mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Probe Measurement Primitives and Analysis Library - Initial Release

Author(s):	POLITO	Alessandro Finamore, Marco Mellia, Maurizio Munafò
	FUB	Edion Tego
	SSB	Gianni De Rosa, Stefano Pentassuglia
	TI	Fabrizio Invernizzi
	EURECOM	Marco Milanesio
	ENST	Pellegrino Casoria, Danilo Cicalese, Dario Rossi,
	NEC	Maurizio Dusi, Saverio Niccolini
	NETVISOR	B. Szabó, T. Szemethy
	TID	Giorgos Dimopoulos, Ilias Leontiadis
	FHA	Michael Faath, Rolf Winter
	ULG	Bennoit Donnet, Korian Edeline, Yongjun Liao
	ETH	Brian Trammel

Document Number:	D2.2
Revision:	1.0
Revision Date:	27 May 2014
Deliverable Type:	RTD
Due Date of Delivery:	1 May 2014
Actual Date of Delivery:	27 May 2014
Nature of the Deliverable:	(S)oftware
Dissemination Level:	Public

Abstract:

This deliverable collects the software released by the mPlane Consortium at month 18. This is a software deliverable, so this document briefly describes the software by collecting the information that is present on the website page at the time of writing. Software and instruction on how to access it must be accessed from `http://www.ict-mplane.eu/public/software`.

Keywords: mPlane software, proxy interface

Disclaimer

Contents

Executive Summary

This deliverable summarizes software that the Consortium made available at month 18 since the beginning of the project. The software that is part of this deliverable is listed below:

1. Blockmon
2. DATI
3. Fastping
4. Firelog
5. GLIMPSE
6. MobiProbe
7. PerformanceVisor
8. QoF
9. RILAnalyzer
10. Scamper
11. Tracebox
12. Tstat
13. probe-local-storage
14. youtube-probe

For each tool, a web page has been prepared following the same structure, and an archive is provided so to allow people to download and run the released software. Detailed information of what has been developed within mPlane is given.

Each tool page can be accesses from from `http://www.ict-mplane.eu/public/software`. The page lists more software that has been developed by mPlane partners which are not part of this deliverable. The list above includes software that either was entirely developed, or that received significant contributions and updates within the project.

1 Software description

This deliverable summarizes the software that has been initially released by the Consortium at month 18 since the beginning of the Project. Each software is described on the mPlane website, accessible from `http://www.ict-mplane.eu/public/software`. Such page(s) will change in the future, and other software will be made available as soon as new tools are released. Note indeed that the page already hosts other software that partners made available during the project, or some alpha release of tools that are not part of this deliverable. This deliverable is thus intended to be a snapshot of the status of the page at the time of writing. Each page follows the same scheme, so that a uniform description of the tools and software is offered. In particular, the following items are present:

- **Description**: a description of the software is provided, with one picture that helps to identify its scope and goals. In case the software has already a webpage describing it, we report in this section only a brief description and provide pointers to the original webpage(s) to avoid replicating data.
- **Quick start**: it provides a simple description on how to download, compile and run the software. Links to external repositories can be used, e.g., to `http://github.com` or to a private svn, in case the software is not natively hosted in the mPlane svn.
- **New features supported by the mPlane project**: here we list the new features that have been added during mPlane in case software has not been entirely developed within the project, but rather it has been extended during the project.
- **mPlane proxy interface**: Here information on how to use the mPlane native architecture interface is offered. The software can either natively support the mPlane interface, or a proxy has to be used to expose capabilities, exchange commands, and retrieve results. Note that the mPlane reference architecture is still a work in progress at the time of writing this deliverable and it will be detailed in D1.4 at month 24. Hence, information here is to be considered a preliminary release and are supposed to change in the future.
- **Official version**: here links to the frozen version of the software at the time of writing this deliverable are provided. For instance, in case the software is hosted in a public or private repository, a .tar.gz archive is provided with the timestamp of its creation.

In the following, we report the printed version of each tool description page. It has to be intended as a snapshot since the page itself is supposed to change in the future. The tools that are included are:

1. Blockmon
2. DATI
3. Fastping
4. Firelog
5. GLIMPSE
6. MobiProbe
7. PerformanceVisor

8. QoF
9. RILAnalyzer
10. Scamper
11. Tracebox
12. Tstat
13. probe-local-storage
14. youtube-probe

Other software present in the front page is not intended to be part of this deliverable. The list above includes software that either was entirely developed, or that received significant contributions and updates within the project. An alpha version of the mPlane Reference Implementation (RI) is present, but it is not part of this deliverable as it will be officially be part of D1.4.

Software

mPlane architecture:

- RI - Reference Implementation (developed by ETH, SSB)

Passive tools:

- Blockmon, a distributed stream-processing platform (developed by NEC)
- DATI, a flexible, high performance passive monitoring platform (developed by TI)
- RILAnalyzer, a tool to perform network analysis from within a mobile device (developed by TID)
- QoF, a TCP-aware flow meter (developed by ETH)
- Tstat, a passive monitoring tool (developed by POLITO)
- MobileProbe, a tool for monitoring smartphone performance for Android devices (developed by TID)

Active tools:

- DMFSGD: a tool to infer delay between distance hosts without performing any measurement (developed by ULG)
- GLIMPSE, an end host-based network measurement tool (developed by FHA)
- Fastping, a fast ICMP scanner for TopHat (developed by ENST)
- MERLIN, a router-level topology discovery tool (developed by ULG)
- mSLAcert, a tool for measurement of multi layer throughput and other active data for Service layer level agreement certification (developed by FUB)
- PerformanceVisor proxy, an universal mplane proxy module for the existing ''PerformanceVisor'' (PVSR) monitoring framework (developed by Netvisor)
- Scamper, a sophisticated active probing tool (developed by CAIDA)
- Tracebox, a tool for topology discovery (developed by ULG)
- youtube-probe, a lightweight active YouTube video download performance evaluation tool (developed by NETVISOR)

Mixing active and passive:

- Firelog: a Firefox plugin to measure HTTP QoE (developed by EURECOM)

Repository tools:

- Hadoop Fair Sojourn Protocol, a scheduler for Apache Hadoop (developed by EURECOM)
- Tstat - LogSync, a simple data export framework (deveoped by POLITO)

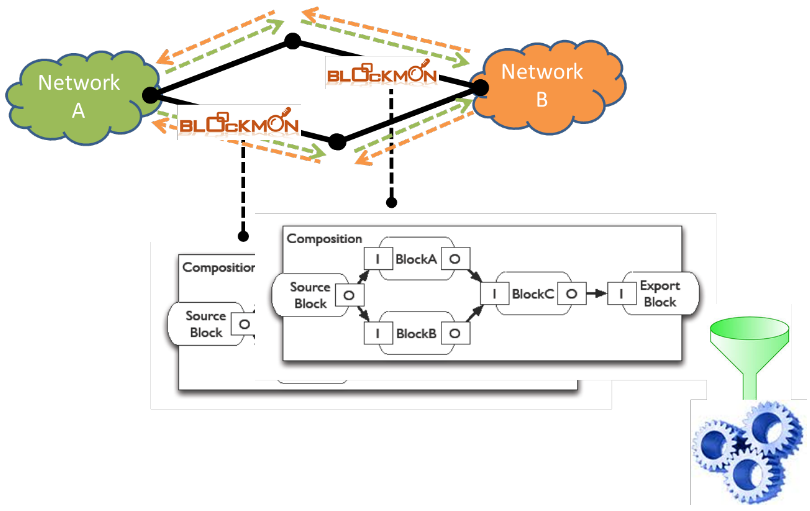
Others:

- IPFIX, an ipfix module for python3.3 (deveoped by ETH)
- probe-local-storage is a small-footprint in-memory DB for time-series data with local processing (developed by NETVISOR)

Blockmon

Description:

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



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

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

More information on Blockmon and its performance are provided in

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

and on the Blockmon official website.

