplane/components/ADTool/conf/adtool_public.conf
adpath mplane/components/ADTool/conf/ad_config.xml
adtool path mplane/components/ADTool/conf/ad_config.xml
Added <Service for <capability: measure (adtool-log_tcp_complete) when now ... future token 5e2e9a
2e schema e567b51c p/m/r 20/3/0>>

Capability registration outcome:
callback: ok
adtool-log_tcp_complete: Ok

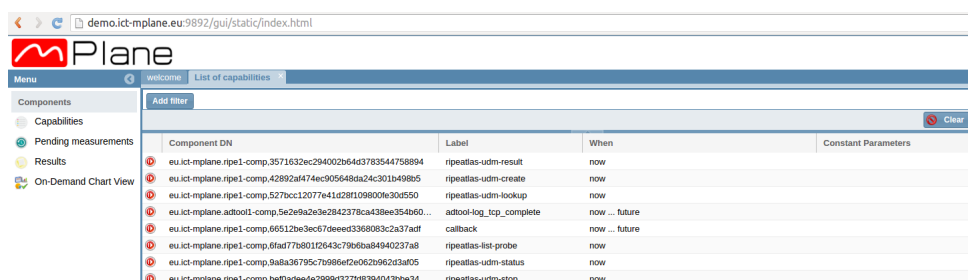
Checking for Specifications...
[ ]

pcasas@comanche-ubuntu: ~/mplane/supervisor_gui/protocol-ri
(mplane_demo)pcasas@comanche-ubuntu:~/mplane/supervisor_gui/protocol-ri$ ./scripts/mpcom --config ./mplane/component
s/ripe-atlas/component_public.conf
Added <Service for <capability: measure (ripeatlas-list-probe) when now token 6fad77b8 schema 9a025b39 p/m/r 1/0/1>>
Added <Service for <capability: measure (ripeatlas-udm-lookup) when now token 527bcc12 schema e55fb4b3 p/m/r 2/0/1>>
Added <Service for <capability: measure (ripeatlas-udm-create) when now token 42892af4 schema 7cb84e77 p/m/r 1/0/1>>
Added <Service for <capability: measure (ripeatlas-udm-status) when now token 9a8a3679 schema 1a8118d7 p/m/r 1/0/1>>
Added <Service for <capability: measure (ripeatlas-udm-result) when now token 3571632e schema 4caac66b p/m/r 1/0/1>>
Added <Service for <capability: measure (ripeatlas-udm-stop) when now token bef0adee schema 52c566a8 p/m/r 2/0/1>>

Capability registration outcome:
callback: ok
ripeatlas-udm-create: Ok
ripeatlas-udm-lookup: Ok
ripeatlas-udm-status: Ok
ripeatlas-udm-stop: Ok
ripeatlas-list-probe: Ok
ripeatlas-udm-result: Ok

Checking for Specifications...
[ ]
```

Figure 12: AD Tool and RIPE Atlas capabilities registration process.



Component DN	Label	When	Constant Parameters
eu.ict-mplane.ripe1-comp.3571632ec294002b64d3783544758894	ripeatlas-udm-result	now	
eu.ict-mplane.ripe1-comp.42892af474ec905648da24c301b498b5	ripeatlas-udm-create	now	
eu.ict-mplane.ripe1-comp.527bcc12077e41d28f109800fe30d550	ripeatlas-udm-lookup	now	
eu.ict-mplane.adtool1-comp.5e2e9a2e3e2842378ca438ee354b60...	adtool-log_tcp_complete	now ... future	
eu.ict-mplane.ripe1-comp.66512be3ec67deed3368083c2a37adf	callback	now ... future	
eu.ict-mplane.ripe1-comp.6fad77b80112643c79b6ba84940237a8	ripeatlas-list-probe	now	
eu.ict-mplane.ripe1-comp.9a8a36795c7b986e2e062b962d3a05	ripeatlas-udm-status	now	
eu.ict-mplane.ripe1-comp.bef0adee4e2999d327b8394043bbe34	ripeatlas-udm-stop	now	

Figure 13: AD Tool and RIPE Atlas capabilities, registered at the public mPlane Supervisor.

```
pcasas@comanche-ubuntu: ~/mplane/supervisor_gui/protocol-ri
(mplane_demo)pcasas@comanche-ubuntu:~/mplane/supervisor_gui/protocol-ri$ ./scripts/mpcli --config
./conf/client_public.conf
ok
mPlane client shell (rev 20.1.2015, sdk branch)
Type help or ? to list commands. ^D to exit.

|mplane| listcap
Capability ripeatlas-udm-result (token 3571632ec294002b64d3783544758894)
Capability ripeatlas-udm-create (token 42892af474ec905648da24c301b498b5)
Capability ripeatlas-udm-lookup (token 527bcc12077e41d28f109800fe30d550)
Capability adtool-log_tcp_complete (token 5e2e9a2e3e2842378ca438ee354b605b)
Capability ripeatlas-list-probe (token 6fad77b801f2643c79b6ba84940237a8)
Capability ripeatlas-udm-status (token 9a8a36795c7b986ef2e062b962d3af05)
Capability ripeatlas-udm-stop (token bef0adee4e2999d327fd8394043bbe34)
|mplane|
```

Figure 14: Verifying that capabilities are correctly registered through mpcli.

Table 12: Use case assessment tests.

Test #1: Run the mpAD_Reasoner			
Description	Command to execute	Command to check	Results
Start the use case	./scripts/mpadtoolreasoner --config ./conf/reasoner_public.conf	none	ADTool starts and the Reasoner console displays results, iteratively analyzing traffic for anomalies and issuing new active measurements through RIPE Atlas.

1.6.3 Use case assessment tests

The use case is run by starting the anomaly detection Reasoner, which interacts with all the mPlane components through the public 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. Table 12 shows the test that verifies if the use case is on track to reach its objective. In particular, it corresponds to the instantiation of the mpadtoolreasoner command, which consists of an extension of the standard mpcli mPlane client, following the design principles of the Reasoner as described in deliverables D4.2 and D4.4.

The subsequent steps which verify the correct functioning of the use case are reported next. (i) First, the Reasoner launches the anomaly detection analysis module, using pre-defined configuration parameters. Figures 15 and 16 depict the execution of the Reasoner and the subsequent instantiation of the Anomaly Detection analysis module respectively. The module is pre-configured to use the YouTube QoE-relevant KPI β as main monitoring variable, and three additional diagnostic signals (time series of average flow download throughput, flow min RTT, and downloaded YouTube traffic volume from YouTube IPs aggregated in /24 sub-networks) to shed initial hints on the detected anomalies, as we did in [?]. At this step, the Anomaly Detection analysis module runs on top of DBStream, continuously analyzing the traffic captured by Tstat and imported into the repository.

(ii) Second, an anomaly is detected by the Anomaly Detection module, and the Reasoners displays the corresponding output in the form of a diagnostic report, as depicted in figure 17. In particular, the anomaly is triggered by a shift in the empirical distribution of the β variable to lower values (e.g., there is an increase in the flows having $\beta < 1$ and a decrease in those having $\beta > 1.5$), suggesting an impact on the QoE of the YouTube flows as experienced by the monitored customers, see [3]. The report additionally indicates that there is also a decrease in the average download throughput, an increase in the


```
pcasas@comanche-ubuntu: ~/mplane/supervisor_gui/protocol-ri
(mplane_demo)pcasas@comanche-ubuntu:~/mplane/supervisor_gui/protocol-ri$ ./scripts/mpadtoolreasoner
--config ./conf/reasoner_public.conf
getcap URL: demo.ict-mplane.eu:9891/
Capability adtool-log_tcp_complete (token 5e2e9a2e3e2842378ca438ee354b605b)
Capability ripeatlas-list-probe (token 6fad77b801f2643c79b6ba84940237a8)
Capability ripeatlas-udm-create (token 42892af474ec905648da24c301b498b5)
Capability ripeatlas-udm-lookup (token 527bcc12077e41d28f109800fe30d550)
Capability ripeatlas-udm-result (token 3571632ec294002b64d3783544758894)
Capability ripeatlas-udm-status (token 9a8a36795c7b986ef2e062b962d3af05)
Capability ripeatlas-udm-stop (token bef0adee4e2999d327fd8394043bbe34)
ReasonerHttpListener running on port 11211
[DEBUG] We are running ADTool capability
Invoke capability with URL: https://demo.ict-mplane.eu:9891/
```

Figure 15: Running the Anomaly Detection use case Reasoner.

min RTT, and an increase in the number of YouTube subnetworks originating the video flows, suggesting that new YouTube servers located at farther locations are being now used, with a subsequent reduction of performance. The report finally displays some of the YouTube server IPs which originate the anomalous video flows, which shall then be used by the Reasoner to instantiate new measurements using the RIPE Atlas frameworks, integrated within the mPlane infrastructure.

(iii) Third, the Reasoner instantiates new active measurements from RIPE Atlas probes to find out if the reduction in the download throughput is caused by end-to-end path congestion in the downlink direction (i.e., from YouTube servers to customers), or on the contrary, caused by some other potential issue related to the YouTube servers and CDN. For doing so, the Reasoner launches firstly direct traceroute measurements using RIPE Atlas to the flagged YouTube IPs (173.194.18.23, 74.125.14.7 and 208.117.236.15), using as source a RIPE Atlas box geographically and topologically located close to the vantage point (we assume that the vantage point can not be used for issuing active measurements). Secondly, the Reasoner performs reverse traceroute measurements using the mPlane DisNETPerf analysis module, see [?], to measure the performance of the paths from YouTube servers towards the vantage point. As explained in [?], DisNETPerf is based on the very same RIPE Atlas framework, thus an additional set of RIPE Atlas measurements are instantiated by the Reasoner in the process. Figure 18 depicts a snapshot of the RIPE Atlas measurement GUI, which evidences the launching of the aforementioned active measurements.

```
pcasas@comanche-ubuntu: ~/mplane/supervisor_gui/protocol-ri

<Service for <capability: measure (adtool-log_tcp_complete) when now ... future token 5e2e9a2e sch
ema e567b51c p(m/r 20/3/0>> matches <specification: measure (adtool-log_tcp_complete-48) when now
+ 200d token 2649485f schema e567b51c p(v)/m/r 20(20)/3/0>
Will interrupt <Job for <specification: measure (adtool-log_tcp_complete-48) when now + 200d token
2649485f schema e567b51c p(v)/m/r 20(20)/3/0>> after 17280000.0 sec
Scheduling <Job for <specification: measure (adtool-log_tcp_complete-48) when now + 200d token 264
9485f schema e567b51c p(v)/m/r 20(20)/3/0>> immediately
Current ts: 1452500177.897769
>>generate XML configuration for ADTool Analysis Module<<   ts: 1452500177.897769
params: refset.min_distr_size
params: database.user
params: analysis.variable
params: analysis.end
params: refset.min_refset_size
params: database.password
params: mPlane.supervisor
Returning <receipt: (adtool-log_tcp_complete-48)2649485f07052b327b25f60282b76539>
params: database.features_table
params: refset.m
params: analysis.granularity
params: database.dbname
params: refset.slack_var
params: database.host
params: analysis.feature
params: refset.k
params: analysis.start
params: refset.width
params: database.flags_table
params: database.port
params: refset.guard

