mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Basic Network Data Analysis

Author(s):	EURECOM	P. Michiardi, A. Barbuzzi (ed.)
	POLITO	A. Finamore, S. Traverso,
		D. Apiletti, E. Baralis, T. Cerquitelli, S. Chiusano, L. Grimaudo
	FUB	A. Rufini, A. Valenti, F. Matera
	NEC	M. Dusi, M. Ahmed
	NETVISOR	T. Szemethy, L. Németh, R. Szalay
	TID	Ilias Leontiadis, Yan Grunenberger
	FTW	P. Casas, A. D'Alconzo, A. Bär
	ENST	D. Rossi, Y. Gong

Document Number:	D3.1
Revision:	0.2
Revision Date:	1 May 2013
Deliverable Type:	RTD
Due Date of Delivery:	1 May 2013
Actual Date of Delivery:	1 May 2013
Nature of the Deliverable:	(R)eport
Dissemination Level:	Public

Abstract:

This document describes the requirements, input, output for the algorithms needed to perform analytic tasks on a large amount of data, in the context of WP3. Starting from the use cases defined in WP1, we identify the algorithms needed to address the various scenario requirements. Operating on a large amount of data, these algorithms strive for parallel and scalable approaches; the designing and implementation of the algorithm itself can be a challenging research task since today very little is known concerning how to develop efficient and scalable algorithms that runs on parallel processing frameworks. The algorithm in the storage layer are characterized by the fact that they operate on a large amount of data, and produce a concise representation of it, extracting features and aggregating it, so that the produced output is easier to handle and understand. Depending on the amount of data produced, on the scenario characteristics and on the time constraints, algorithms can require a real time (or near real time) or a batch processing. For each algorithm and use case, the input data and the initial state, the computation to run and the output produced are described.

Keywords: algorithms, storage, big data

SEVENTH FRAMEWORK PROGRAMME

Disclaimer

The information, documentation and figures available in this deliverable are written by the mPlane Consortium partners under EC co-financing (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission.
The information in this document is provided ``as is'', and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.

Contents

Document change record

Version	Date	Author(s)	Description
0.1	10 April 2013	A. Barbuzzi (EUR)	initial draft
0.2	29 April 2013	A. Barbuzzi (EUR)	draft

1 Introduction

This document defines and describes basic algorithms that operate on the data storage layer, generally on very large amounts of data. The repositories expect to receive input data from a multitude of probes, defined in WP2. This data is processed, analyzed and aggregated in the storage layer using parallel and scalable frameworks, and later served to the Reasoner, defined in WP4, which makes use of it.

WP3 deals with the storage of data from the probes, which produce different kind of data: *big data*, *real time data*, and possibly *small data*. Some of this input data requires further processing in WP3, some of it could be even served to WP4 directly. First, WP3 manages the storage of such data. Different technologies can be used, such as traditional DBMS, distributed DBMS, Hadoop File System (HDFS), HBase, and so on. Some execution engines (e.g., Hadoop, HBase, BlockMon, ...) will use an underlying storage engine or access directly to a data stream and execute two main classes of algorithms, according to the requirements of particular use-cases: real time algorithms and batch processing algorithms. Real Time algorithms should be applied to data as soon as it arrives, results are produced as soon as possible, and they generally guarantee a response within strict time constraints. On the other hand, batch algorithms are applied to the data with no stringent constraints on completion times. They are delay tolerant (as they operate on large volumes of data) and I/O (disk and network resources of the computing cluster) plays a crucial role; they are computational hungry.

Algorithms produce data that is consumed by other algorithms (in WP3) or by the Reasoner (WP4). The Reasoner, by construction, accepts only data of small size. Therefore, the algorithms typically aggregate data and extract a set of features conveying a summary of information and whose size is more convenient and feasible to be treated.