Quick start:

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

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

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

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

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

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

        <block id="sniffer" type="PcapSource" invocation="async" threadpool="sniffer_thread">
            <params>
```

```
                <source type="live" name="eth0"/>
                <!--bpf_filter expression="!tcp"/-->
            </params>
        </block>

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

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

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

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

New features supported by the mPlane project

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

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

mPlane proxy interface

Coming soon.

Official versions

- May 8th, 2014 - frozen release for D2.2:

  ```
  svn checkout https://svn.ict-mplane.eu/svn/public/software/blockmon
  ```

- For the latest release, please always refer to the github repository

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

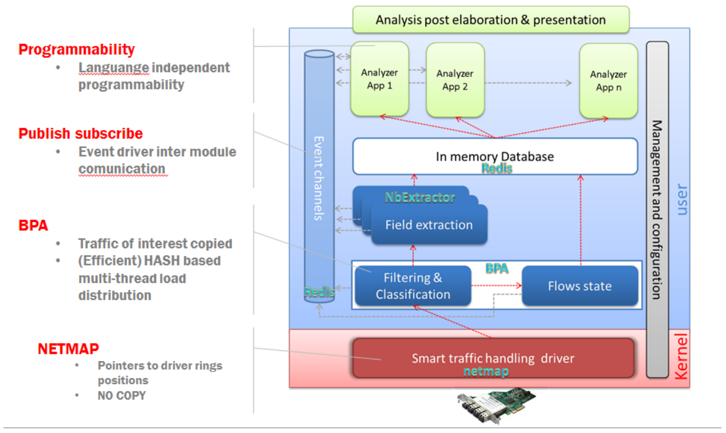
DATI

View Edit Revisions Log

Public page *DATI* has been updated.

Description

DATI is a flexible, high performance passive monitoring platform. Build on FreeBSD, it leverages on NetMAP fast and safe network driver to access network devices, reaching up to 14.88 Mpps with a CPU running at less than 1 GHz. The goal of the platform if to give developers an high level view of traffic of interest without the complexity of high speed traffic capture and classification details.
The system can recognize traffic flows from static BPF signatures and collect state information (volume, packets number and session duration) as well as application-level data based on an XML description of the protocol stack.


Programmability
Languange independent programmability
Publish subscribe
Event driver inter module comunication
BPA
Traffic of interest copied
(Efficient) HASH based multi-thread load distribution
NETMAP
Pointers to driver rings positions
NO COPY
Analysis post elaboration & presentation
Analyzer App 1
Analyzer App 2
Analyzer App n
Event channels
In memory Database
Redis
NbExtractor
Field extraction
BPA
Filtering & Classification
Flows state
Redis
Management and configuration
user
Smart traffic handling driver
netmap
Kernel

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

New features supported by the mPlane project

With the support the mPlane project some specific analyzer App have been written, enabling the platform to extract following measure and features

- TCP RTT delay
- Radius informations
- Routing protocols informations (OSPF, BGP, LDP)

mPlane proxy interface

The mPlane proxy interface for DATI is written in nodejs leveraging on the mPlane nodejs library.

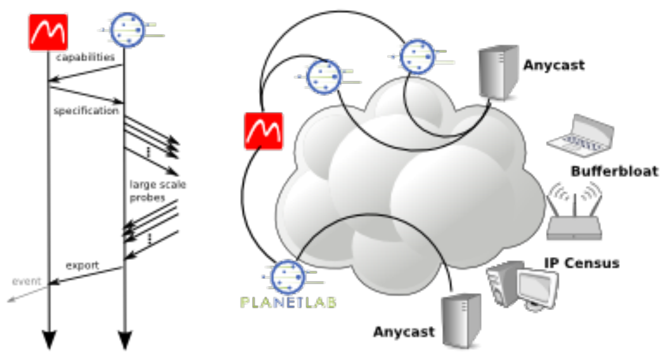
Fastping

Description:

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

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

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



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

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

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

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

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

Quick start:

Fastping is a python script and

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

Example of use of the tool as a standalone shell tool and as mPlane probe can be found in the software package (HOWTO.shell and HOWTO.mplane in the main directory respectively) as well as in the fastping documentation

New features supported by the mPlane project

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

- As of May 15th 201, only the mPlane client/server probe and the repository are implemented.
- The implementation of a supervisor for all fastping probes (i.e., PlanetLab and beyond) is a planned ongoing effort

mPlane proxy interface

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

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

mPlane custom registry

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

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

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

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

Official version

- May 25th, frozen release for D2.2

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

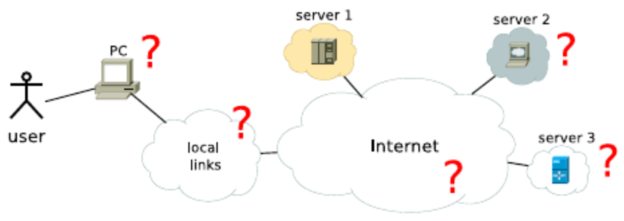
- We strongly recommend to use up-to-date release available through the github https://github.com/fp7mplane/protocol-ri/tree/fastping

Firelog

Description:

FireLog is a browser based tool to collect web page browsing data and users' subjective experiences. It is designed with the goal of helping end users to understand questions related to their web page browsing:

- Which/what (types of) web pages have poor experiences.
- What happens underneath the browser while user is feeling bad about their browsing.
- What happens to other user(s) for similar pages.
- What is the cause for the poor experienced web page.



Generally, FireLog system is composed by two parts: a client side capturing engine, and a centralized server repository. Currently, the client capturing engine is based on a lightweighted firefox extension. The capturing engine is installed at the end user and will capture key events during user's browsing. Meanwhile, it sends back the data to a FireLog server triggered by user's events such as closing an old tab/window, etc. The FireLog server is configured with Apache/PHP and a database system. Processing and diagnosis engine is also instrumented. More detailed description about its mechanism and architecture are available at the plugin web page.

Quick start:

- Download the tarball and follow the instructions on README.txt.

New features supported by the mPlane project

The current version of the probe is enanched with several features:

- The probe is divided into a passive part and an active one:
  - The passive part collects browser data during a browsing session
  - The active part performs active measurements (i.e., ping and traceroute) towards the contacted web servers
- A diagnosis column result is added. The probe locally performs a diagnosis to identify the root cause of a poor performance in the browsing session.
- A mix of raw and aggregated results are sent to the mPlane repository for further analysis.

mPlane proxy interface

The mPlane proxy inferface can be found on the branch "firelog" on mplane-github (see below).

To start the service, simply launch:

```
python3 firelog.py -4 local_ip -u url
```

and connect with the mPlane client.