*****ADTool is Running*****
*****
** Symptomatic signal -- youtube_qoe_beta *****
*****
** Diagnostic signal 1 -- flow_down_th *****
** Diagnostic signal 2 -- flow_min_RTT *****
** Diagnostic signal 3 -- vol_down_24_subnet **
*****
*****
```

Figure 16: Execution of the Anomaly Detection analysis module.

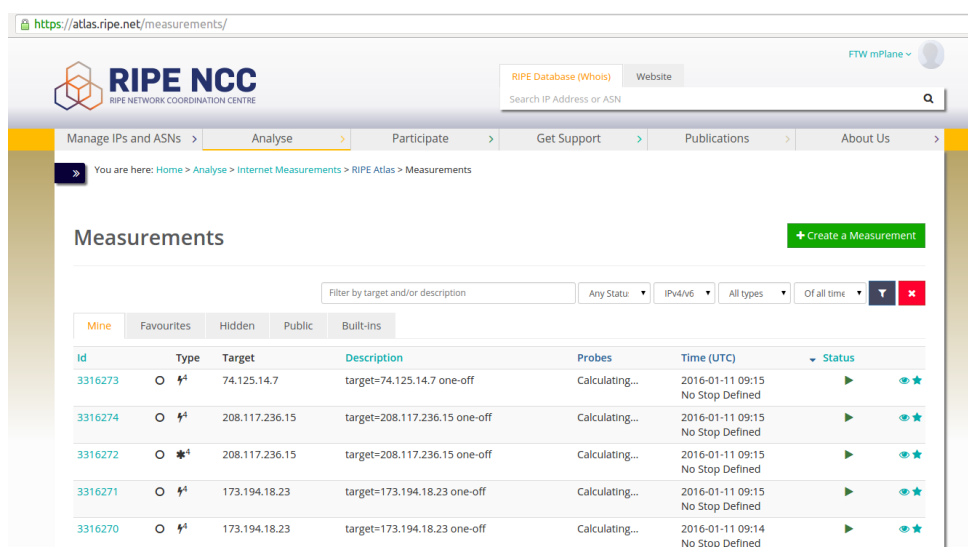
```
pcasas@comanche-ubuntu: ~/mplane/supervisor_gui/protocol-ri
(mplane_demo)pcasas@comanche-ubuntu:~/mplane/supervisor_gui/protocol-ri$ ./scripts/mpadtoolreasoner
--config ./conf/reasoner_public.conf
getcap URL: demo.ict-mplane.eu:9891/
Capability adtool-log_tcp_complete (token 5e2e9a2e3e2842378ca438ee354b605b)
Capability ripeatlas-list-probe (token 6fad77b801f2643c79b6ba84940237a8)
Capability ripeatlas-udm-create (token 42892af474ec905648da24c301b498b5)
Capability ripeatlas-udm-lookup (token 527bcc12077e41d28f109800fe30d550)
Capability ripeatlas-udm-result (token 3571632ec294002b64d3783544758894)
Capability ripeatlas-udm-status (token 9a8a36795c7b986ef2e062b962d3af05)
Capability ripeatlas-udm-stop (token bef0adee4e2999d327fd8394043bbe34)
ReasonerHttpListener running on port 11211
[DEBUG] We are running ADTool capability
Invoke capability with URL: https://demo.ict-mplane.eu:9891/
[DEBUG] Spec token: 20d01920b287377e00150debdad38afb
[DEBUG] Spec label: adtool-log_tcp_complete-0
[DEBUG] POST message received from ADTool

*****Anomaly Detected -- Diagnostic Report*****
*-----*
** symptom - youtube_qoe_beta: [>1.5,-][<1,+] *****
*-----*
** diagnosis - down_th_flow: [<300kbps,+] *****
** diagnosis - min_RTT_flow: [>60ms,+] *****
** diagnosis - vol_down_24_subnet: [<10,-][>10,+] *
*-----*
***** Top Server IPs involved *****
IP list: ['173.194.18.23', '74.125.14.7', '208.117.236.15']

We got address: 173.194.18.23
We got mode: traceroute
We can create ripe measurement
Invoke capability with URL: https://demo.ict-mplane.eu:9891/
Starting to do measurements for IP: 208.117.236.15...

We got address: 74.125.14.7
We got mode: traceroute
We can create ripe measurement
Invoke capability with URL: https://demo.ict-mplane.eu:9891/
```

Figure 17: Detection of an Anomaly in YouTube traffic and instantiation of RIPE Atlas active measurements, including DisNETPerf.



The screenshot shows the RIPE Atlas web interface at <https://atlas.ripe.net/measurements/>. The page displays a list of active measurements under the 'Measurements' section. The table below represents the data shown in the screenshot.

Id	Type	Target	Description	Probes	Time (UTC)	Status
3316273	O	74.125.14.7	target=74.125.14.7 one-off	Calculating...	2016-01-11 09:15 No Stop Defined	▶
3316274	O	208.117.236.15	target=208.117.236.15 one-off	Calculating...	2016-01-11 09:15 No Stop Defined	▶
3316272	O	208.117.236.15	target=208.117.236.15 one-off	Calculating...	2016-01-11 09:15 No Stop Defined	▶
3316271	O	173.194.18.23	target=173.194.18.23 one-off	Calculating...	2016-01-11 09:15 No Stop Defined	▶
3316270	O	173.194.18.23	target=173.194.18.23 one-off	Calculating...	2016-01-11 09:14 No Stop Defined	▶

Figure 18: Instantiation of active measurements on the fly, using the RIPE Atlas framework and the mPlane DisNETPerf analysis module.

1.7 Verification and Certification of Service Level Agreements

1.7.1 Use case description and objective

Aim of this Use Case is the verification and the certification of the SLA between ISP and client related to the line capacity. For such an aim a novel probe, mSLAcert, was designed and tested in the framework of MPLANE project that is able to measure some key network parameters that are: TCP throughput, UDP throughput, RTT, packet losses and jitter. The specific parameter that characterizes the bandwidth delivered by the ISP to the client is the UDP throughput, but the other parameters are important to analyze the network status. All the details on the mSLAcert probe are reported in <http://www.ict-mplane.eu/public/mslcert-active-probe>.

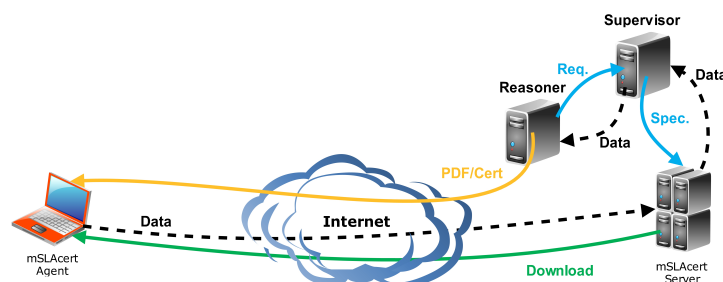


Figure 19: Schematics of SLA use case.

mSLAcert is composed of two components, a server and an agent. The measurement is based on RTT tests and TCP/UDP downloads from server to agent; at end of the download tests the agent sends a report back to the server, reporting the measured parameters. The reasoner can request additional tests for the SLA verification to the supervisor. The supervisor, sends the test specifications to the probe, which after completing the test it will send the data back to the supervisor, that could also be seen by the reasoner. After the result comparison carried out by the reasoner, a PDF paper will prepared to certify the delay and throughput measured by the agent. mSLAcert can be adopted both by the user to certify its access line and by the ISP to check the lines of its users.

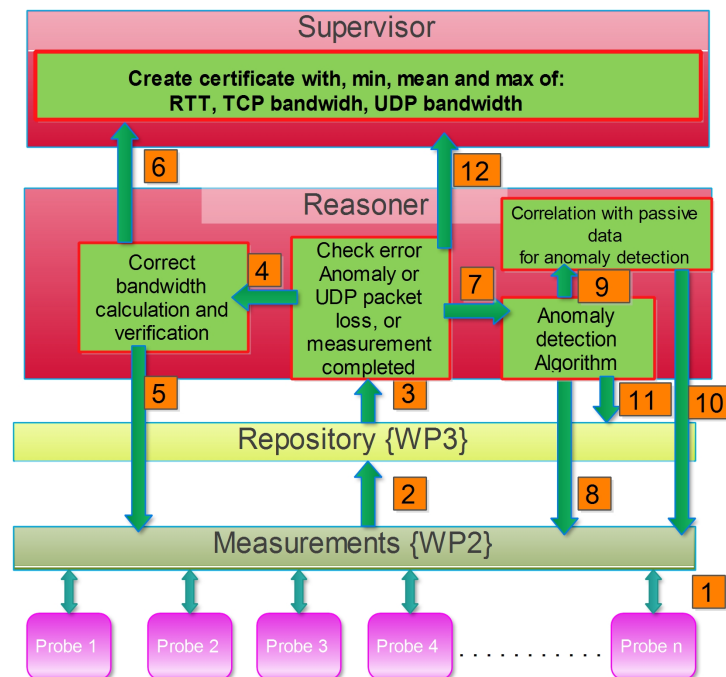


Figure 20: Work flow of SLA use case.

In the figure 20 the mPlane components involved in this use case are described. They are active probe (mSLAcert), the supervisor and the reasoner. The probe performs active measurements based on their specifications (step 1 of 20), and it tests RTT, throughput, jitter and datagram loss. After these tests data are stored on the repository, however for such a use case the amount of data is small, so data are stored locally (step 2-3). When the measurement is concluded the reasoner checks the data for possible problems (step 4,7,9). Based on the result of the reasoner, it can request new measurements from the probes, with different specifications (step 5,8,10) or request additional analysis from the repository (step 11). When the reasoner decides that all the measurements are correct it asks to the Supervisor to release a PDF that will certify the RTT and throughput of the client (step 6 and 12).

It has to be underlined that The mPlane code has changed a few times during the project duration. In fact, at the beginning, mSLAcert was a series of bash shell scripts that carried out SLA measurements in compliance with D2.1/D2.2. With the development of the first mplane RI (Reference Implementation) in Python v3 language by ETH, was required to change the code of the probe. The main issue, was adapting the protocol of mSLAcert and the use of IPERF to RI code. The code of the probe acted as a server and as a probe, there were no supervisor.

Second step of development was the integration with the supervisor, at the beginning developed by TI and SSB. Main issues were changing the structure of the code and the authentication with the supervisor. Usual problem were from the certificates and the roles on the configuration files of the Supervisor and probe. A problem that was encountered and it was due to the supervisor that did not accept HTTP; therefore we first solved it by modifying the supervisor, and currently this issue has been solved by SSB and ETH with suitable modifications of the supervisor. On the next changes, now, the code is more modular, so all the changes that are made to the supervisor do not impact the probes.

Component list and versions in the tests

Component name	Role	Software
mSLAcert Server	probe	<i>mSLAcert_main v. 3.0.4</i>
mSLAcert Agent	probe	<i>mSLAcert_Agent v. 1.0.0</i>
mSLA reasoner	reasoner	<i>reasoner_mSLA v.1.0.0</i>

1.7.2 Use case preliminary tests results

Here we report a summary of the all tests that were carried to verify the correct operations of mSLAcert. First of all the probe's capabilities were successfully registered to the supervisor first in FUB LAB (details of FUB LAB are described in D.5.2). mSLAcert probes were also installed in TelecomItalia and Fastweb. Furthermore this probe was included in the Virtual Machine prepared by Telecom Italia.

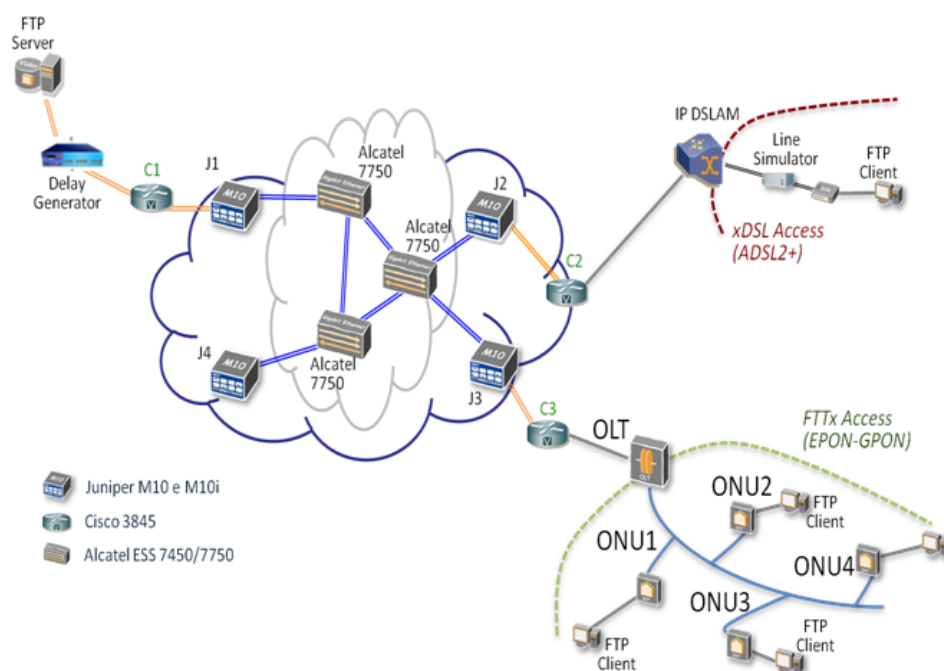


Figure 21: Schematics of the used test bed.

A wide LAB trial was carried out in the FUB Lab to verify mSLAcert, where different access architectures were tested in different network conditions. The Lab, illustrated in Fig. 21, was described in details in D.5.2. For sake of brevity here we report only the tests regarding the GPON configuration where the user capacity was 100 Mb/s, with different RTT obtained with a delay line. Furthermore other network impairment were added as congestion in the core part and packet losses. All the Tests planned in the Table 6 of D.5.4 were successfully carried out and here we report the main results following the same Test schemes illustrated in such a table.

Test 1: RTT test A delay was introduced in the network test bed between 0 and 100 ms and we measured the corresponding RTT. In all the tests the correspondence between delay and measured RTT was perfected, also by adding packet losses with probability lower than 10⁻⁶. RTT was also investigated by

introducing traffic congestion in the core network of the test bed reported in fig. 12 of D.5.4. In particular a traffic of 1Gb/s was introduced in a 1 GbE link between two routers causing packet forwarding in alternative links with consequent RTT increasing. We also repeated the tests reported in table 1 of [6] where min, max and average RTT were measured in the presence of multiple TCP and UDP fluxes in conditions of link congestion, confirming the behavior described in such a table.

Table 13 lists the initial tests that verify integration and deployment, and bootstrap the use case. In Figure 22 we show an example of the output of the probe integration and capability registration.

Table 13: Use case preliminary tests.

Test #1 : Launch supervisor			
Description	Command to execute	Command to check	Results
Execute supervisor	<code>./scripts/mpsup --config ./conf/supervisor.conf</code>	—	<i>ListenerHttpComponent running on port 8890</i>
Test #2 : Launch client			
Description	Command to execute	Command to check	Results
Execute mPlane Client	<code>./scripts/mpcli --config ./conf/client.conf</code>	—	<i>/mplane/</i>
Test #3 : Launch probe			
Description	Command to execute	Command to check	Results
Probe mSLAcert capabilities registration	<code>./scripts/mpcom --config ./conf/component.conf</code>	—	<i>callback: Ok *-*.ip4: Ok</i>
Test #4 : Integration of probe			
Description	Command to execute	Command to check	Results
Integration check	<code>getcap https://"IP_Spv":8890</code>	<i>/mplane/ listcap</i>	<i>Added <Service for <capability: measure *-*.ip4 when now ... future / 1s token</i>

Note: the '*' means that there are more than one capability for the probe. The capabilities are:msla-average-ip4; ping-detail-ip4; udpsla-detail-ip4; udpsla-average-ip4; tcpsla-average-ip4; ping-average-ip4; msla-detail-ip4; tcpsla-detail-ip4; msla-AGENT-Probe-ip4;