The algorithm definition is use case driven and follows the use cases described in WP1. Therefore, for each scenarios defined in WP1 and described in Section 3 of [2], the required algorithms are described; for each use case, we identify the available input produced by the probes, the algorithms necessary to process it and the produced output. Note that, while the description of the algorithms is related to WP3's tasks and topics, a detailed description of the input and how it is produced is related to WP2, while the consumption of the output concerns the Reasoner and WP4.

This document is organized as follow. First, a brief introduction is made in Chapter 1. In Chapter 2, the algorithms are discussed and analyzed, focusing on the input, on the output produced and on the algorithms themselves. Finally, Chapter 3 makes a general summary and draws the conclusions.

2 Algorithms Description

This chapter describes the algorithms used in each specific scenario. Each section contains the following items:

- an **Introduction**, which briefly describes the algorithms motivation and requirements;
- an **Input Description**, which describes the data available for the algorithms, produced by the probes and stored in the storage layer;
- an **Algorithm Description**, which illustrates the algorithms needed to analyze the data and to fulfill the use case requirements.
- an **Output Description**, which describes the output produced by the listed algorithms.

Note that the output of the algorithm can be stored in the storage layer, and as such can be used as input for other algorithms, allowing the creation of a pipeline of algorithms. For the complete scenarios' descriptions, please refer to Section 3 of Deliverable D1.1[2].

Note that both the algorithms and the Input/Output described in this document consist of an initial definition and they could possibly change and be refined during the project evolution. They are meant to provide the high level description of capabilities and data that the storage layer has to deal with. Engineered solution must be able to cope with these requirements and offer the scalability and flexibility needed to cope with the described constraints .

2.1 Supporting DaaS troubleshooting

2.1.1 Introduction

This section provides an overview of the algorithms we intend to focus on as for detecting the Quality of Experience of users accessing content using Desktop-as-a-Service (DaaS) solutions through thin-client connections, as presented in Section 3.1 in Deliverable D1.1[2]. Together with the algorithms, we describe the input features we plan to gather from the connections and the expected output of the algorithms, which will be exploited by the reasoner for taking further actions.

Thin-client solutions allow users to connect to remote services and access content offered by a virtual PC as if they had physical access to the remote machine. The reasons behind deploying thin-client solutions and hosting virtual PCs in datacenters are related to easiness of maintenance, cost optimizations and security. Although thin-client solutions were firstly designed for LAN environments, where bandwidth and delays between users and their virtual PCs are generally not an issue, with the advent of the cloud paradigm, hosting virtual PCs in remote datacenters poses challenges in terms of whether or not the network has enough resources to sustain a good Quality of Experience (QoE) for end users also in WAN environments.

As users perform real-time activities through thin-client protocols, e.g., reading and writing emails, web browsing and watching video content, the responsiveness of the network becomes a crucial parameter to determine the QoE perceived by end-users when interacting remotely with their virtual PC. Knowing the applications that users are currently running on their virtual PC helps shedding light on the QoE that they perceive given the actual network conditions: the entity of a bottleneck in the network may be tolerable for a user that is using the thin-client solution for reading a document on its virtual PC, whereas it may be unacceptable for a user watching multimedia content.

2.1.2 Input Description

Here we provide a description of the intended features that we plan to collect from probes about the single thin-client connection. Those features will represent the input to the algorithm for detecting the kind of application being run on top of the thin-client connection.

As thin-client connections usually make use of encryption to protect the content being exchanged, in the first place we plan to collect IP-level features of the packets of the connections, such as packet size, rate, inter-arrival time, and TCP-level features such as payload length and number of observed packets, whether they carry data or acknowledge only, TCP flags, etc. These features will be collected on a per-connection basis, i.e., on a per-thin client basis.

The idea is to have the probe to collect such information about the connections as the connections are active and periodically exporting their statistics. Tuning the timer based on which exporting the information will be investigated: this timer needs to be a good balance between classification accuracy and ensuring a continuous and timely classification.