Official version

- May 15th, 2014: frozen release for D2.2
  - Firelog prototype presenting all the functionalities [tarball] (no mPlane proxy, but standalone)
- Firelog mPlane interface [gihub]

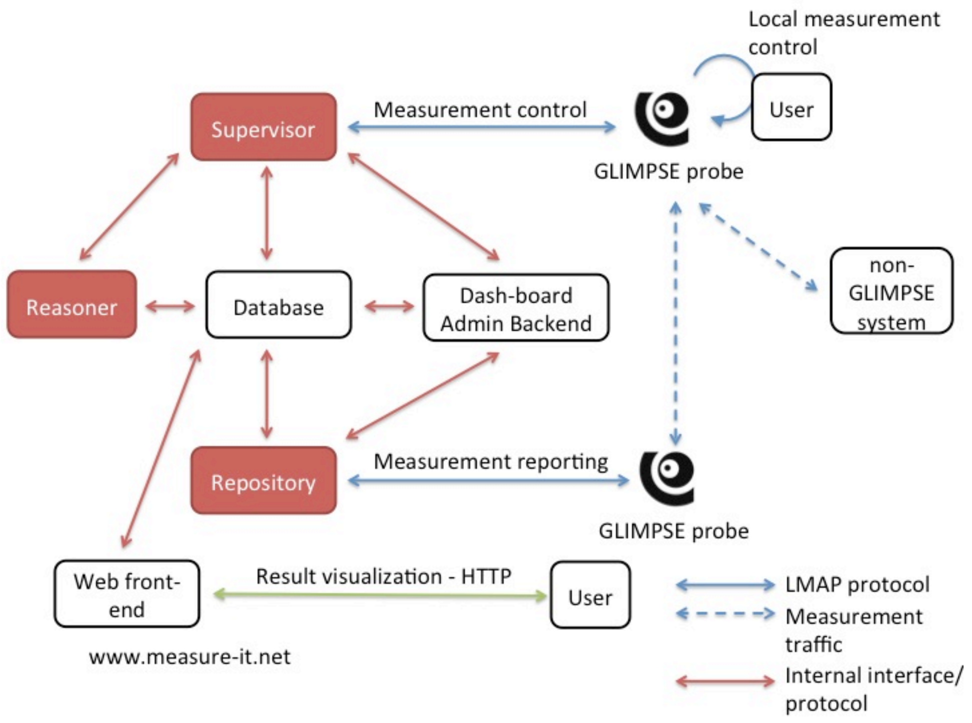
File:

901firelog.tar.gz

GLIMPSE

Description:

GLIMPSE is an end host-based network measurement tool. It is written in C++ with Qt and is cross-platform by design. The release version of GLIMPSE will be made available at http://www.measure-it.net at a later stage. The main focus of GLIMPSE is troubleshooting.



The source-code of GLIMPSE can be found in the public mPlane svn repository (see quick start section). This is a pre-release version which showcases the probe software with its measurement tools. To make sure this version can be used after changes to the GLIMPSE-backend the communication with the backend-servers is disabled in this version. The interfaces and back-end structures are under heavy development right now and change frequently.

Quick start:

You can check out the pre-release version here:

```
svn checkout https://svn.ict-mplane.eu/svn/public/software/glimpse
```

To compile the software you should download and install Qt and QtCreator from here. This version of GLIMPSE was tested with Qt 5.2.1. The requirements for the different platforms are:

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

Run QtCreator, open the client.pro from the source-code, add your Qt configuration and select the "mobile" project near the "Play" button. This launches the graphical version of the probe on the desktop computer or your mobile device (if configured). Login and registration are not necessary. You may also run the console-version of the probe, but as server communication and therefore receiving measurement tasks are disabled for this version the probe will just start and do nothing.

Usage

After starting the probe you see the login screen. Just click login without entering any data to get to the main menu. Measurements can be defined and started from the "Toolbox" tab. Results can be seen on the "Reports" tab (after a measurement has been executed). The event-log is shown on the "Home" tab for debugging purposes. As server communication is disabled in this version now schedules are shown in the "Schedule" tab.

New features supported by the mPlane project

GLIMPSE was completely developed as part of the mPlane project. The key features are

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

On-demand troubleshooting buttons and scheduled measurements are deactivated in the pre-release version (see below).

Measurement tools included as the time of this writing:

- Ping (UDP and ICMP)
- Traceroute (UDP)
- Bulk transfer capacity (TCP)*
- UPnP lookup
- Trains of packet pairs*
- HTTP download

You may run these measurements from the "Toolbox"-tab in the application. Measurements marked with an asterisk need another GLIMPSE probe as peer for the measurement. The default settings for these measurements are preconfigured with the address of another probe running at our own servers at the moment. As development goes on this probe may not be available anymore and you need to setup a second probe which is publicly reachable (this is going to change to allow NAT traversal) through port 5005 and 5006.

mPlane proxy interface

We are going to implement a proxy interface to make a GLIMPSE probe work within a mPlane architecture (supervisor with reasoner/repository): the probe has a wrapper to convert capabilities and specifictions received from a mPlane supervisor to the GLIMPSE data format, and the results are wrapped into the mPlane format before sending to a mPlane repository. The probes are also going to advertise their own capabilities in mPlane format.

These interfaces are not implemented yet.

Official version

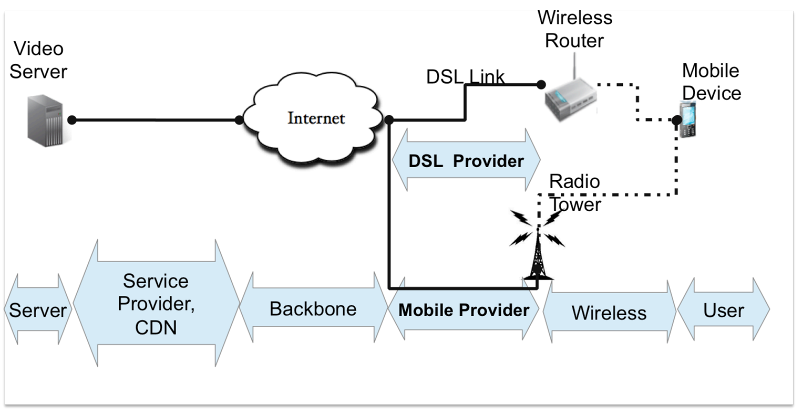
- May 15th, 2014 - frozen release for D2.2 [tarball] [svn]

File:
914glimpse.tar.bz2

Mobile Probe (Android)

Description:

The application periodically launches a YouTube video from a list of short and long videos, while it logs hardware, system and network measurements in the background. When a number of measurements becomes available, they are aggregated and sent to a remote server. For the network measurements the application uses a precompiled tstat binary for Android which is packed inside the apk file.



Collected Metrics:

- Hardware: CPU usage, free memory, RSSI (when on WiFi), cell tower information, location information from GPS and WiFi, connectivity state
- System: Playback state, re-buffering events, re-buffering duration, load time, HTTP requests, video decoder state
- Network: statistics per tcp flow as they are provided by tstat

Requirements:
The application requires root access and the pcap library to be present in /system/lib in order to allow tstat to listen on the network interface. It is also required to have the official YouTube application installed on the device.