```
mSLAcert :~$ cd mSLAcert/protocol-ri.git/trunk/
mSLAcert :~/mSLAcert/protocol-ri.git/trunk$ export PYTHONPATH=.
mSLAcert :~/mSLAcert/protocol-ri.git/trunk$ ./scripts/mpcom --conf ./conf/component.conf
=====
Added <Service for <capability: measure (ping-average-ip4) when now ... future / 1s token c6fe84cc schema e2ca42e6 p/m/r 2/0/4>>
Added <Service for <capability: measure (ping-detail-ip4) when now ... future / 1s token e79f0363 schema db8ef547 p/m/r 2/0/2>>
Added <Service for <capability: measure (tcpsla-average-ip4) when now ... future / 1s token c3bab8fe schema f0db4d6e p/m/r 2/0/4>>
Added <Service for <capability: measure (tcpsla-detail-ip4) when now ... future / 1s token 234f7591 schema 321927e7 p/m/r 2/0/2>>
Added <Service for <capability: measure (udpsla-average-ip4) when now ... future / 1s token d92eb85d schema 7e0df173 p/m/r 2/0/7>>
Added <Service for <capability: measure (udpsla-detail-ip4) when now ... future / 1s token 61c84cdc schema 73d4f815 p/m/r 2/0/5>>
Added <Service for <capability: measure (msla-average-ip4) when now ... future / 1s token 9b20b332 schema f3746171 p/m/r 2/0/15>>
Added <Service for <capability: measure (msla-detail-ip4) when now ... future / 1s token 1e235cbd schema 5ff95292 p/m/r 2/0/7>>

Capability registration outcome:
udpsla-detail-ip4: Ok
msla-average-ip4: Ok
tcpsla-average-ip4: Ok
ping-detail-ip4: Ok
tcpsla-detail-ip4: Ok
ping-average-ip4: Ok
udpsla-average-ip4: Ok
callback: Ok
msla-detail-ip4: Ok