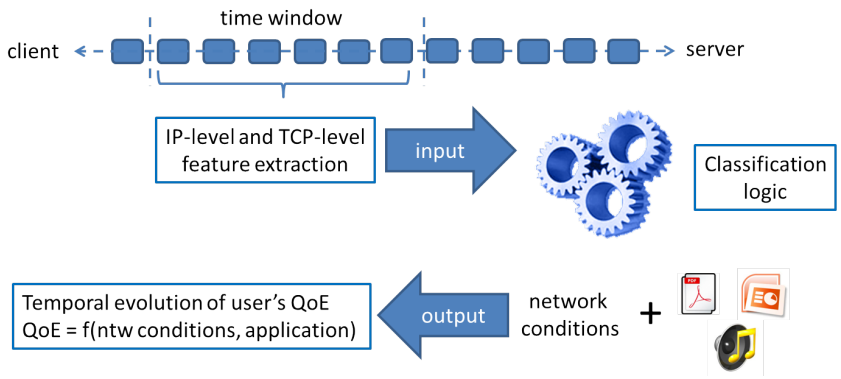

Figure 1: Overview of the algorithm for supporting DaaS troubleshooting. Input features and output data are shown.

2.1.3 Algorithms Description

The idea is to consider statistical classification techniques to detect on-the-fly the application that runs inside the thin-client protocols, by passively observing features of packets of the thin-client connection. The main goal here is the design of an effective statistical classification technique which can effectively take advantage of the available features provided by the probes as described in the previous section.

Given that information, we combine them with the actual network conditions along the path, which may affect users experience when using thin-client protocols: an example of such metrics is the Round Trip Time (RTT) and the available bandwidth. A threshold-based algorithm will be able to infer users' QoE, and its design and tuning we envision to be one of the main challenge. Possibly, more complicated classification algorithms could be used, for example, by considering machine learning approaches.

2.1.4 Output Description

At this stage of the work, we think of two possible outputs from the algorithms.

One output could be a time series of alarms that are pushed to the reasoner, for instance to alert that some users along a given path are experiencing bad QoE. The reasoner can then decide to run further measurement campaigns, such as active measurements, to better locate the problem, or query the repository to get a historical log of the path (i.e., its congestion level, uptime, and so on). Figure 1 provides an overview of how the overall algorithm would work in this case.

Another possible output is to export to the reasoner the kind of application that the algorithm of the previous section has identified to be running on top of the thin-client connection under observation. The reasoner will then take care of correlating such information with the network conditions along the path (for instance by querying the repository) and run the threshold-based algorithm to infer users' QoE. This could represent an advantage in case we manage to have an unique threshold-based scheme for inferring users' QoE rather than having it distributed. However, at the current stage this remains an open question that we plan to address.

2.2 Estimating content and service popularity for network optimization

2.2.1 Introduction

This section presents the algorithms necessary for the prediction of content popularity, as well as their inputs and expected outputs. These algorithms will be used mainly in the context of use case of Section 3.2 as presented in Deliverable D1.1[2]. This use case relies on content popularity information to optimize the network performance. For instance, by detecting early which content items will receive attention in the future, it is possible to proactively place these items in caches closer to the end users. This results in bandwidth savings as well as in an enhanced user experience.

Moreover, beyond this use case, the popularity information computed by such algorithms is useful for several other purposes such as: media advertising, trend forecasting, movie revenue estimation and traffic management. It could be even a key enabler for novel value-added services such as media curation. In general, it is interesting for operators to understand and predict the collective behavior of users in their networks. As such, we will deal in this section with the broad topic of content popularity estimation and prediction based on network measurements. The algorithms introduced hereafter are therefore applicable for use case 3.2 of D1.1 as well as for future mPlane use cases that may utilize the popularity information.