Quick start:

Installation and first use:

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

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

General Usage:
From the main interface the user can monitor the status of the synchronisation with the server and the number of measurements and bytes sent.
The application's menu offers the options to Restart/Stop monitoring and completely Exit the application.

New features supported by the mPlane project

The probe was developed entirely within the mPlane project.

mPlane proxy interface

The proxy can be found here: https://github.com/fp7mplane/protocol-ri/tree/mobile_probe_tid

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

Official version

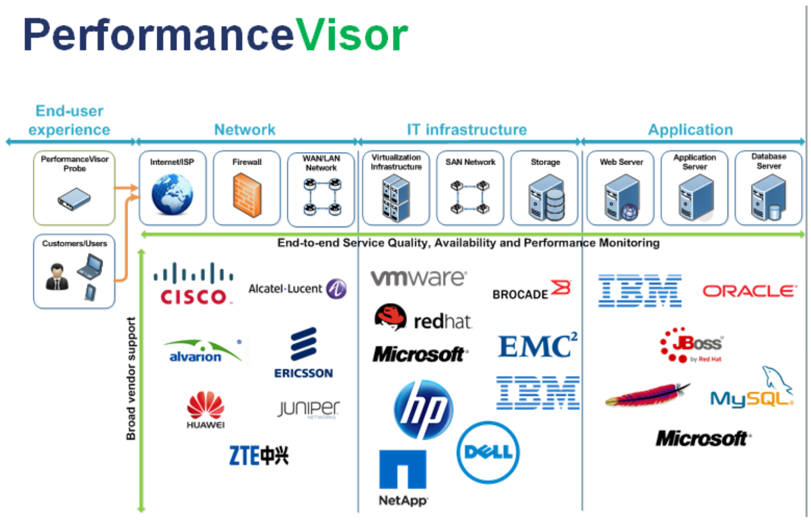
- May 15th, 2014, frozen release for D2.2: [tar.gz] [svn]

File:
921mobiprobe-youtubeprobe.tar.gz

PerformanceVisor proxy

Description:

PerformanceVisor (PVSR) is a commercially available monitoring solution from NETvisor Ltd. It is a unified platform for monitoring not just networks but also service quality and end-user experience, applications and various components of IT infrastructures. PVSR's measurement collectors support numerous open and vendor-specific monitoring protocols , as summarized by the following figure:



As part of the mPlane project we developed an universal mPlane proxy application for PerformanceVisor. It can be configured to map different measurement capilities from PerformanceVisor onto mPlane capabilities. The proxy turns PVSR into an mPlane repository and also into an mPlane probe, since it offers capabilities for the protocol verbs "*query*" and "*measure*":

- If the proxy receives a "*measure*" request for a measurement then first it checks whether the measurement already exists in PVSR's configuration or not. If it does not exists then it creates and starts the measurement, otherwise it updates the measurement configuration to match the parameters received (if necessary).
- If the proxy receives a "*query*" request for measurement results, then it looks up the measurement in PVSR's configuration, verifies that specified measurement parameters match, and then it answers the query.

The proxy supports the "*now ... future / period*" "*when*" specifications for "*measure*" and "*past ... now / period*" "*when*" specification for "*query*":

- The data collection cycle is specified by the period parameter, which means that it must be a valid PVSR data collection cycle period, for example 1m or 5m
- The duration must be an integer multiple of periods
- The beginning time is interpreted differently when the measurement does not exist in PVSR, because in this case enough time must be allowed to PVSR to update its data collector configuration. In order to do that the proxy waits additional minutes for the measurement results. Obviously this only applies to the "*measure*" specifications

Once the measurements are finished the proxy returns the measured values for each collection time during the measurement duration, i.e. the proxy returns multiple results for each specifications (except then duration = period).

Quick start:

Since PerformanceVisor is a commercially available product, first you have to request a demo from NETvisor Ltd. After you have the proper PVSR installation (you uploaded the license file and started all PVSR modules), you have to

- Download the proxy application from here https://github.com/fp7mplane/pvsr-proxy-probe
- Copy the "*mplane*" directory from the mPlane reference implementation into the same directory where you downloaded the proxy application
- The proxy application also needs *pvsr_soap_client* Python package which can be found on the PerformanceVisor server in the */opt/pvsr/lib* directory: just copy the *pvsr_soap_client.py* to the same directory where you downloaded the proxy

application

- Optional: if you want to use the default *pvsr_proxy_probe.cfg* then you will have to add these lines to the *mplane/registry.txt* file:

    pvsr.availability: natural

    pvsr.user: string

    pvsr.pwd: string

    pvsr.proxy: string

    pvsr.regex: string

    pvsr.count: natural

    pvsr.ipsla.config_entry: natural

The next step is the configuration if the proxy by editing the *pvsr_proxy_probe.cfg* file. The file format is JSON:

- soap: SOAP configuration for PVSR
    - *url*: the PVSR SOAP server URL
    - *wsdl_url*: the proxy downloads the WSDL from this URL
    - *user* and *password*: PVSR user and password information
- *logging*: the configuration file for the Python logging module
- *default_site*: the proxy can automatically create Jaga and Synthetic transaction equipments in PVSR if they do not exist. If it does so then it created them below this site. If the site does not exists then it creates the site as well. If the parameter is not specified then the proxy uses "mPlane"
- *delete_created_measurements*: if it is set to false then the proxy will not delete those measurement specification in PVSR which it created once the measurements are done
- *pvsr_default_conf_check_cycle*: if it is specified (in seconds) then the proxy assumes that the default measurement cycle is this in PVSR as well. Be aware that if you change this then you will have to reconfigure PVSR as well. The parameter only affect cases when the specified measurement does not exists in PVSR
- *measurements*: hash containing the available measurements through the proxy interface. The name will be used as the label of the capability as *<key>-measure* and *<key>-query*
    - *types*: the PVSR measurement types belonging to this capability. The key of the hash items must be a valid PVSR measurement type
        - "*first*" and "*second*": the mPlane registry name of the first and second PVSR measurement values.
        - *multiply*: optional parameter, integer. If specified then the proxy will multiply the result received from PVSR before forwarding to the mPlane client
    - *index_mplane_name*: optional. The mPlane parameter used as the Index parameter for the measurement in PVSR
    - *mplane_constants*: optional hash, the proxy places each parameter into the capability
    - *uda_name2mplane_name*: optional hash, the key is a PVSR measurement additional parameter name code, the value is an mPlane parameter name
    - *uda_constants*: optional hash, used only if the measurement does not exists in PVSR. The key is a PVSR measurement additional parameter name code
    - *equipment*: name of the PVSR equipment offering the measurements
    - *check_udas:* optional parameter. If set to false then the proxy do not check wether the values for the PVSR measurement UDAs match the values in the specification or not
    - *verb_measure:* optional parameter. If set to false then the proxy do not offer the verb "*measure*"
    - *verb_query:* optional parameter. If set to false then the proxy do not offer the verb "*query*"

The last step is to create the equipments in PVSR, however this step is not required for the Jaga and Synthetic transaction equipments.
The default configuration already contains several measurement specifications as examples:

- Jaga ping measurement
- HTTP availability check
- Cisco Ping
- Cisco IP SLA ping