Checking for Specifications...
```

Figure 22: Launching and registration at the supervisor of mSLAcert probe.

Test 1-2-3: TCP, UDP throughput and comparison tests Tests were carried out in FUB LAB in GPON access over two days in different network conditions (RTT, loss, congestion). Comparison between TCP and UDP throughput clearly illustrate the advantages of our method for SLA verification and certification. The main results are summarized in table below.

Network conditions	Line capacity	RTT	TCP throughput	UDP throughput
GPON (PC windows7)	100 Mb/s	100 ms	42 Mb/s	95.6 Mb/s
GPON (windows7) with link congestion .	100 Mb/s	100 ms	Max 31 Mb/s Av 21 Mb/s Min 8 Mb/s	Max 95.6 Mb/s Av 68 Mb/s Min 48 Mb/s
GPON (Cubic) Test 10 s long	100 Mb/s	100 ms	Max 73 Mb/s Av 58 Mb/s Min 38 Mb/s * Max remains stable at 73 Mb/s after 12 s	95.6 Mb/s
GPON (New Reno) Test 14 s long	100 Mb/s	100ms	Max 54 Mb/s Av 52.5 Mb/s Min 51 Mb/s * Max reaches 73 Mb/s after 150 s	95.6 Mb/s
GPON (Cubic)	100Mb/s BER 10 ⁻⁶	0.4 ms	Max 90 Mb/s Av 86 Mb/s Min 78 Mb/s	95.6 Mb/s
GPON (Cubic)	100Mb/s BER 10 ⁻⁶	40 ms	Max 5.5 Mb/s Av 2.8 Mb/s Min 2.3 Mb/s	95.6 Mb/s
GPON (New Reno)	100Mb/s BER 10 ⁻⁶	0.4 ms	Max 91.1 Mb/s Av 90.6 Mb/s Min 90.2 Mb/s	95.6 Mb/s
GPON (New Reno)	100Mb/s BER 10 ⁻⁶	40 ms	Max 3.6 Mb/s Av 3.1 Mb/s Min 1.9 Mb/s	95.6 Mb/s
GPON (Cubic)	100Mb/s BER 10 ⁻⁷	40 ms	Max 24.2 Mb/s Av 7.8 Mb/s Min 6.1 Mb/s	95.6 Mb/s
GPON (New Reno)	100Mb/s BER 10 ⁻⁷	40 ms	Max 4.9 Mb/s Av 12.8 Mb/s Min 4.4 Mb/s	95.6 Mb/s

1.7.3 Use case assessment tests

We collaborated with FW for the implementation of mPlane in FW testbed. To make further tests on FW testbed remotely we connected with FW through a VPN. The main issues encountered were:

1. When importing the configuration file sent by FW, as transfer protocol on the VPN client were automatically set UDP on port 10000, instead of TCP port 10000.
2. The second issues encountered were the rules on the Firewalls, there were added rules on FUB firewall to allow SSH protocol and TCP on port 10000, from source private IP FUB to public IP FW. The same rule were added on FW firewall.
3. It remains open the issue of the reach ability of FW server, since the VPN acts the connection but we are not able to connect with the server, or ping it. Possible issue, there must be added the same exception on all the firewalls of the path of the connection within FW network. It has to be pointed out that the final connection adopted between FUB and Fastweb, that has been also adopted in the final test in Heidelberg, is described in D6.3.

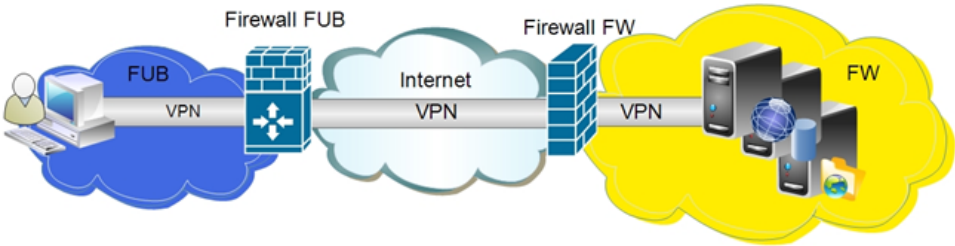


Figure 23: Scheme of connection between FUB and Fastweb

Connection with Telecom Italia:

With TI, the issues were the followings: 1. To enable the SSH protocol on FUB and TI Firewall, also to add the exception of the IP addresses. 2. To configure the file of the probe, so they could use the certificates generated by TI. 3. Still open issue is the connection with the supervisor, and a possible solution could be achieved by adding the roles on the supervisor configuration file.

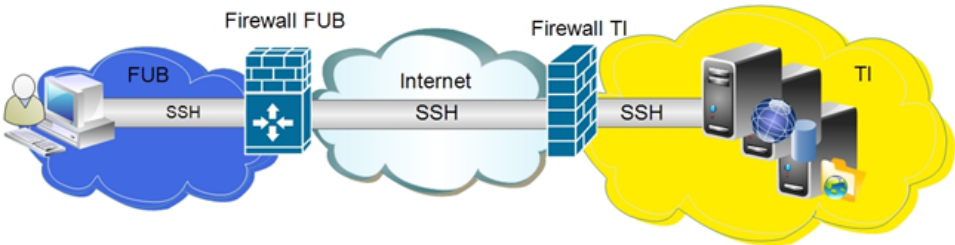


Figure 24: Schema of connection between FUB and Telecom Italia.

For the SLA verification test, the data from the public Supervisor could be analyzed. In Figure 25, we show the results of the test done for the SLA verification, for further detail please look at <http://213.140.7.241:9892/gui/static/index.html>.

destination.ip4	195.37.61.86	NEC Heidelberg								
source.ip4	89.96.78.130	Fasweb Milan								
when	2015-11-30 11:16:10.238198 ... 2015-11-30 11:16:18.669595									
<div>Values</div>		<div>Chart</div>		<div>Add to On-D</div>		<div>TCP max</div>	<div>UDP send</div>	<div>Jitter</div>	<div>Error</div>	<div>UDP Capacity</div>
time	delay.twoway.icmp	mSLA.tcpBandwidth	mSLA.udpCapacity	mSLA.udpCapacity	mSLA.udpCapacity	mSLA-Bandwidth_t				
2015-11-30 11:...	167000	43	99	11.819	4.7	92.4				

Figure 25: SLA tests on a real network.

2 Experiments and data collection

This chapter describes the experiments performed on specific use cases. Data collected in these experiments are available on mPlane official site.

2.1 Estimating content and service popularity for network optimization

2.1.1 Data and experiments

2.1.1.1 Data

The design and evaluation of the modules for this use case require realistic HTTP logs. In this case, we first have to collect a significant amount of data to train our supervised algorithms. Then, we employ the actual live deployment to feed the trained analysis modules and obtain popularity prediction results.

Training set.

Before feeding the system composing this use case, we have to train the machine learning algorithms building the the analysis modules for popularity estimation. Hence, we first collect a trace from the MongoDB which reports a set of URLs, together with their popularity time-series. The set of URLs building our model is available at <http://tstat.polito.it/traces-webbrowse.shtml>.

Live feed. We leverage the live stream of HTTP logs we collect at the backbone link of the campus network of Politecnico di Torino. We obtain this data stream by running Tstat. An example of anonymized HTTP log that we use to feed our system is available at <http://tstat.polito.it/traces-webbrowse.shtml>.

2.1.1.2 Experiments

To validate the feasibility of a system for estimating the future popularity of contents, we have considered different kinds of experiments. In the following, we first describe the architecture we deployed for this use case, and, then, the evaluation activities we run on it.

Deployed architecture

The online deployment of the system is based in Polito's premises. Tstat, installed at the egress link of our campus network, captures all HTTP requests traversing the link and log them in simple text files. Since all YouTube traffic is now delivered using encrypted HTTP transactions, we decide to leverage the modules for the extraction of content-URLs out of HTTP traffic, which have been developed for the Content Curation use case. Hence, we employ the mPlane protocol to stream HTTP logs to a repository which is attached to the analysis modules to extract content-URLs out of HTTP traffic, whose output is used to build timeseries and compute URLs' future popularity using the algorithms described in the deliverables D4.1 and D4.3.

2.1.2 Evaluation activity and results

Initial evaluations on real traffic have enabled us to explore the parameter space for the popularity prediction module. We explored a large parameter space ranging from the aggregation of requests to the setting of model variables. The evaluation of the performance of our algorithms showed two findings with respect to the state of the art. First, unlike the state of the art, because the underlying models of our approach are probabilistic graphical models, they enable us to evaluate over ranges of values, rather than against specific configurations of the training setup. For example, compared to regression based state of the art models, see [5], instead of building individual regression models for different targets, i.e. predict 24hrs ahead having observed an hour, we construct a single model that covers the complete domain of observed values, i.e. predict X having observed Y (over the range of values in the data). Second, our method outperforms the state of the art both in long and short term prediction. We observed improvements between 5-65% percent improvement in accuracy over predicting various ranges of observed and future content views.

Concerning the online deployment of the modules, we found that models could be further simplified, with respect to use cases. For example for caching, since we are not interested in long horizon predictions (i.e. over 10s of hours in the future), we found that short time horizon models, i.e. built with data from just a few hours were sufficient, further we found that under this use-case, we could also transfer knowledge between content types. For example, models built for video content were accurate in predicting the behaviour of non-video urls, and vice-versa. Finally a detailed discussion can be found in [4].

2.2 Web Content Promotion and Curation

2.2.1 Data and experiments

2.2.1.1 Data

The design and evaluation of the modules for this use case require realistic HTTP logs. We use two types of HTTP logs: ground truth traces and traces collected at real networks.

Ground-truth traces. We generate HTTP logs in a fully-controlled testbed. We manually visit the top-100 most popular websites according to Alexa ranking. When we are in the main page of each of these sites, we randomly visit up to 10 links they reference. We collect all the visited URLs as they appear in the browser bar. In parallel, we capture all the HTTP requests. This trace contains a total of 905 user-URLs, corresponding to 39,025 HTTP requests. This trace is available for the download at <http://tstat.polito.it/traces-webbrowse.shtml>.

HTTP logs. For this use case, we employ several HTTP logs we collect at the backbone link of the campus network of Politecnico di Torino. In all cases, we obtain the traces by running Tstat. An example of anonymized HTTP log that we use to feed our Content Curation platform is available at <http://tstat.polito.it/traces-webbrowse.shtml>.

2.2.1.2 Experiments

To validate the feasibility of an automatic content curation system, we have considered two different experimental approaches.

First, we evaluated the accuracy of the modules composing the Content Curation platform using the set of traces described above.

The second evaluation approach has been carried out by employing the actual use case live deployment and studying the data obtained by Google Analytics about the interaction of users visiting the website <http://webbrowse.polito.it> (WeBrowse in the following) which collects the URLs extracted by the Content Curation modules. Second, we asked the users to leave a feedback about the content promoted in the website in an evaluation form.

Deployed architecture

The ground-truth traces described above have been collected in a controlled environment, i.e., using a standard PC and a browser to browse the list of websites and log the URLs actually contacted by the browser.

For the online deployment of the content curation system, we leveraged the mPlane deployment in Polito's premises. Tstat, installed at the egress link of our campus network, captures all HTTP requests traversing the link and log them in simple text files. Then, we employ the mPlane protocol to stream HTTP logs to a repository which is attached to the analysis modules described in the deliverables D4.1 and D4.3.

2.2.2 Evaluation activity results

The test conducted using ground-truth traces allowed us to choose the proper algorithms and parameters settings which maximize the accuracy when extracting content URLs from HTTP traffic. Details can be found in [1]. Interestingly, the online implementation of our analysis modules achieves the same accuracy as ReSurf [8], the most prominent algorithm alternative to our approach in the literature, if not slightly better (82.97% of recall and 90.52% of precision). More importantly, our modules are lightweight, and, when comparing the processing time on the same trace, we find that they are 25 times faster than ReSurf. A more comprehensive description of the results of our experiments is available in [7].

A second series of tests was dedicated to understand how users might welcome a Content Curation platform such as WeBrowse. To this end, we advertised WeBrowse in two rounds, each time contacting different sets of users. 93% of those who were contacted during the first announcement (R1 in short) are mostly students and professors from the Computer Science and Electronic departments of Politecnico di Torino. We specifically targeted these users to collect feedback with a more technical nature. The second round of advertising (R2) reached a wider population, i.e., professors, researchers, and students from different areas (engineering and architecture) and administrative employees. We observe that, in total, the website was visited by more than 1500 users, and 115 of them filled the evaluation form. In general, feedback about the system is very encouraging, and testify that our approach to Content Curation is welcomed by the users.