At this early point of the project, the details of the algorithms are still under deep investigation. However, here we plan to provide (i) which input data are going to feed them (Section 2.2.2), (ii) which output data they will produce (Section 2.2.4) and (iii) a broad overview of the algorithms we plan to use (Section 2.2.3). In particular, we are interested in the granularity of the data, the need of the algorithms for accessing a historical archive and the speed at which data will be consumed and produced.

2.2.2 Input Description

A lot of input information can be used to predict popularity. These can range from the simple counts of number of views to more ``sophisticated" application specific data (comments related to the content within blogs, or popularity indexes of social networks etc). However, our aim is to keep our methodology as general as possible in order to not over-specialize prediction algorithms for a particular application context. Moreover, information such as comments in blogs would be difficult to retrieve by passive measurement running at the network or transport level using a probe that performs Deep Packet Inspection.

The algorithm takes in input a time series describing the arrival process of requests (e.g. HTTP GET) targeting a certain content (i.e. a URL). In particular, each request will be described by a tuple reporting the id (a hash) of the requested content, the timestamp of the request, the user-id/IP (properly anonimyzed) corresponding to the user who generated the request, and the location in the network where the request was raised, i.e., the location of the probe. Such input data may easily correspond to the output logs of a traffic classifier (a probe such as Tstat or BlockMon) performing measurements at transport and application level on a network link or it can be provided by logs of proxy servers managed by operators. The output throughput is of course going to be proportional to the speed of the observed network links, that can span from a few Mb/s to tens of Gb/s.

Such input data, can be augmented, if needed, with additional application-specific information like explained above. If this information is proved useful, we might want to obtain it by designing new active probes that will actively request content items to extract it.

Therefore, as a summary, the input data will have the following format: <ContentID, Time, UserID, Location, meta data>; where meta data refers to application-specific information (e.g. number of comments, shares on social networks etc).

2.2.3 Algorithms Description

Predicting the popularity of content is in general a challenging task, since it is difficult to incorporate into models the effects of external phenomena (e.g. media, natural, geo-political events). Besides, cascades of information are difficult to forecast. This difficulty is further exacerbated when dealing with short time scales (e.g. hours, days). Indeed, in use cases like the one we defined in D1.1, it is important to estimate the growth rate of content as early as possible: the earlier the network reacts by adapting content placement, the more bandwidth it saves. Thus, for this use-case, it is important to rely on almost real-time measurements, so that the prediction process can immediately produce precious insights to proactively distribute the content on the network.

The methodology adopted to predict the popularity of content within a network can be splitted is described hereafter. First, we need to categorize the behavior of content over time to reveal distinct patterns of popularity growth. Indeed, it is expected that relatively a few classes of behavior will be identified. E.g. videos reporting news show very similar popularity trends, being capable of attracting a fairly large attention for a few days, and becoming completely unpopular as the reported event becomes old. We instead expect to see that music videos show a constant over time popularity growth.

Even if algorithms must be flexible enough to process data arriving at diverse speed, different kinds of content may exhibit very different dynamics, and then require very different processing performance: e.g, network data carrying YouTube traffic exhibits much larger volumes with respect to other UGC VoD (User Generated Content Video-on-Demand) systems, so the corresponding request arrival rate is expected to be large too. Algorithms may indeed need customization towards specific kinds of services to improve their processing speed.

Based on the input data described in the previous section, i.e. the tuple <ContentID, Time, UserID, Location, meta data>, other pre-processing algorithms will run a procedure to:

- retrieve the history for content associated to ContentID;
- update the number of hits and its time series.

These steps likely correspond to fairly fast operations, i.e. a query and an insertion, for a relational database relying on this data structure.

Clustering and predictive algorithms will be employed to forecast the popularity growth associated to each content: we plan to estimate the popularity growth over space and time, to get insight not only on which content is of interest but also if there are specific areas that are attracted by it.