New features supported by the mPlane project

The application allows a wide range of measurements to be imported into the mPlane architecture, already implemented in PerformanceVisor. The proxy component was developed entirely for mPlane.

mPlane proxy interface

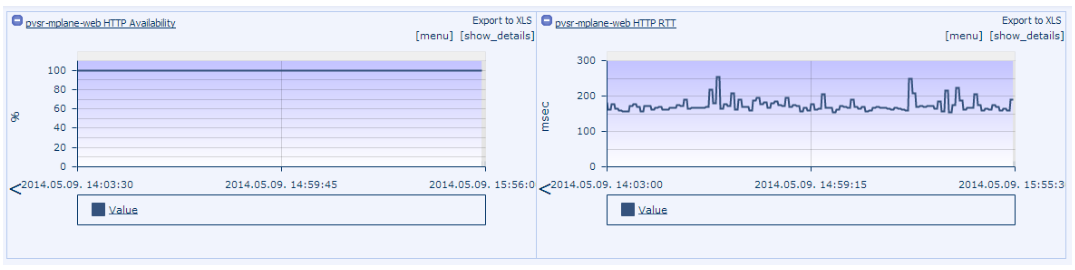
Since this is a generic proxy framework between mPlane and PVSR, the available capabilities, their parameters and results, etc… all depend on the *pvsr_proxy_probe.cfg* configuration file.

Examples

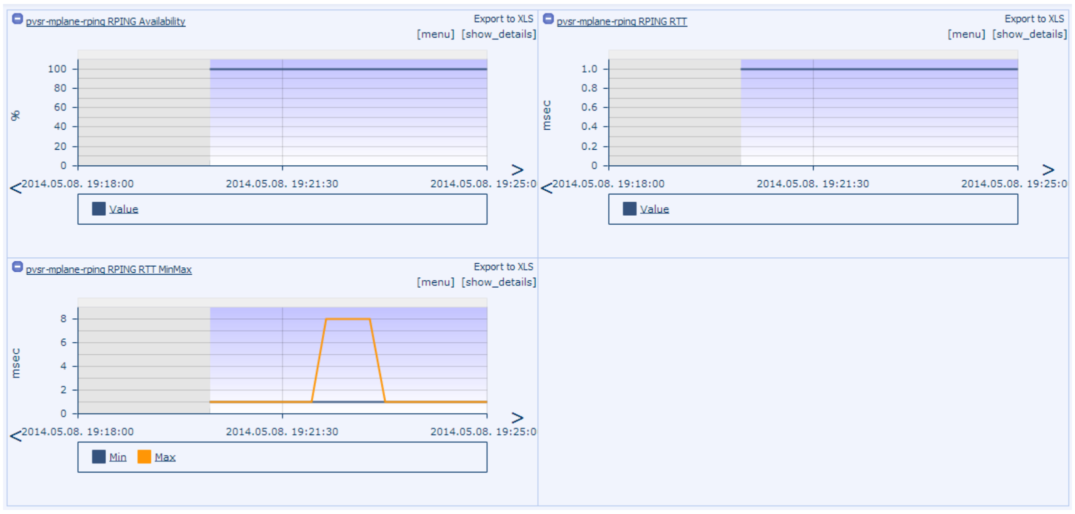
For sample specifications please have a look at this session log: https://www.ict-mplane.eu/sites/default/files//public/public-page/performancevisor-plugins//841examplecalls.txt

The PVSR measurement charts for this session:

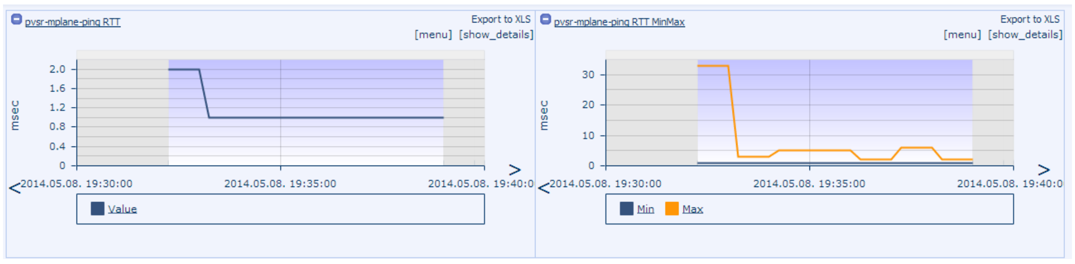
pvsr-mplane-web:



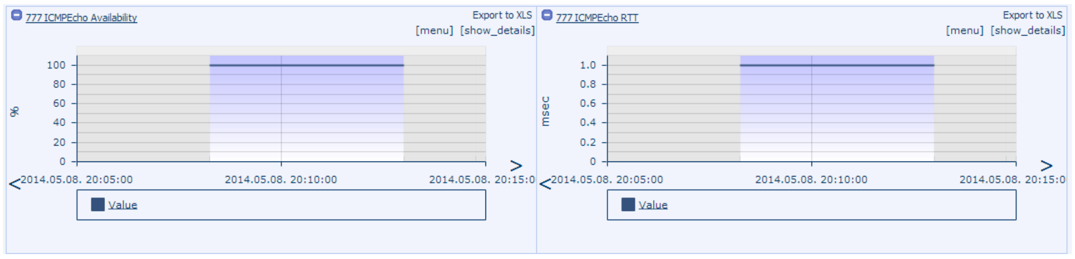
pvsr-mplane-cisco-ping:



pvsr-mplane-ping:



pvsr-mplane-cisco-ipsla:



Official version

- May 15th, 2014 stable version for D2.2
- Development version [github]

File:
Sample calls between the proxy and the reference implementation command line client
910pvsr-proxy-probe-master.zip

QoF

Description:

QoF is a TCP-aware flow meter, available at http://github.com/britram/qof.

QoF (Quality of Flow) is an IPFIX Metering and Exporting process, designed for passive measurement of per-flow performance characteristics.

QoF is primarily intended to support research into passive measurement of performance metrics for TCP flows; however, it can also be used for general flow measurement, especially in environments where the deployment of technologies which inspect packet payload is restricted. QoF is a fork of YAF version 2.3.2, with the following major differences from the YAF codebase:

- Removal of all payload inspection code.
- Replacement of packet acquisition layer with WAND's libtrace.
- Replacement of most command line flags with a YAML-based configuration file, which allows implicit feature selection through direct specification of the information elements to appear in QoF's export templates.
- Support for new information elements focused on passive TCP performance measurement.

QoF is licensed under the GNU General Public License, Version 2.

Quick start:

To install QoF:

1. Make sure you've got QoF's direct dependencies: libglib-2.0 (the GNOME C modernization layer) and libyaml and their headers. On Debian systems, install the `libglib-2.0-dev` and`libyaml-dev` packages.
2. Download and install libfixbuf from CERT; version 1.2.0 required.
3. Download and install libtrace from WAND at the University of Waikato. On Debian systems, this is in the `libtrace3-dev` package. Building libtrace requires bison and flex headers, as well.
4. Install QoF. This works the same as it does everywhere: `./configure --prefix=/some/where && make && make install`; if installing straight from the git working directory, use `autogen.sh` first. You may need the `--with-libtrace` flag to `./configure` if not installed in a system path. If installed to the same prefix as libfixbuf, the autotools script should automatically find it.