We summarize the feedback of the 115 respondents in the following. We split the questions in our evaluation form into two main groups. The first group helps us evaluate whether users like WeBrowse and the second focuses on our promotion methods. Not all respondents answered all questions.

Do users like WeBrowse? We ask questions about their experience with WeBrowse and to compare WeBrowse with the service they use to discover content on the Web. Tab. 14 summarizes the responses. Overall, respondents were positive: the wide majority of respondents find WeBrowse at least interesting or extremely interesting. Similarly, 71% in R1 and 91% in R2 of respondents find it useful or extremely useful. Interestingly, responses during working hours (9am to 6pm) found WeBrowse more interesting than answers in other hours. This positively correlates with the dynamic behaviour of WeBrowse.

We also ask users in R2 to list the services they usually use to stay informed, and compare them to WeBrowse. As shown in Tab. 14, 25 of respondents rely on news portals to keep informed; Facebook comes second (12 respondents). Interestingly, 41 respondents say that WeBrowse is simply different from these services. These answers are encouraging as we see WeBrowse as a complement, and not a replacement, to existing curation systems.

Finally, the 62% (70%) in R1 (R2) say they would like to have WeBrowse as a service offered by their network. 44% (30%) in R1 (R2) would use WeBrowse at least once a day.

How good are WeBrowse's promotion algorithms? We ask users to rank the three different tabs in WeBrowse (Top, Hot and Live Stream) using a Likert Scale from the most interesting to the least interesting. We calculate the average ranking score for each tab. The scores are fairly close, with the Top tab coming first, then Hot, and Live Stream as last. This result indicates that users have different tastes, and having different tabs with different promotion methods is important to please a large user population.

Finally, see [7] for a more comprehensive description of the evaluation of the system.

Table 14: User feedback from the WeBrowse evaluation form.

How interesting is WeBrowse's content?	R1	R2	How useful is WeBrowse?	R1	R2	Which service do you use to keep informed?	R2	How do you compare this service to WeBrowse?	R2
extremely interesting	8	24	extremely useful	4	14	Web Newspapers	25	More interesting	5
very interesting	25	19	very useful	20	20	Facebook	12	Less interesting	7
interesting	17	10	useful	18	16	Google News	4	Simply different	41
poorly interesting	5	1	poorly useful	10	1	Twitter	3		
not relevant	4	12	not relevant	7	4	Newsletters	2		
						Other Media	10		

2.3 Active measurements for multimedia content delivery

2.3.1 Data and experiments

Looking back to our original agenda for Mplane assessment tests (i.e. as defined in D1.1), there is noticeable change of scope and goals in most Use Cases. For the Multimedia content delivery UC, the most important change is that we departed from YouTube videos to content delivered by smaller content providers using standard-based protocols. The primary reason for this shift is that here we are focusing on technologies that can be leveraged by minor content providers and ISP-s (like smaller telecommunication companies found in many EU countries), who are in full control of their CDN and want to assure their service through state-of-the-art monitoring. At the same time, Mobile Network performance UC (see below) has essentially become a YouTube video accessibility and performance testing UC (basically from a ISP perspective), so YouTube remains covered by Mplane as a whole.

As a result, for the Multimedia Content Delivery UC we consider an Internet or telecommunication service provider who also operates an "Over-the-Top Video CDN", to serve its clients with "multi-screen", i.e. offers some capabilities to access content besides the primary screens, i.e. the TV sets in their home. This type of bonus/premium service has recently become popular with providers and subscribers are also using it more and more frequently.

These CDN-s typically serve both off-line, VoD titles (i.e. movies), and live content of popular TV channels, which are accessed real-time or with relatively short (i.e. less than 1-2 days) delay. For the live content, a sophisticated video-ingestion mechanism is operated which continuously transcodes the signal to multiple formats and qualities, and stores them on the CDN member servers. VoD service is also challenging, but from other reasons: while no real-time ingestion is needed there, the growing variety of titles offered requires some deep (e.g. tape) storage and caching mechanism, which needs to be monitored.

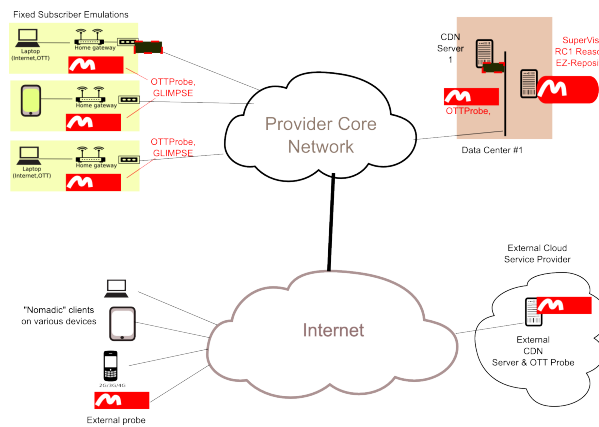
Our experiments are carried out over a test CDN service, built in the Fastweb test plant that emulates most aspects of such a service:

1. Content permanently available and content ingested live.
2. Most users in the home network of the ISP, with some "nomadic" ones further away
3. Variety of user devices, access bandwidth (about 0.3..100 Mbps) and OTT streaming protocols (HLS, MPEG-DASH, MSS) used.

2.3.2 Deployed architecture

The test network finally built is shown in the figure below. It is a distributed network, with 2 CDN servers, and a number of clients; some of them served by the exemplary provider (emulated by Fastweb), while others are connecting from various sites on the Internet (from Italy, Hungary and Romania). Please note that the other, higher-level Mplane components are centralized i.e. they operate as a single instance. Distribution of functionality at this level was out-of-scope for this UC, and the fact that the Supervisor (and also the Repository, accessed through Indirect Export") was accessible from the public Internet made such a single central server architecture cleaner.

The figure also indicates that a few network problem emulation devices (MiniProbes with netem deployed) were available for our experiments.



To show the architecture and connections of the Mplane Components, we are showing a detailed architectural figure, which combines the concepts shown in D4.3 and 3.4 deliverables. This upgraded version also shows an essential concept, i.e. the work distribution between the Reasoner and the Repository through the single "QueryByCriteria" capability. Please see the mentioned deliverables for further details.

- The SVGUI itself allows the visualization of current and past measurements (and also make it possible to schedule new ones, although this is not used in the demo). This is a low-level view of the results, in form of tables and charts directly drawn from the measurements.
- The dashboard, on the other hand is capable of displaying higher level, customized information on conveniently configurable visual "widgets", i.e. charts, tables, topologies/maps, etc. If some data is available from the backend modules (primarily the Reasoner and/or the Repository), the efficient visualization is provided.

Considering the effort on building these tools, and the generality of their usefulness, they belong to the principal achievements of this Use Case.

Reasoner configuration

As described in WP4 deliverables, the configuration of the RC1 Reasoner is critical for successful issue detection and root cause analysis. While Probes and the Repository 'just' need to operate correctly and efficiently, the Reasoner will be eventually responsible for orchestrating those other components, for declaring problems and for coming up with diagnoses.

The current version of the Reasoner used in the tests and demos fulfills this task with the following steps and algorithms:

- **0: A priori knowledge.** The reasoner will have a complete knowledge of the network topology i.e. the major segments of the network paths between the servers and the clients. This information is to be provided for the Scheduler from external sources. Currently, this is a simple static database, but we expect that in larger scale the providers address management system and inter-AS routing databases will provide this information.
- **1. Scheduling of routine supervisory measurements.** All probes available in the system are configured to for a redundant and periodic coverage of A. all content servers and B. all content titles/programmes, C. all network paths, (except for individual access lines, where coverage is optional). All probes are configured with a "bandwidth.baseload.kbps" parameter, which defines the amount of average bandwidth the probe is permitted to generate. This may be set to 0 for probes that do not want to participate in the routine tests (e.g. probes on mobile links with traffic-dependent fees).
- **2. Screening of routine test results.** We need an efficient way to determine if the services (including content, servers and network) is "fully OK", or there is some suspicion about the operation at some parts. To tolerate results which are "failed for natural reasons", we set a % threshold on the total and failed counts of measurements. The Repository makes it possible to determine such failure percentages with a single "QueryByCriteria" query, i.e. one that uses wildcards for all of the criteria parameters. This technique is scalable until the combined number of supervised servers and content items is small enough (e.g. up to a few dozens, like in the experiment) so that any component's failure causes a rise above the threshold. With larger services, multiple queries (e.g. for partitions created from the content list) can be used for robust screening. As for grade selection, we use some generic grade class which reflects user experience (like "CombinedQoS") and medium ranges, (like up to 3 out of 5), i.e. searching also for transactions that were eventually successful, but had definite problems.
- **2/b External triggers.** In accordance with the policies implemented by real-world operators, we can make screening a non-essential step, if it is also possible to externally trigger investigations based on "customer complaints". Screening in this case provides some additional "proactivity" for the service monitoring.

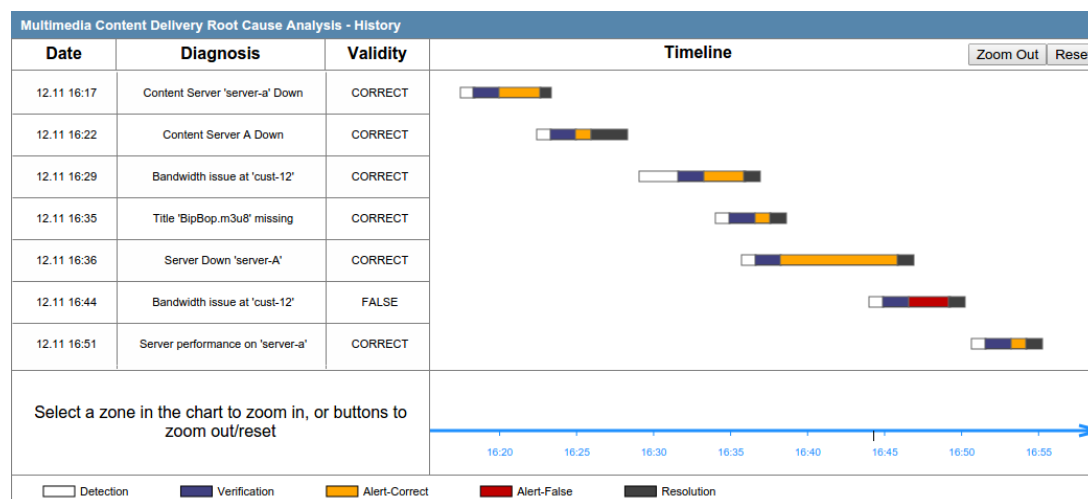


Figure 28: On-dashboard visualization of the timeline of some test cases executed and diagnosed (including one false diagnosis).

- 3. Hypothesis search.** When the screening indicates failure rates above threshold, the Reasoner triggers the search for some possible root causes by running more specific Repository queries. This step is more application-domain dependant than the earlier ones, as each domain seem to have different "error patterns" to consider in the diagnosis. With a multimedia content service, we selected a "server-network-content" priority order, i.e. the Repository is queried for individual CDN servers, network segments, and finally content titles in this order. With narrower searches, generally lower grade threshold levels can be used to identify "significant" problems only (but again, this may be application-dependant).
- 4. Diagnosis consolidation.** Finally, the Reasoner will start up additional Probe measurements, to verify the hypothesis (or hypotheses). The approach of the current Reasoner is to start up a redundant set of measurements, with at least 3 of all other parameters varied, plus lower-level measurements using other Probes. E.g. in case of a server failure hypotheses, ≥ 3 content pieces and ≥ 3 sources and both servers (as we do not have more) are designated for verification tests, plus HTTP or PING thests are also defined. Once at least 70..90% of tests support the hypothesis, the diagnosis is considered verified, but on-demand tests remain operational (possible with lower frequency), to make "back to normal" transition quickly detectable.

Test results and takeaway