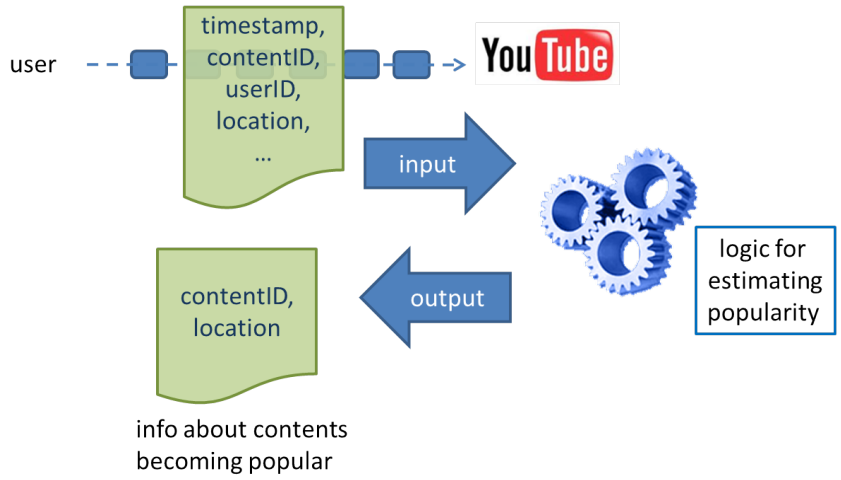

Figure 2: Overview of the algorithm for estimating content and service popularity. Input features and output data are shown.

2.2.4 Output Description

The result of the prediction process may be a simple time series of messages that are *pushed* to the reasoner to indicate which contents are going to become popular in the near future at specific place of the network. Similarly to the input data, the output can be seen as a series of couples <ContentID, Location>. Such information is indeed enough for the Supervisor/Reasoner to proactively push contents to the caches close to associated location. More advanced forecasting techniques may also predict how long that content is going to maintain its popularity, so that an eviction time may be specified and exploited to proactively evict unpopular contents from the cache.

The objective of the prediction algorithm may be as well (or in addition) to keep an up-to-date ranking of the content items that will become popular in the next period (minutes, hous, day, etc). Such a ranking can be *pulled* by the reasoner whenever it is needed.

In addition to all the prediction capabilities, the algorithms and database, designed in the context of WP3, should be able to provide (upon request) some simple historical statistics about popularities of content items, such as (non-exhaustive list): (i) what were the most popular items in a specific period (last day, week, month, etc); (ii) what is the number of hits for a certain content during a specified period.

Figure 2 provides an initial overview of the input and output features of the algorithm for estimating content and service populairty that we envision.

2.3 Active measurements for multimedia content delivery

2.3.1 Introduction

In this scenario, mPlane is used to pinpoint or narrow down systems or network segments responsible for degradations in popular multimedia streaming services (e.g. YouTube), as described in Section 3.3 of Deliverable D1.1[2]. Initially, there are a number of active probes within the network constantly or periodically downloading certain pre-configured streams, and monitoring the download process to observe the delivered quality of the streams. There are other, also mPlane-managed probes on the Internet, that are capable of active monitoring of these multimedia streaming services, but are not configured to do so (e.g. to conserve network resources, or to preform other measurements). There are also other, passive probes on the network observing and measuring traffic at network junctions (e.g. ISP peering points, enterprise network uplinks etc.). The position (geographical, topological) and other connectivity-related attributes of all the probes are recorded by mPlane in a distributed fashion.

When one of the active probes ($probe_A$) identifies a degradation of the streaming service it monitors, an alert (trigger) is raised towards the mPlane Supervisor. Then, the Supervisor needs to be able to schedule additional measurements for the same stream and/or service in question using the other ``spare'' probes available on the network. Additionally, it needs to check if there are flow or traffic measurements from passive probes on the network that are relevant in evaluating the active probes' results, or can be correlated to them (e.g. traffic/availability measurements on a link or junction taking part of the monitored stream's delivery). The Supervisor also needs to find out if there are other probes outside of its administrative domain measuring the same or related services, and obtain their measurement results via mPlane interfaces.