To run QoF, writing to an IPFIX file:

qof —yaml *yaml-file* —in *libtrace-uri* —out *ipfix-filename*

The *libtrace-uri* for a PCAP file named `foo.pcap` would be `pcapfile:foo.pcap`. Note that libtrace supports compressed trace files natively (e.g. `pcapfile:foo.pcap.gz`); see the libtrace documentation for more.

There's a sample *yaml-file* configuration file in the test directory. The most important configuration directive is `template:`, which lists the Information Elements which will be exported by YAF.

QoF includes a set of tools in Python for analyzing IPFIX output for research purposes; these are described in the GitHub wiki.

New features supported by the mPlane project

The entirety of the feature set added to QoF since it was forked from YAF in November 2012 was added with the support of the mPlane project.

mPlane proxy interface

Proxy interface development is ongoing, pending the development of an mPlane to IPFIX Information Element bridge interface.

Official version

- Release 0.9.0 of 4 November 2013 is available here: qof-0.9.0.tar.gz. The manual is here: qof.pdf.
- Users of QoF should track the master branch at github: https://github.com/britram/qof

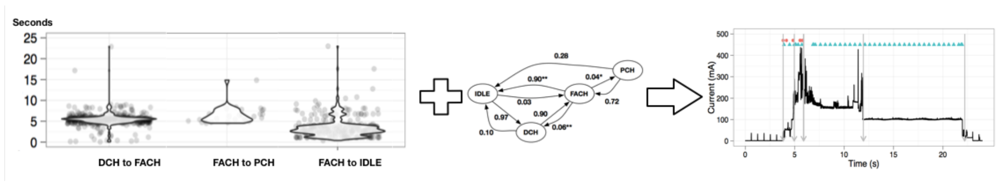
RILAnalyzer

Description:

RILAnalyzer: a tool to perform network analysis from within a mobile device, available at http://andriusa.github.io/RILAnalyzer

Modern smartphone platforms add new challenges for the cellular networks they are running on. Equally though, complexity of cellular networks is hard to deal with for application and system developers, worsening each other's performance and efficiency. Such difficulties are largely caused by the lack of cross-layer understanding of interactions between different entities - applications, devices, the network and its management plane.

To address the issue, we are releasing RILAnalyzer publicly. It is a tool that provides mechanisms to perform network analysis from within a mobile device. RILAnalyzer is capable of recording low-level radio information and accurate cellular network control-plane data, as well as user-plane data. Such data can be used to identify previously overlooked network and connectivyt management issues and infer how the different configurations interact with application logic, causing network and energy overheads.



Quick start:

You will need a rooted device to install the software. We tested it on Samsung Galaxy SII using CyanogenMod 10.1. You can find detailed instructions at the website (http://andriusa.github.io/RILAnalyzer)

New features supported by the mPlane project

RILAnalyzer was developed as a tool to perform network analysis from within a mobile device to address th emobile connectivity use case. Data collected by this tool are exported and analyzed by the mobileProbe.

mPlane proxy interface

Data are stored and synchronized to a mongoDB database via the mobileProbe. The mplane proxy can access the data through the mongDB capability extractor found here (still in progress). https://github.com/fp7mplane/protocol-ri/tree/mobile_probe_tid

Official version

- May 15th, 2014: You can download the version here: [tar]

Scamper

Description:

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

scamper comes with several probing utilities: ping, traceroute (various versions), Doubletree, tracebox, alias resolution, ...

scamper supports two types of output: text (on the standard output) and warts files (warts is a binary format for storing packets into the scamper system).

Quick start:

Building scamper from the source is very easy:

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

scamper can be run under Linux, Mac OS X, BSD.

scamper is a command line tool that can be launched as follows:

```
scamper -c -p -w -M -o -O -f
```

Options are the following:

- `-c command` specifies the command to use for scamper. Possible choices are delias, neighbourdisc, ping, trace, tracelb, sniff, sting, and tbit;
- `-p pps` specifies the target packets-per-second rate for scamper to reach. By default, this value is 20;
- `-w window` specifies the maximum number of tasks that may be probed in parallel. A value of zero places no upper limit. By default, zero is used;
- `-M monitorname` specifies the canonical name of machine where scamper is run. This value is used when recording the output in a warts output file;
- `-o outputfile` specifies the default output file to write measurement results to. By default, stdout is used;
- `-O options` allows scamper's behaviour to be further tailored. The only relevant choice here (for mPlane) is warts (it ouputs

  results to a warts binary file);
- `-f listfile` specifies the input file to read for target addresses, one per line, and uses the command specified with the -c option on each.

New features supported by the mPlane project

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

- tracebox support

mPlane proxy interface

Scamper's Tracebox mPlane proxy interface.

Official version

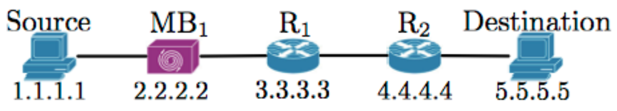
- May 15th, 2014: ADD TARBALL frozen release for D2.2

```
svn co https://svn.ict-mplane.eu/svn/public/software/scamper-tracebox
```

Tracebox

Description:

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



(a) topology

```
# tracebox -p 'IP / TCP / mss(9000)' -n 5.5.5.5
tracebox to 5.5.5.5 (5.5.5.5): 30 hops max
1: 3.3.3.3 TCP::SequenceNumber
2: 4.4.4.4 IP::TTL IP::CheckSum TCP::CheckSum TCP::SequenceNumber
           TCPOptionMaxSegSize::MaxSegSize
3: 5.5.5.5
```

(b) output

The figure above ((a) topology) shows a simple network, where MB_1 is a middlebox that changes the TCP sequence number and the MSS size in the TCP MSS option but that does not decrement the TTL. R_1 is an old router while R_2 is a router that is able to quote, in the returned ICMP message, the entire message that is responsible of the problem. The server always answer with a TCP reset. The output of running tracebox between ``Source'' and ``Destination'' is given by the below part of the figure ((b) output). The output shows that tracebox is able to effectively detect the middlebox interference but it may occur at a downstream hop. Indeed, as R_1 does not quote the full packet, tracebox can only detect the TCP sequence change when analyzing the reply of R_1. Nevertheless, when receiving the full message quoted from R_2, that contains the complete IP and TCP header, tracebox is able to detect that a TCP option has been changed upstream of R_2. At the second hop, \tracebox shows additional modifications on top of the expected ones. The TTL and IP checksum are modified by each router and the TCP checksum modification results from the modification of the header.

Tracebox comes in three flavours:

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

Quick start:

Original Software

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

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

Tracebox requires:

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

To build Tracebox:

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

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

Scamper Port

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

Source can be found at http://www.caida.org/tools/measurement/scamper/.

To build Scamper:

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

The Scamper Tracebox implementation is the most complete and efficient. It involves different options to modify the text output format (e.g: traceroute-like output vs simplified middlebox locations only output, add the ICMP quoting size standard used by each hop, ...). It also contains other non-tracebox middlebox detection methods callable through the following arguments:

- --proxy detects potentiel proxies between host and destination.
- --statefull detects statefull middleboxes between host and destination.
- --frags path packet fragmentation allowance.