As noted above, the Reasoner was tested with a few iterations of some basic (and single) failures only, as our principal goal was to test and demonstrate the "mPlane mechanisms" rather than to provide solutions for this problem domain (which would be close to impossible with such a limited and artificial testbed). We created a simple Event chart and an "event registration engine" to visualize the test progress and outcomes. This (and the test scripts mentioned earlier) allowed us to run a series of tests with multiple failures and diagnoses within a period of 1-2 hours. The event register knows about the real cause of the problems, so the correctness of the hypotheses is also indicated in the event diagrams displayed (i.e. orange: correct and red: false; in Figure 2.3.3).

As for a **high-level summary** of this Use Case experiment and investigation, we see that mPlane provided an architecture for these type of supervision/monitoring/diagnosis problems, which is sophisticated, but only at a level which is justified, manageable, and probably necessary. It was possible to build Compo-

ment prototypes that demonstrate root cause analysis at a level which is about as good/efficient and significantly more flexible than some well known "market leader" commercial products built with obscure, wired-in proprietary logic (and with dozens of man-year effort). The separation of concerns, i.e. Probe-Repository-Reasoner is a good idea also from the practical/useability/deployability perspective. While the Reasoner is still the most complex, and dominant component, the Repository can be defined in a way that it relieves the Reasoner with usefully preprocessed and "compressed" information.

Probes are definitely the most straightforward components, and -thanks to the the mPlane reference implementation- Probes are easy-to build, even upon existing network testing technologies built with some other architecture in mind. This is demonstrated by the impressive array of mPlane Probes presented by WP2.

We see that the EZ-Repo Repository can be easily extended with more sophisticated query fuctions (true, its current efficiency and robustness also needs to be improved significantly). The (currently mock) database component for storing queryable data should be revised and the current appraoch of using relational-like data may be changed to a no-SQL technology.

Finally, we consider the Reasoner component is a relatively moderate success: it is complex also in this UC, and although we targeted generality, in many parts it is rather application dependant, where some minor change of the tested system (e.g. different topologies or different - no matter if more or less- amount/quality of a-priory information) will need significant changes to the reasoning algorithms implemented. There is nothing wonder about here: troubleshooting of complex systems needs intelligence, and also good instinct, which could not be expected from mPlane either. Of course, the current RC1 reasoner was written with re-use and extensibility in mind, so such changes and improvements can be done in a relatively simple way.

2.4 Quality of Experience for web browsing

2.4.1 Data and experiments

With the main goal of validating the diagnosis algorithm for finding the appropriate root cause for a high page load time, we describe here the carried out experiments, by analyzing at first the tuning of the system parameters and then describing the network test-bed considered in the validation process and the obtained results (see [2] for further details).

The proposed algorithm (see Deliverable D4.3, Section 2.6) presents several parameters that have to be tuned before launching the probe. Nonetheless, this phase is not so critical: we design the algorithm so that small changes in the different parameters values result in the same diagnosis result. In more detail, we have to determine the following quantities:

- EWMA parameter α : in our settings we have used the value $\alpha = 0.9$, which is “classical” in many network applications;
- CUSUM parameter c : we have set $c = 0.5$, as in other previous works on CUSUM;
- Algorithm thresholds: the choice of these thresholds, that usually represents a critical aspect in the application of CUSUM based methods¹ in other fields (e.g., network anomaly detection), has resulted not to be that critical in this application scenario.

Indeed, the problem normally connected to the choice of these thresholds is that it has a direct impact on the number of detected anomalies, but also on the number of false positives (events signalled as anomalous that are, in fact, normal events). Nonetheless, in our application scenarios, we can accept a certain number of false positives, without affecting the system performance. This is due to the fact that having a false positive, without the signalling of the problem, does not lead to any conclusion. Hence, from a practical point of view, we have tuned these thresholds to a value that is equal to the mean value of the CUSUM obtained during a normal session plus a corrective factor computed as a function of the CUSUM variance (i.e., scaling).

2.4.2 Deployed architecture

To validate and verify the behavior of the diagnosis algorithm and the performances of the proposed probe, we have taken into considerations two distinct experimental scenarios: a controlled laboratory testbed to validate the proposed diagnosis algorithm, and a set of browsing sessions into the “wild” Internet, to verify the suitability of the developed probe for real-world applications.

At first, an exhaustive set of experiments has been conducted in a testbed composed of four distinct PCs, configured as depicted in the figure 29, so as to verify the effectiveness of our proposal.

Given the setup of the testbed we have been able to emulate three distinct cases:

- “normal” functioning
- congestion on the local network
- congestion the backbone network

¹Thresholds regarding page sizes are of course domain-dependent, and vary from page to page when browsing real web sites.

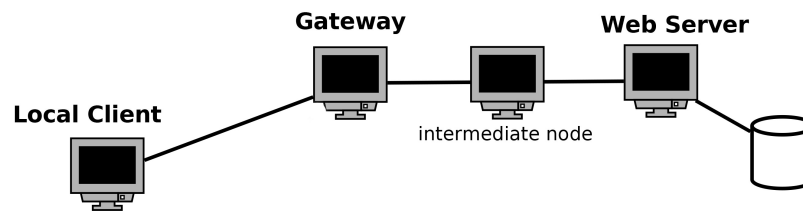


Figure 29: Schematics of the used test bed.

The three cases have been realized by using *netem*, which providing us with the ability of automatically adding variable losses and delays on the network, has allowed the realization of a labeled dataset (ground truth).

2.4.3 Evaluation activity results

Note that in this testbed we have not involved any human interaction, meaning that the diagnosis algorithms has been used over all the sessions and not only when a “dissatisfaction signal” was generated. It is important to highlight that this fact could bias the results, in terms of a bigger number of false positives (that could be not relevant in the “real-world” scenario, where the user not necessarily raise an alarm), but not in terms of false negatives. Indeed, the problem normally connected to the choice of the algorithm thresholds is that it has a direct impact on the number of detected anomalies, but also on the number of false positives (events signaled as anomalous that are, in fact, normal events). Nonetheless, in our application scenarios, we can accept a certain number of false positives, without affecting the system performance. This is due to the fact that having a false positive, without the signaling of the problem, does not lead to any conclusion. Hence, from a practical point of view, we have tuned these thresholds to a value that is equal to the mean value of the CUSUM obtained during a normal session plus a corrective factor computed as a function of the CUSUM variance (i.e., scaling). For this reason, in this tests we have also performed a preliminary training phase aimed at computing the threshold values.

Table 15 shows the obtained results. In more detail, over a total of about 1800 distinct browsing sessions, the algorithm has not produced any false negative, and it has introduced 11 false positives. Moreover, in case of really anomalous sessions (i.e., very high latencies and packet loss ingested) the algorithm has always correctly identified the cause.

Finally, to conduct a preliminary performance evaluation of the probe, verifying its suitability for real world use, we have conducted experiments into the “wild” Internet. This last scenario is not used to validate the diagnosis algorithm, as we do not have any control on the full path between the probe and the web server, but to verify if the developed system is able to deal with a real operative network scenario. The overall process of browsing a URL and running the diagnosis algorithm for a single session spans from 1 to 3.5 minutes, that we think it is a reasonable time for providing the end user with a diagnosis for a poor QoE. This time span is due to the browsing timing itself, which differentiates between small web sites (e.g., Google front page) and complex web sites (e.g., news web sites with a high number of servers to contact to fetch different objects). Most of the time is spent performing the active measurements: we have to wait for Ping messages and Traceroutes to return their results. As previously mentioned, all the results are stored locally and sent to a central repository for further analysis. We store all the collected data and the diagnosis result in JSON files, growing from less than 20 kB (small sites) to a maximum of 800 kB (very big sites).

Considered Case	Algorithm Output		
	"Normal" Functioning	Local Network Congestion	Backbone Network Congestion
"Normal" Functioning	1617	9	2
Local Network Congestion	0	112	0
Backbone Network Congestion	0	0	159

Table 15: Experimental Results

2.5 Mobile network performance issue cause analysis

2.5.1 Data and experiments

Notice that in previous deliverables we have described the performance of our system in a controlled (lab) environment.

In this section we describe and discuss the results of the system's evaluation in two real world settings. In the first environments clients are in a corporate WiFi network where we can artificially introduce faults. In the second case, clients access videos over a wide range of wireless networks including both 3G and WiFi, where faults are not controlled and occur naturally. In both cases, clients retrieve videos from both a private server and YouTube.

2.5.2 Deployed architecture

The purpose of the the real world experiments with induced faults, is to get labeled data that will enable us to evaluate the robustness of the trained model on a real wireless network which is characterized by unpredictable topology, constant variations in traffic, signal strength and number of connected devices.

For the measurements, we distribute five Galaxy S II to equal number of users for a period of one week. The phones are again equipped with an application that automatically launches random videos from the top 100 list, while coordinating the network and hardware probes. The users were instructed to carry the phones with them while inside the wireless range in order capture variations due to movement and received signal quality.

In these experiments the videos are streamed from both our private video server and from YouTube with probabilities 0.25 and 0.75 respectively. We select these probabilities so that we end up with a dataset where the majority of measurements corresponds to YouTube sessions and a smaller part to streams from our server.

Using the same methodology as the one in the controlled experiments, we introduce five different types of faults, lan congestion, wan congestion, mobile load, low rssi and wifi interference. Furthermore, we ensure that the conditions of the network are sufficient to successfully load a video just before and after the induced fault. However, since this a semi-controlled environment, we cannot fully guarantee that during each video flow there are not additional (spontaneous) problems over the unmanaged Internet links or video services.

The collected dataset consists of 2619 instances from which 1962 are good, while 463 have mild and 194 have severe QoE issues.

2.5.3 Evaluation activity results

Our goal is to evaluate the ability of the classifier to predict labels in the real world scenario based on the training that was performed using the controlled dataset.

In this part we demonstrate the system's capability of detecting *the existence* of problematic instances using either one of the probes or the combination of all three. The detection is done with 88% accuracy