In order to accomplish the described tasks, the Reasoner execute a two-step process. First it issues a query to the repository asking for a list of ``nearby'' candidate probes needs to be obtained, ordered by certain criteria (most often some metric of *distance* from the probe(s) signaling the problem). Second, it checks that these probes are indeed capable of performing the additional stream measurements that are needed to narrow down the problem region.

2.3.2 Input Description

In this scenario, mPlane data layers are expected to store and work with two kinds of data:

Measurement results represented as time series

Topology data representing the location of the probes from multiple aspects (e.g. geographical and network topology).

2.3.2.1 Measurement results as time series data

Multimedia stream delivery is characterized by the following metrics. These metrics are computed by the probe every m milliseconds, and the probe locally provides n-second-resolution `Min/Max/Avg` aggregations if requested. In practice, m is 1000 and n is 60 thus raw measurements are taken each second and the aggregates are computed for each minute. m and n are of course configurable.

These aggregations are computed over 60 1-sec raw measurement values. *Counter*-type variables (e.g `bytesReceivedCnt`) are not aggregated as they monotonically increase. The parameters exported by the probes are listed in the following table.

Metric	Unit	Description
`nominalBitrate`	bps	content's adverstised momentary bitrate
`actualBitrate`	bps	actual bitrate
`bytesReceivedCnt`	bytes	bytes received (incrementing counter)
`qualityIndex`	percent	momentary ratio of of selected stream alternative's bandwidth vs. the best one (if available)
`bandwidthUtilization`	percent	actual vs. nominal bandwidth
`dataInterArrivalJitter`	msec	Jitter of OS-level chunks received by the Player
`firstDataDelay`	msec	delay of 1st data item of the transfer (last valid value)
`bufferLevel`	percent	(momentary) Player buffer level
`httpCode`	enum	last HTTP Status Code

Some of the probes are also capable of measuring the underlying TCP transport characteristics, also represented as a n-milliseconds resolution time series, and aggregated locally the same way as above. All TCP metrics are collected both ways $src \rightarrow dst$ and $dst \rightarrow src$). The additional metrics collected by these probes are listed in the table below.

Metric	Unit	Description
`windowSize{Init,Min,Max}`	bytes	Initial,Min,Max advertised TCP window size
`bytesTransmittedCnt`	bytes	bytes transmitted
`bytesReTransmittedCnt`	bytes	bytes retransmitted
`msecIdle`	msec	maximum idle time between two consecutive packets of the stream within the 1-sec sampling period
`msecRTT{Min,Max}`	msec	TCP RTT
`ooPacketsCnt`	pkts	count of number of Out-of-Order pkts

2.3.2.2 Topology and static network characterization data

Topologies - as ususal - should be represented by graphs. The exact representation (data model) within mPlane is not yet finalized. Generally, the following topology information needs to be represented, with the indication of registered mPlane probes' location within:

- Physical and logical network topology and network information (e.g. nominal link bandwidth)
- Routing and proxying/tunneling (to the level that is relevant to the HTTP(s) delivery of the multimedia services being examined)
- Autonomous Systems interconnection or other information to map administrative/provider boundaries

2.3.3 Algorithms Description

The mPlane storage layer is expected to help locating active and passive probes near $probe_A$ and the other probes iteratively involved in the troubleshooting process. Thus, WP3 algorithms need to find answers queries related the the location and connectivity of each probe. A non-comprehensive list of queries is listed hereafter:

- probes served by the same ISP or access network as $probe_A$ and their relation (in the topology)
- show the Network/Autonomous System/Provider→CDN→Consumer path for the network stream (to the extent possible from the topology data stored)
- probes (active and passive) connected to networks ``upstream'' from $probe_A$

2.3.4 Output Description

The results are lists of mPlane probes, ordered by their distance from $probe_A$ experiencing service degradation. While working to find the root cause of the degradation, the Reasoner will try to determine the set of network endpoints impacted. Thus, similar queries are going to be issues iteratively, and the results are processed by the Reasoner.