Android Port

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

To install tracebox Android:

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

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

mPlane proxy interface

The current mPlane Tracebox interface contains 3 capabilities (each one subdivided for IPv4/IPv6):

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

New features supported by the mPlane project

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

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

Official version

- May 15th, 2014: frozen release for D2.2 [tarball] [svn]
- Development version [github]

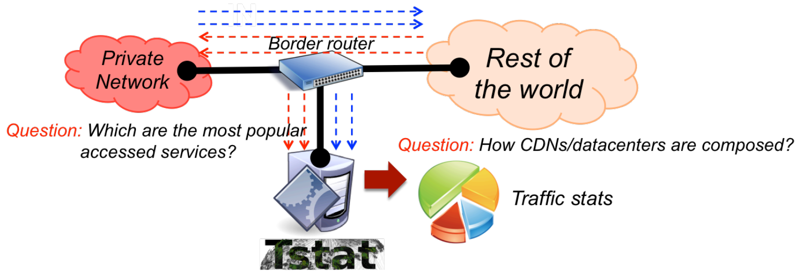
File:

915scamper-tracebox-20-05-14.tar.gz

Tstat

Description:

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


Private Network
Border router
Rest of the world
Question: Which are the most popular accessed services?
Question: How CDNs/datacenters are composed?
Traffic stats
Tstat

As shown in the figure, Tstat can produce real time analysis processing live traffic acquired from both standard PC NICs (using the libpcap) or DAG cards. Moreover, it can also process previously recorded packet-level traces of various formats (libpcap, snoop, etherpeek, etc.).

Tstat supports different output formats:

- **log files**: text files collecting per-flow stats. Each file is related to a different protocol or application (e.g., TCP/UDP traffic, video streaming, etc.);
- **histograms**: a set of histogram, each one related to a specific traffic features (e.g., incoming/outgoing IP bitrate, number of TCP flows, etc.). Statistics are collected over a time window (by default 5 minutes) and saved in separated text files;
- **RRD**: a set of histograms saved in RRD format
- **pcap**: dump files

Quick start:

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

```
svn checkout http://tstat.polito.it/svn/software/tstat/branches/mplane/
```

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

```
./autogen.sh
./configure
./make
```

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

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

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

For more information, please refer to the official website http://tstat.polito.it

New features supported by the mPlane project

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

- **HTTP module**: it allows to save text log files reporting information about HTTP queries/responses
- **IP address anonymization**: it allows to mask local IP address monitored using hashing functions
- **Blockmon integration**: we collaborated with NEC to integrate Tstat analysis modules in Blockmon

- **log_sync:** a client/server application which allows to continuously export from Tstat logs from probes to repositories
- **improved log configurability**: rather than collecting a monilitich set of stats, Tstat now offers more fine-grained control on which set of features a saved in the logs. Per-flow stats are now grouped in macro classes which can be added or removed at runtime
- **Android integration**: thanks to the effort of TID, Tstat works also on rooted Android devices
- **OpenWRT integration**: thanks to the effort of NETVISOR, starting from release 37196, the OpenWRT Linux distribution contains Tstat as a "Network Utilities" package

mPlane proxy interface

Tstat mPlane proxy interface is available on github

To test the software:

- **STEP1**: run Tstat

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

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

  This exectute Tstat in live capture from the eth0 interface, using the default runtime configuration, and saving the logs in /tmp/tstat_output_logs

- **STEP2**: run the Tstat mPlane proxy

  ```
  >>> git clone https://github.com/fp7mplane/protoco-ri.git tstat_proxy
  >>> cd tstat_proxy
  >>> git checkout tstat
  >>> python3 mplane/tstat_proxy.py -T path/to/tstat/runtime.conf -c conf/client-certs.conf
  ```

  This will create a secure enabled mPlane service listening on 127.0.0.1 and port 8888

- **STEP3**: run the mPlane client and schedule an experiment

  ```
  >>> python3 mplane/client.py -c conf/client-certs.conf
  >>> |mplane| connect https://locahost:8888
  >>> |mplane| when now + 5m
  >>> |mplane| runcap 1
  ```

  This connect the client to the Tstat mPlane proxy, and schedule to collect the TCP End-to-End features provided in the log_tcp_complete for 5 minutes

- **STEP5**: check for results
  - Tstat provides on messages on stdout reflecting the variation of the runtime configuration

    ```
    [Wed May 14 23:44:14 2014] TCP log level set to 1 (Core + End_to_end)
    [Wed May 14 23:44:14 2014] created new outdir tstat_output_logs/2014_05_14_23_44.out
    [Wed May 14 23:49:30 2014] TCP log level set to 0 (Core)
    [Wed May 14 23:49:30 2014] created new outdir tstat_output_logs/2014_05_14_23_49.out
    ```

For further information please refer to the description provided in github, the official Tstat website and its SVN documentation

Official version

- May 15th, 2014 - frozen release for D2.2
  - tstat v3.0 [tar.gz]
  - tstat mPlane proxy [tar.gz]

File:
858tstat-3.0.tar.gz
859tstatproxy.tar.gz

probe-local-storage

Description

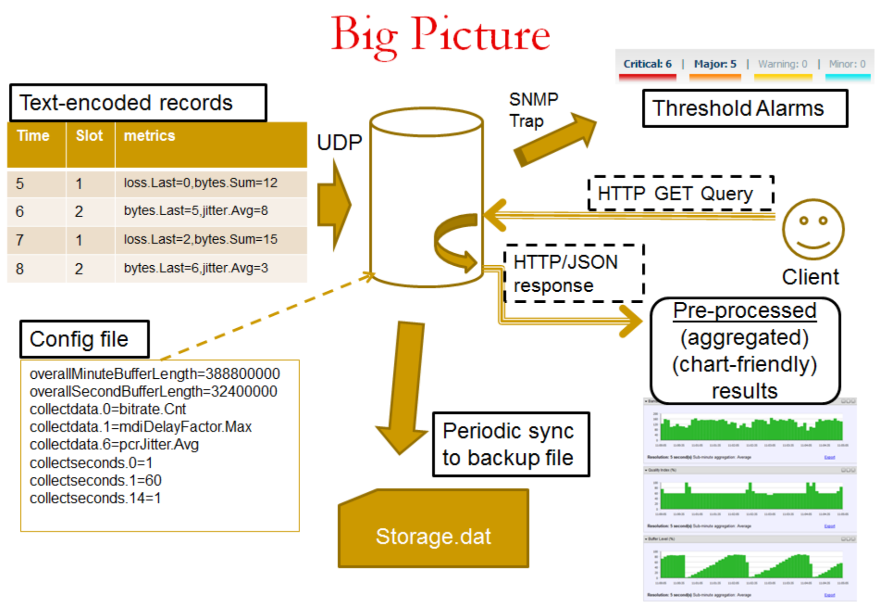
probe-local-storage is a software module for lighweight probes, implementing an in-memory database for time-series data (integer-valued metrics sequences). The module is developed in C++ with minimal external dependencies, suitable for embedded environments

Features include:

- In-memory DB for time-series numerical data with
  - Statistical/Aggregation functions to support pre-processed data retrieval and charting
  - (experimental) RLE support for efficient memory usage
  - Threshold alert evaluation (under development)
  - Perodic backup to volatile storage (SD card)
  - HTTP/SOAP/JSON interface (SNMP under development)
  - On-line reconfiguration
  - Small memory consumption