when using the mobile probe, 84% when using the router and 81% when measurements from the server probe are only used. The combination of the three probes yields accuracy of 88.1%.

Figure 30 illustrates the Precision and Recall values for this phase of the evaluation. Overall, the results match the controlled experiments. In this case too, the mobile VP outperforms the other two VPs. However, one notable difference is the increase in both Precision and Recall for the mild problem detection. This can be attributed to the fact that the variations and background noise in the current environment is less than the variations we simulated in the controlled experiments.

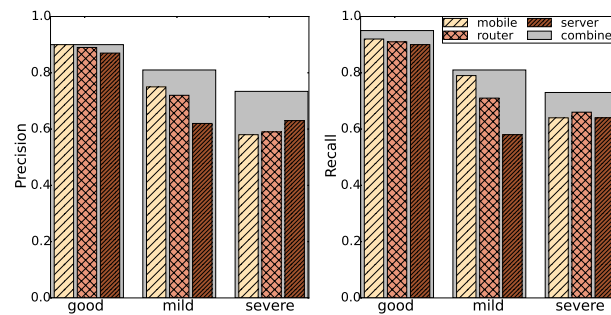


Figure 30: Precision and Recall for problem detection in the real world experiments per vantage point.

Furthermore, we also observe equally good robustness of the trained model in terms of detecting *the exact root cause* of a playback problem. In this case, the combined use of the three vantage points allows correct detection with accuracy of 82.9%. When using separately the mobile, the router and server VP we obtain accuracies equal to 81.1%, 80.5% and 79.3% respectively.

From Figure 31 we see better performance for device load and wireless medium issues which is to be expected given the strong correlation of these faults with specific hardware metrics. In the lan congestion scenario we observe better results from the mobile and the router VP while for the case of wan congestion the server is detecting problems with higher accuracy.

For each of the entities that participate in the video delivery this means that the VP on the client's device is necessary for detecting the root cause of the majority of problems. ISPs on the other hand, can effectively discover LAN faults but also wireless errors such as low rssi and interference. Finally, content providers can perform WAN fault identification with good accuracy but fall short when it comes to finding faults that occur on the device or in the wireless medium.

Takeaway: Our findings here are in agreement with those in the previous experiments for problem detection and root cause identification. This is a strong indicator that *our system that was initially trained in a fully controlled environment can be successfully applied in the wild*. At the same time, smaller differences in the detection of some problems emphasize the importance of continuous training. While collecting large-scale groundtruth in the wild might not be feasible, it is still possible to acquire some labels as specific problems can be recognized by experts within each entity (e.g., network engineers). Furthermore, groundtruth about the quality of experience can be given by means of crowd-sourcing (i.e., people complaining at call centers, or feedback provided by the users within the application).

Deployment Without Induced Faults

The final step in the evaluation is *detecting faults that were not induced by us* and, therefore, might be more complex. Furthermore, a particularly important aspect of this evaluation is to test the system in mobile networks, given that there is a constantly growing number of users who watch video over cellular broadband connections.

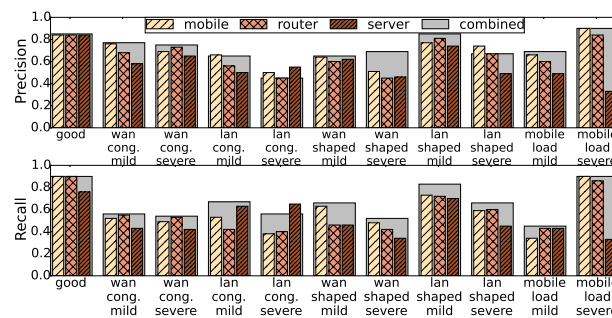


Figure 31: Precision and Recall for problem detection in real world experiments

In this scenario too, we distribute five Samsung Galaxy S II devices to equal number of users for one month with the instruction to carry the phones with them at all times. The phones contain SIM cards with unlimited 3G data-plans and the users were allowed to connect them to any WiFi access point. This approach allowed us to test the system on a multitude of networks that use either cellular or 802.11 technology.

The videos are again streamed from both our private video server and YouTube with 1:3 ratio so that the final dataset is richer in measurements from the YouTube service. A probe collects network statistics on our video server for the sessions streamed from it. With this methodology we can have three different VP combinations, i) (mobile, router, server) when the user is streaming video from our server while using our WiFi, ii) (mobile, router) when YouTube videos are streamed on our WiFi, iii) (mobile, server) when videos are delivered from our server over other networks and iv) (mobile) when streaming from YouTube on other networks. Given that the majority of the videos were delivered over 3G and in order to make the results comparable between 3G and WiFi, we removed any features from the router (therefore only the mobile and server vantage points are used).

Similar to the previous scenario we use the trained model from the controlled experiments. For the real-world experiments, although all mobile-based measurements (e.g., hardware as well as the number of re-buffering events) are always available, the number of other metrics varies depending on the number of VPs that were used. The real-world dataset contains 3495 instances from which 2940 are good and 555 problematic.

Does it Work in the wild with real faults? Since the experiments are done in the wild, we are unaware of the root cause behind the stalls and, therefore, we can only mark instances as good and problematic.

In terms of identifying the *existence* of a problem, the mobile probe, server and their combination still achieve a high accuracy and recall, as shown in Figure 32.

Similar to the controlled experiments, we find that the mobile VP is a better choice than the server for identifying both good and problematic instances while the combined use improves the system's accuracy.

Takeaway: Overall, the results from the real world experiments verify that the system is equally effective when detecting problems in the wild even when fewer VPs are available. Although the detection of healthy video sessions is achieved with high accuracy, there is some loss regarding the identification of problematic videos. This loss occurs due to differences in the characteristics of the faults that we encounter in the real world as compared to the ones we induced manually in the previous sections. This effect can be minimized by introducing more VPs (e.g., on 3G RNCs) in order to get more fine grain information about how smaller variations affect the video QoE and by furthermore training the classifier with a wider range of problems. Finally, as discussed in the previous section, these figures are likely to

be improved once more labeled faults are fed into the training set.

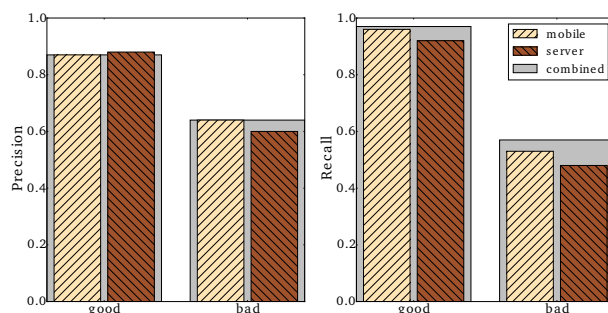


Figure 32: Precision and Recall for problem detection per VP pair in the real world.

Identifying the Root-cause

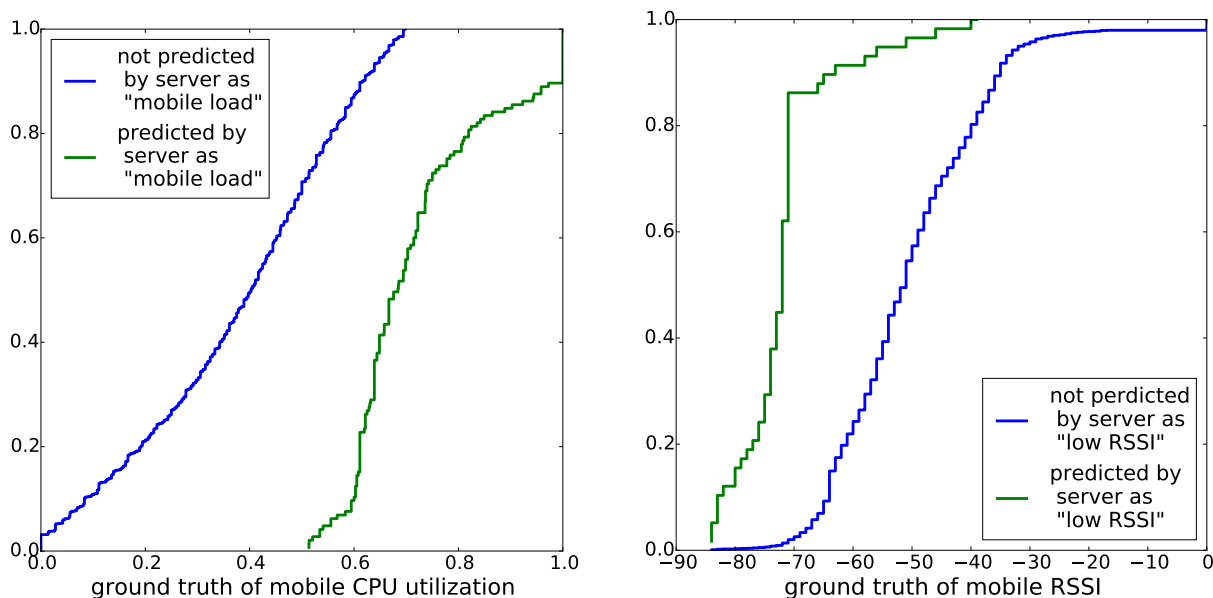


Figure 33: Comparing the server estimations about CPU (left) load and RSSI (right) to the ground truth

We can use the trained model from the controlled experiments to predict the root cause of faults that occurred in the problematic sessions. The results of the predictor's output can be found in Table 16. As we observe, the most common type of problems occur within the users' local network (13% of all instances). Surprisingly only few (2%) of the instances are estimated to be caused by low RSSI or WiFi interference as typically the videos fail to even start a TCP flow when there is very low signal. Furthermore, a number of instances (4%) were problematic due to an estimated high mobile load.

As discussed in the previous section, we can directly calculate that the algorithm correctly identifies good instances with 85% accuracy. Furthermore, although it is not possible to verify all of these estimated root causes, we still have the ground truth for some of them: mobile load and low RSSI.

Figure 33(left) shows the distribution of CPU load on the mobile device for problematic videos sessions as predicted by the video server Vantage Point. Two different distributions of the CPU ground truth are given: video sessions that server VP labeled as high "mobile load" and the remaining video sessions.

The results show that, although the server vantage point only has access to transport layer metrics (TCP statistics), the video flows that were estimated as high mobile load have indeed much higher CPU utilization.

Similarly, Figure 33(right) shows the distribution of RSSI for the instances that were considered as low RSSI from the point of view of the server's vantage point. As before, we observe that the server vantage point can successfully identify these instances despite the fact that the phones were connected to various WiFi and 3G networks.

Takeaway: These results further reinforce our hypothesis that a model that was trained in a controlled environment is robust enough to be applied as a starting point on a real world environments where the network conditions and the faults can be highly dynamic and unpredictable. Furthermore, we observed that even the service provider VPs can identify problems that occurred within the users network or device (e.g., low RSSI or high CPU usage) without any external information.

GOOD	WAN CONG.		LAN CONG.		MOBILE LOAD		LOW RSSI		WIFI INTER.	
	M	S	M	S	M	S	M	S	M	S
2499	163	166	18	446	2	132	26	0	43	0

Table 16: Real-world root cause predictions (M=mild, S=severe)

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

2.6.1 Data and experiments

To verify the correct functioning of the Anomaly Detection use case and to test the capabilities of mPlane to detect anomalies in large-scale services such as YouTube in a real scenario, we deployed different mPlane components in the operational network of Fastweb and at Polito premises, and analyzed the real traffic of its customers for a period of one month, detecting and diagnosing a major anomaly impacting the QoE of Fastweb customers (conversations with the mPlane Fastweb team confirmed that customers negatively perceived the detected degradations).

The dataset used for the analysis corresponds to one month of YouTube flows, collected at a link aggregating 20,000 residential customers who access the Internet through ADSL connections. The complete data spans more than 10M YouTube video flows, served from more than 3,600 Google servers. To identify and diagnose performance issues, we rely on the Anomaly Detection analysis modules, applied to several features describing the YouTube traffic delivery and its performance, such as download throughput, traffic volume served per each observed Google server, etc. Flows were captured using the Tstat passive monitoring probe. Using Tstat filtering and classification modules, we only keep those flows carrying YouTube videos. The complete dataset is imported and analyzed through the DBStream repository. Finally, using the server IPs of the flows, the complete dataset is complemented with the name of the Autonomous Systems (ASes) hosting the content, extracted from the MaxMind GeoCity ASes databases².

Deployed architecture

Figure 34 depicts the architecture deployed in for the aforementioned experiments. YouTube flows captured and filtered by Tstat at Fastweb premises are exported to a DBStream instance running at Polito, where the anomaly detection modules continuously run. YouTube flows are served from multiple geo-distributed caches, as part of its omnipresent CDN.

2.6.2 Evaluation activity results

We focus now on the obtained results for this specific scenario. The mPlane anomaly detection modules started detecting an anomaly impacting the QoE of YouTube users on a precise day (Wednesday the 8th of May) and during peak load time/heavy traffic load hours. The anomaly persisted for several consecutive days, occurring always at peak load times. The complete mPlane diagnosis process pointed to a load balancing policy employed by Google's CDN, which resulted in the aforementioned customer experience degradation. Next we report the obtained results, focusing on the specific week where the anomaly was originally detected.

The origin of the analyzed anomaly is the cache selection policy applied by Google from Wednesday on, and more specifically, the servers selected between 15:00 and 00:00 that were not correctly dimensioned to handle the traffic load during peak hours, between 20:00 and 23:00, leading to users' QoE degradation. In [3] we have shown that it is possible to detect such an anomaly with the mPlane