Then, the Reasoner will contact and negotiate capabilities with probes selected based on the above queries, and program them to execute similar active measurements as the root probe did. As the Reasoner is working iteratively, the above queries may be repeated for additional probes based on the findings of the re-programmed probes.

Additionally, if the Reasoner can find passive probes on the network path between the stream source and the active probe(s) receiving it, it may query and re-configure them to execute traffic measurements for the streams transmitted towards the active probes. This way, link congestion issues and transmit errors (drops) can be identified and their causes pinpointed. The list of such passive probes is also provided by the mPlane data layer.

2.4 Quality of Experience for web browsing

2.4.1 Introduction

The algorithms presented in this section deal with mechanisms to identify and pinpoint the cause of performance degradation in web browsing scenarios, as described in Section 3.5 of the Deliverable D1.1[2]. This scenario uses the input of a web browser plugin, installed in end-users browsers, able to log performance indexes related to the loading and browsing of web pages. The collected data is transmitted to the repository, where it can processed, analyzed and fed to the Reasoner.

2.4.2 Input Description

When a user loads a Web page, a large number of operations happens behind the scenes. A typical web page contains up to hundreds of different elements: images, CSS, javascript code, etc. Therefore, in order to fully render a page, all those elements need to be downloaded, possibly parsed, executed or decoded.

For each of those elements, we need to keep track of many different metrics, such as the IP addresses and port numbers of the client and server, the duration delay of the DNS requests, the TCP handshake duration, the TCP connection duration, the duration of the download, the HTTP return code, the client load, etc. Moreover, each query should be binded to the user-generated web request that originated it, in order to be able to process all the elements related to the same request together.

Even if in the initial deployment the number of probes can be small, the system should be designed to scale, and to be able to store and process the input from a possible very large number of probes (in the order of millions).

The parameters logged by the probes can be divided into five main categories: request identifiers, which are the parameters used to identify each request and associate it to the originating user request; network information, which contains lower level information, such as TCP and DNS status; HTTP information, which contains HTTP parameters and specific metrics; information about status of the client, such as system load and wireless signal strength; general performance metrics, such as the time required to complete the requests.

The preliminary set of collected metrics is listed in the tables immediately below.

Request identifiers

Metric	Description
`URI`	Unique Resource Identifier of current object (e.g. HTTP://www.google.com/news/sports/go.img)
`Session URI`	page URI of the whole browsing session that current object belonging to
`Host`	host/server name for current object

Network information

Metric	Description
Local IP address	client IP for current connection
Local port	client port for current connection.
Remote IP address	server IP for current connection
Remote port	server port for current connection (normally 80)
DNS start timestamp	DNS event start timestamp
DNS end timestamp	DNS event end timestamp
DNS duration delay	DNS duration delay (e.g. end time - start time)
SYN timestamp	TCP handshake event start timestamp (e.g. SYN packet is sent)
SYNACK timestamp	TCP handshake event end timestamp (e.g. SYN/ACK packet is received)
TCP connecting duration	TCP connecting duration (TCP end time - TCP start time)

HTTP information

Metric	Description
Server connection header	HTTP Connection header.. useful to check if persistent connection is disabled by the server.
HTTP method	HTTP query method, such as GET or POST
HTTP response code	HTTP response code from the server for current request (e.g. normally 200)
HTTP request size	HTTP request size
HTTP header size	HTTP response header size
HTTP body size	HTTP response data body size
HTTP cache size	if object is from cache, this is the object size
HTTP referrer header	HTTP Referrer header
HTTP id	random number inserted at HTTP header
HTTP Location header	HTTP Location header
Content type	content type of current object (e.g. png, flv, html....)
Content length	content length declared by the HTTP header
Accept Encoding	AcceptEncode HTTP header, set by the browser
Content Encoding	ContentEncode HTTP header, e.g. zip, gzip, etc.
Content load timestamp	timestamp of browser event of end of parsing current page
Server HTTP version	server HTTP version (normally 1.1 or 1.0)
HTTP request timestamp	timestamp of sending HTTP request
first byte timestamp	timestamp of receiving the first bytes from the server side
Application RTT	time duration between HTTP.sent and receiving 1st byte
HTTP end timestamp	ending timestamp of current HTTP transfer
Download duration	time duration of the whole data download