Intended usage:

- store 1sec resolution time series data and their n*60 sec aggregations
- up to a few days/weeks for a few dozen metrics or *measurement slots (channels)*
- in a few dozen MBs of memory
- provide charting-friendly aggregated and pre-processed data over HTTP/JSON query interface


Big Picture
Critical: 6 | Major: 5 | Warning: 0 | Minor: 0
Threshold Alarms
SNMP Trap
UDP
HTTP GET Query
Client
HTTP/JSON response
Pre-processed (aggregated) (chart-friendly) results
Periodic sync to backup file
Storage.dat

Text-encoded records

Time	Slot	metrics
5	1	loss.Last=0,bytes.Sum=12
6	2	bytes.Last=5,jitter.Avg=8
7	1	loss.Last=2,bytes.Sum=15
8	2	bytes.Last=6,jitter.Avg=3

Config file

```
overallMinuteBufferLength=388800000
overallSecondBufferLength=32400000
collectdata.0=bitrate.Cnt
collectdata.1=mdiDelayFactor.Max
collectdata.6=pcrJitter.Avg
collectseconds.0=1
collectseconds.1=60
collectseconds.14=1
```

An overview about probe-local-storage with furtjher details can be found in the file *probe-storage-slides.pdf* within the source.

probe-local-storage is meant to be used as a daemon process.

Quick start

probe-local-storage depends on the following software libraries:

- net-snmp-devel (net-snmp 5.x)
- libjson-c-dev
- libreactor, libaggregation (developed by NETvisor, also provided within the project)
- the uci configuration file library, (source also provided with the project for convenience)

Compilation and quickstart: follow the included *QuickStart.txt* file, which shows how to

- compile the code (with dependencies)
- configure the database for accepting records
- sample queries to verify basic operation

New features supported by the mPlane project

The current, open-source version of *probe-local-storage* is a complete rewrite of an earlier commercial version.

Official version

- currently in the project SVN at https://svn.ict-mplane.eu/svn/private/software/probe-local-storage
- Software release as of May 15th, 2014: probe-local-storage-r1287.tgz

File:
899probe-local-storage-r1287.tgz

youtube-probe

Description:

The probe module downloads a YouTube video (specified by the 11-character *video_id*), performs a lightweight playback emulation (frame header parsing), and calculates the following metrics:

Metric	**Description**
rebuffer.counter	number of video stalls during playback emulation
delay.buffering.ms	total time spent buffering (initial and rebufferings)
octets.layer7	number of video bytes downloaded
delay.download.ms	total video download duration
delay.urlresolve.ms	time needed to resolve video URL (by youtube-dl module)
delay.srvresponse.ms	delay of 1st reply packet after HTTP request
bandwidth.min.bps	worst bitrate (of 1sec intervals during the download)
bandwidth.max.bps	best bitrate (of 1sec intervals during the download)
bandwidth.avg.bps	average video download bitrate

The probe terminates playback emulation when the download is complete, i.e. usually it finishes sooner than the video duration (the probe supports bandwidth throttling via the underlying CURL library).

The probe queries the FLV (YouTube format code *5*) version version of the video, as it is the only format all YouTube videos are available in. Extending the probe to support the new DASH format promoted by YouTube is underway.
Table Comparison of YouTube media encoding options on Wikipedia describes more details on the FLV and other YouTube video formats.

The software contains an FLV processing module that originates from the public domain flvlib, ported to python3 and enhanced by NETvisor.

youtube-probe is available on github, temporarily as a branch of the mPlane protocol Reference Implementation project.

Quick start:

First, check and satisfy the following software dependencies youtube-probe relies on:

- pyCURL module (ver. 7.19.3.1 tested) is used to download the video content
- youtube-dl (ver. 2014.04.04.7 tested) python module and CLI is used to obtain the URL for the YouTube video id
- all dependencies required by the mPlane Reference Implementation framework
    - pyton3.3 with the following modules:
        - pyyaml
        - tornado

Testing the probe natively (not via the mPlane interface)

The yp-test utility is provided for testing the probe directly, providing a very simple CLI:

```
Usage: yp-test.py [<video-id>=riyXuGoqJlY]

YouTube Download Test Client

Options:
 -h, --help show this help message and exit
 -b BWLIMIT, --bwlimit=BWLIMIT
 limit download bandwidth to BW kBps
 -v, --verbose be more verbose, each -v increases verbosity
```

A test run for a sample video with bandwidth throttling:

```
(mplane)tiv@mplane:~/youtube-probe$ python yp-test.py -b 25000 EJQsBL6vEk4
[INFO] 0.007 Limiting pycurl bandwidth to 25000
[INFO] 0.007 YouTubeClient of video_id EJQsBL6vEk4, metrics: {'delay.srvresponse.ms': 0, 'octets.layer7': 0,
'bandwidth.min.bps': 0, 'rebuffer.events': 0, 'bandwidth.avg.bps': 0, 'bandwidth.max.bps': 0, 'delay.buffering
'delay.urlresolve.ms': 0, 'delay.download.ms': 0}
[INFO] 0.008 Query for media URL
[INFO] 1.572 URL extacted, starting download
[INFO] 1.829 Player starts BUFFERING
[INFO] 3.867 3 seconds of media buffered, starting playout
[INFO] 3.867 Player starts PLAYING, buffered: 3.971 secs
[INFO] 62.092 Player stalled at 58.225 secs of media
[INFO] 62.092 Player starts REBUFFERING
[INFO] 63.227 3 seconds of media buffered, starting playout
[INFO] 63.227 Player resumes PLAYING, buffered: 3.058 secs
[INFO] 102.467 Player stalled at 97.464 secs of media
[INFO] 102.467 Player starts REBUFFERING
[INFO] 104.815 3 seconds of media buffered, starting playout
[INFO] 104.815 Player resumes PLAYING, buffered: 3.937 secs
[INFO] 158.336 Player stalled at 150.984 secs of media
[INFO] 158.336 Player starts REBUFFERING
[INFO] 159.608 3 seconds of media buffered, starting playout
[INFO] 159.608 Player resumes PLAYING, buffered: 3.165 secs
[INFO] 196.646 Player stalled at 188.021 secs of media
[INFO] 196.646 Player starts REBUFFERING
[INFO] 199.165 3 seconds of media buffered, starting playout
[INFO] 199.165 Player resumes PLAYING, buffered: 3.938 secs
[INFO] 224.015 Download finished
Done: YouTubeClient of video_id EJQsBL6vEk4, metrics: {'delay.srvresponse.ms': 256.65926933288574, 'octets.lay
'bandwidth.min.bps': 0.0, 'rebuffer.events': 4, 'bandwidth.avg.bps': 199328.9885519995, 'bandwidth.max.bps': 3
'delay.buffering.ms': 9.3124361038208, 'delay.urlresolve.ms': 1564.445972442627, 'delay.download.ms': 222442.5
```

New features supported by the mPlane project

The probe was developed entirely within the mPlane project.

mPlane proxy interface

youtube-probe supports the native mPlane probe interface

Official version

- May 15th, 2014, frozen release for D2.2 [tarball]
- Development version available on github at https://github.com/fp7mplane/protocol-ri/tree/youtube-probe

File:

900protocol-ri-youtube-probe.zip