²MaxMind GeoIP Databases, <http://www.maxmind.com>.

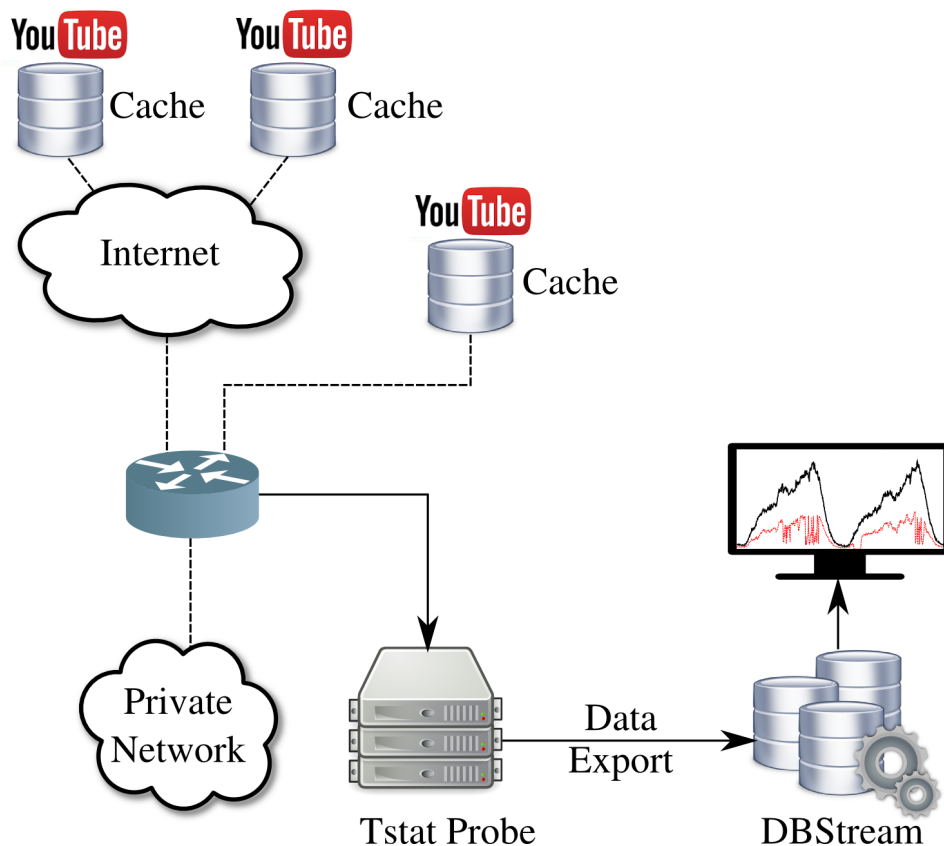


Figure 34: Deployed architecture for the Anomaly Detection use case experimentation.

anomaly detection modules, analyzing the time series of the distribution of the Average video download rate, along with the median of the β parameter: recall that β is a QoE based KPI defined as the ratio between the average download rate and the video bit rate, which allows to estimate the presence of stallings in the video playback.

Fig. 35 plots the output of the distribution-based anomaly detection module for the average download rate, and flags the presence of changes during the peak hours from Wednesday to Friday and on Sunday. Fig. 36 plots the trend of the median β parameter in the period and two thresholds for $\beta = 1$ and $\beta = 1.25$, which in turn identify three regions for the video QoE, i.e., bad, poor and fair represented with red, orange and green, respectively. These thresholds are derived from the QoE mappings presented in [3], and correspond to 400 and 800 kbps, respectively, in case of 360p average bit rate videos. The figure reports a reduction of the throughput on Tuesday at peak-load time, between 20:00 and 23:00 UTC. However, from Wednesday on, this drop gets below the bad QoE threshold. The drop in the throughput coupled with the marked drop in the time series of β reveals the presence of a change that is heavily affecting the user experience. Therefore, we use them as symptomatic signals, i.e., those that trigger the complete analysis process (see deliverables D4.2 and D4.3 for additional details on such a classification).

The list of features we are using for the diagnosis process is summarized in Table 17. Given that the diagnosis part focuses on the YouTube servers, as diagnostic signals we have considered the distribution of flows per server IP, and the elaboration time (i.e., the time elapsed from the video request and first returned video segment). Furthermore, we have considered the minimum internal and external RTT, which are representative of the network distance from the vantage point to the end device and from

Table 17: Tstat flow-level ticket information.

Field Name	Description
client IP	Anonymized device identifier
server IP	remote YouTube server IP address
avg download rate	average flow down-link throughput
elaboration time	delay between client request and server reply
external RTT	RTT measured between VP and remote server
internal RTT	RTT measured between VP and end device
beta	ratio between video bit-rate and throughput

the vantage point to servers, respectively. Results reported in Fig. 37(a) show that a different set of Google servers was selected to serve the YouTube traffic in the afternoon from Wednesday onward. Also, Fig. 37(c) and Fig. 37(d) show that the new servers were farther located from our vantage point, and that there was no relevant ISP internal routing change in the same period. However, the selection of the new servers negatively impacted the elaboration time (see Fig. 37(b)), to the point that the perceived service QoE fell below the acceptability threshold for a considerable share of the user population (cfr. Fig. 35).

To conclude, the final diagnosis of the event is that a new cache selection policy applied by Google from Wednesday on provokes an anomaly, i.e., a decrease of average downlink throughput with consequent QoE degradation. The presence of the new policy is confirmed by two diagnostic signals (distribution of flows per server IP and per external RTT bins). The new servers deployed in the afternoon, from 15:00 to 00:00 were potentially not correctly dimensioned to handle the traffic load during peak hours, between 20:00 and 23:00, as indicated by the change in the elaboration time distribution. Notice that by combining the detector output for the symptomatic and diagnostic signals, we have automatically drawn the same conclusions as already obtained manually in [3]. Finally, we do not discard the event in which the anomaly is actually caused by inter-AS path congestion in the downlink direction; indeed, a more evolved version of this use case employs the DisNETPerf analysis module [?] and the mPlane RIPE Atlas integration proxy to analyze the performance of Internet paths through active measurements.

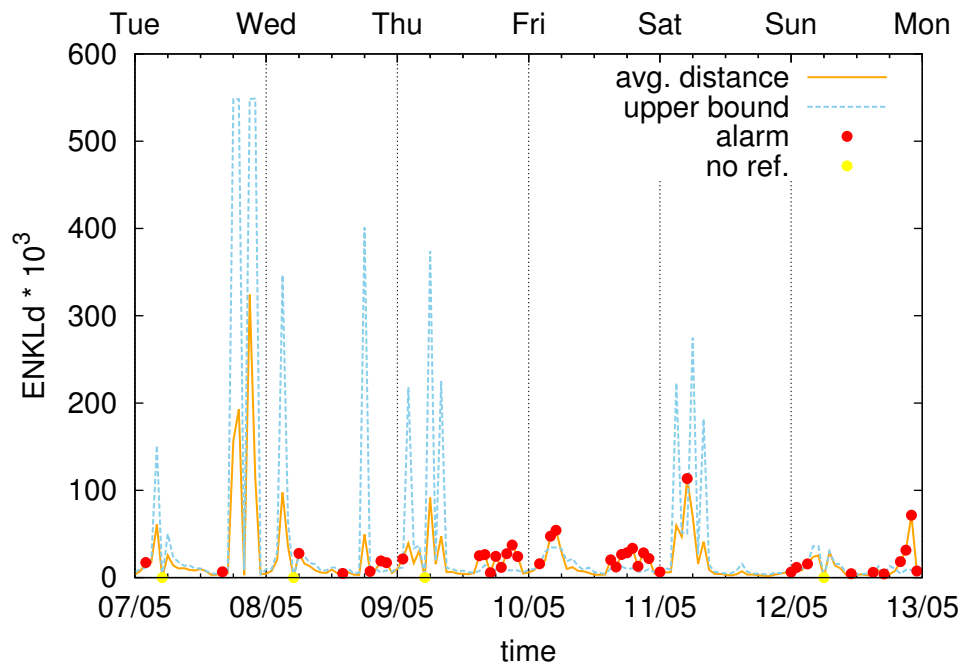


Figure 35: Output of the anomaly detection analysis module for the symptomatic signal distribution of Average download rate, used as trigger for the diagnosis procedure during the YouTube anomaly.

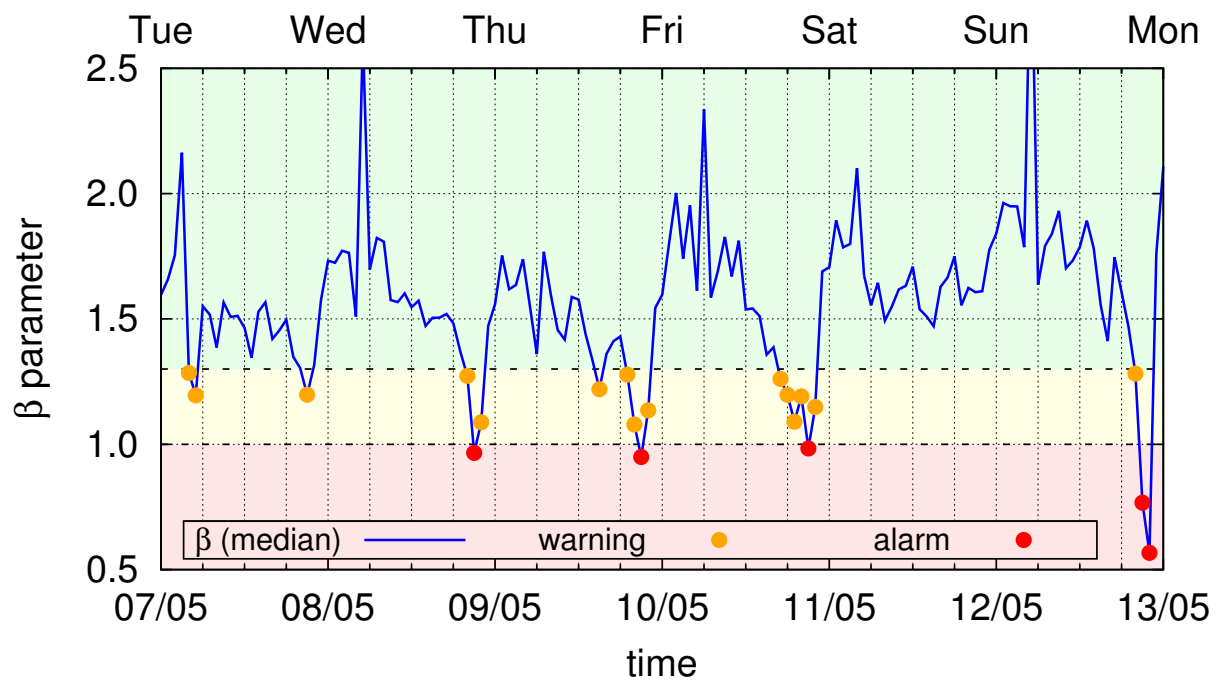


Figure 36: Median of β per hour for all YouTube flows.

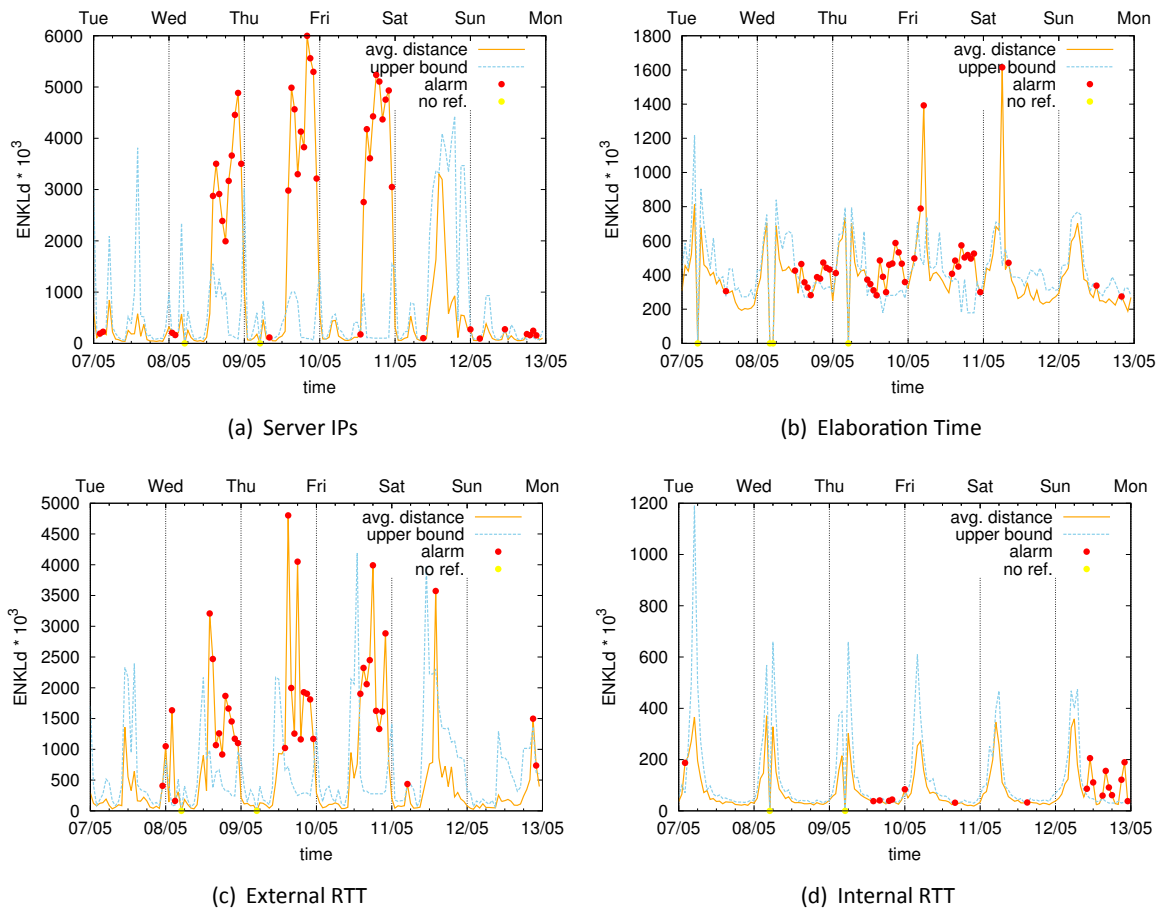


Figure 37: Output of the anomaly detection analysis module for the diagnostic signals in the YouTube anomaly. The anomaly is caused by a shift in the distribution of flows across server IPs. The analysis on the distribution of elaboration times and internal/external RTT complements the picture on the event and support the diagnosis process.

2.7 Verification and certification of service-level agreements

2.7.1 Data and experiments

To fully deploy this use case, it was carried a collaboration with Fastweb. We carried out only load and delay tests. In such a case, due to the fact that we investigate about a low bit rate line (10 Mb/s) we investigate on SLA measurements based only on TCP throughput and RTT by looking at the role of traffic load and delay tests. We (FUB, POLITO and Fastweb) carried out tests for a period of three months. The test were configured for SLA measure every 3 minutes, each test lasted a minimum of 10 seconds. The measurements on the network were carried one day "yes" and one day "no".

To have a better understanding of measures of SLA, we deployed also TSTAT probe for the passive measurement of each test, by integrating both probes with mPlane protocol. We analyzed all the measurements to identify the links that were presenting problems. The most problematic links were the ones that presented high congestion, and in this cases the measurement of the SLA showed a much less value than the available line capacity.

By analyzing the passive data, we could determine if one link has congestion or not, by first identifying a phase when this link is more stable, step that could be completed with the correlation between the active and passive data.

Deployed architecture

In Figure 38 it is shown the testbed used for this experiments. The active probes that make SLA measurement, based only on TCP and RTT, are deployed all over the ISP network. The active probes make measurements, or data transfer to a server that is located into the Internet. All the incoming connections at the server are passively monitored by TSTAT probe. The monitored data from all the probes, the active and passive data, are stored for latter analysis on a remote database.

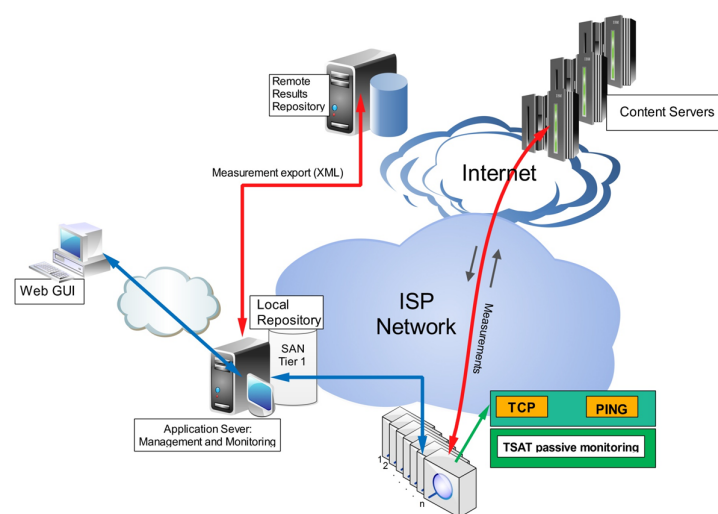


Figure 38: Schema of the measurement test bed in Fastweb for SLA measurement and passive monitoring.

2.7.2 Evaluation activity results

From all the analysis we show here only one case for the sake of simplicity. In figure 39 it is shown the SLA test, TCP throughput, of a congested link. The plot shows the tests over 24 hours. It can be noticed that during the night the line is more stable, and this is true since we have less traffic during the night. From around 6 a clock in the morning we can notice the starting of congestion, that worsened during the day.

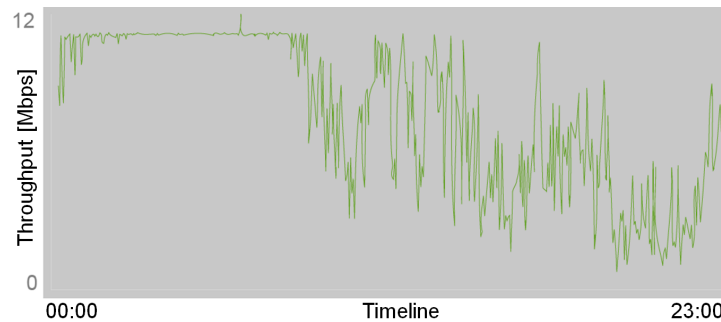


Figure 39: SLA test of a congested link

In figure 40 it is shown the distribution of the values of the throughput during 24 hours. There is a high variance of the throughput of this congested link, from 1 Mbps to 12 Mbps. This is out of our control, since we are on a network where we have production traffic, we can expect for some links to be congested. But if we understand when the link is congested than, we can flag those SLA measurement as SLA under congestion.

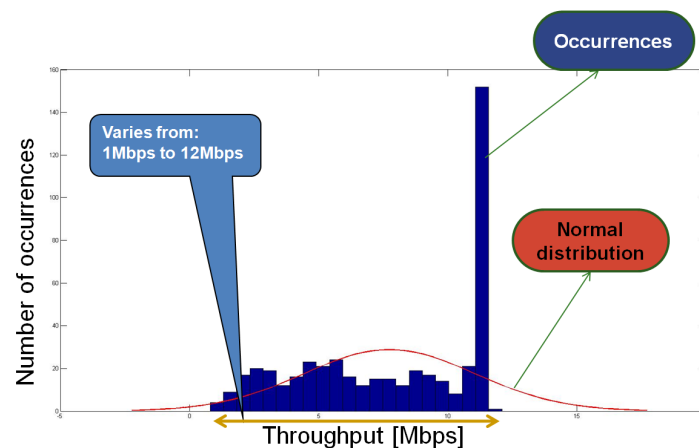


Figure 40: Number of occurrences of the throughput.

To better understand and flag this cases we have made the correlation between the active and passive measurement, this data it is shown in figure 41. For the correlation we have used the Spearman correlation coefficient, a linear correlation. In a linear correlation the value of the coefficient varies from -1 to +1, the sign means that the values are changing in the same direction if it is positive or in opposite direction if it is negative.

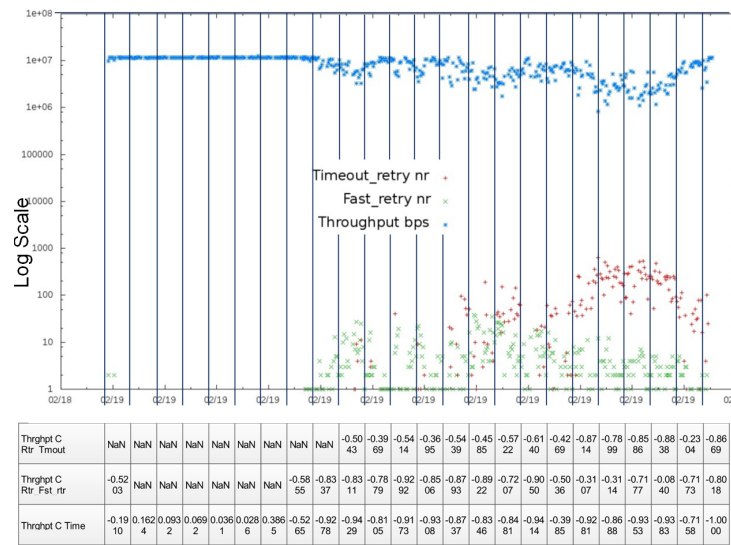


Figure 41: Correlation of Active and Passive data

From the correlation in figure 41 we can distinguish between all phases of the link. We can notice a first phase of the link where no congestion occurs, and the correlation between the throughput of the active measurement and the number of segment retransmitted during time out and during fast retransmit (data that we get from the passive measurement) it is null. With the presence of the congestion we can notice that the correlation between the throughput and the fast retransmit increases, not a very heavy congestion, but however it is present. This change in the correlation coefficient could be used as an alarm that signals the start of the congestion. Conversely on heavy congestion we can notice that we have a bigger correlation between the throughput and the time out. Also this could be used to identify when we are in a link with heavy congestion.

2.8 Publicly available datasets

All public dataset produced in the context of mPlane project are available through the mPlane site, in the Open Dataset section. For reader simplicity they are reported in the following table

Open Datasets

mPlane partner	Measure	Access type	Anonymous
FUB	Residential QoS: throughput, jitter, RTT, packet loss	Web registration	No
FUB	ISP QoS: throughput, jitter, RTT, packet loss	Web registration	No
ENST	QoS: Queuing delay, BitTorrent internet campaign		No
ENST	QoS: Queuing delay, Testbed validation		Yes
ENST	QoS: Fairness, LEDBAT+AQM experiments		Yes
ENST	QoE: Completion time, LEDBAT+BitTorrent		Yes
NETVISOR	IPSLA	Open	N.A.
ETH	ECN connectivity impairment	Open	Yes
ENST	Anycast: list IP/24, measurements, ground truth	Open	Yes

References

- [1] Z. Ben-Houidi, G. Scavo, S. Ghamri-Doudane, A. Finamore, S. Traverso, and M. Mellia. Gold mining in a river of internet content traffic. In *6th International Workshop on Traffic Monitoring and Analysis, TMA*, London, 04/2014 2014. Springer, Springer.
- [2] C. Callegari, M. Milanese, and P. Michiardi. Troubleshooting web sessions with CUSUM. In *TRAC 2015, IEEE 6th International Workshop on Traffic Analysis and Characterization, August 24-27, 2015, Dubrovnik, Croatia, Dubrovnik, CROATIA*, 08 2015.
- [3] P. Casas, A. D’Alconzo, P. Fiadino, A. Bär, A. Finamore, and T. Zseby. When youtube does not work - analysis of qoe-relevant degradation in google cdn traffic. *Network and Service Management, IEEE Transactions on*, 11:441–457, Dec 2014.
- [4] M. D. Mohamed Ahmed. A vocabulary for growth: Topic modeling of content popularity evolution. Technical report, Nov 2015. <https://www.dropbox.com/s/x13zkkvs85e696i/contentpredictionalgorithm.pdf?dl=0>.
- [5] H. Pinto, J. Almeida, and M. Goncalves. Using early view patterns to predict the popularity of youtube videos. In *Proceedings of the sixth ACM international conference on Web search and data mining (WSDM 2013)*, pages 365–374. ACM, February 2013.
- [6] A. Rufini, M. Mellia, E. Tego, and F. Matera. Multilevel bandwidth measurements and capacity exploitation in gigabit passive optical networks. *IET Communications*, 8:8, 11/2014 2014.
- [7] G. Scavo, Z. Ben Houidi, S. Traverso, R. Teixeira, and M. Mellia. Technical report: Mining HTTP logs on-line for network-based content recommendation. http://www.retitlc.polito.it/traverso/papers/webbrowse_tech-rep.pdf.
- [8] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin. Resurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference*, 2013.