Client status

Metric	Description
`Browser version`	current browser version
`Client OS`	client operating system info
`Client ID`	a random number unique for each user
`Current wifi quality`	current wifi quality (e.g. signal strength in percentage)
`Cpu idle`	current idle CPU resource
`Cpu percent browser`	CPU percentage required by the browser
`Free memory`	total free memory of client PC
`Memory used`	total memory in use of client PC
`Browser memory percentage`	percentage of memory used by the browser
`Ping gateway`	ping delay to the default gateway
`Ping nameserver`	ping delay to the configured DNS server
`Ping google`	ping delay to www.google.com host
`Object aborted`	whether current object is stopped by the user before finishing downloading all the data

Performance metrics

Metric	Description
`Request timestamp`	starting time of the request event in the browser
`Session start`	starting time of the browsing session
`Load timestamp`	timestamp of page fully load event
`Full load time`	Page load time
`Content load time`	Page parsing time
`Abort time`	Page abandoned time
`Server operating sytem`	Server declared operating system at HTTP header
`Cache status`	browser cache status of current object

2.4.3 Algorithms Description

There are different algorithms that should be applied to the collected data. The algorithms to apply to the data can be classified in two main categories: ``data preprocessing'' and ``data analysis''.

2.4.3.1 Data preprocessing

Data pre-processing is a fundamental step in the data mining process. Data can be inconsistent, corrupted or redundant, resulting in out-of-range values, impossible data combinations, missing values, etc. Without a preliminary pre-processing phase, the analysis of such data can an produce misleading results. Data pre-processing include different sub-phases such as data cleaning, data normalization, format conversion, etc.

Data cleaning is the process of detecting and correcting corrupt or inaccurate records, caused, for example, by user entry errors or corruption in transmission or storage. *Data normalization* is the process of reducing data to its canonical form, to minimize redundancy and dependency. *Data transformation* converts a set of data values from the data format of the source into the data format of the destination system.

2.4.3.2 Data analysis

The analysis algorithms aim to allow both a manual and an automatic troubleshooting. Simple statistic functions are applied to the collected data (mean value, standard deviation, median, etc.). Such information can be used to do exploratory analysis, understand outages, measure SLAs (availability, latency, etc.), help ISP to do capacity planning.

Collected measurements can also be clustered, e.g. using for example K-meams algorithm. K-means clustering is a method of cluster analysis which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. This results in a partitioning of the data space into Voronoi cells. Such technique allows to discover and explain the performance differences among users and identify the root causes for poor performances. Clustering techniques can be used to cluster users, flows, servers, ISP or geographical locations. For example, one possibility is to exploit the measurement of several clients, e.g. of all the web clients of the different devices in the same home, in order to improve the potential of identifying more precisely the cause of the performance impairment. For instance if the WiFi at home is overloaded and some of the end systems access the Internet via WiFi while others are connected to the home gateway via Ethernet, the use of multiple devices should allow to distinguish between congestion of the WiFi link as compared to congestion of the access link of the ISP. Clustering can be exploited to identify problems in the access link peculiar to specific ISP. It is important to understand that clustering algorithm requires as input euclidean distances, so the algorithm implementation requires the proper definition and implementation of distance metrics. In k-means, the number of clusters k is an input parameter: an inappropriate choice of k may yield poor results. As pointed in [5, 4], this choice is crucial for the performances and the result of the analysis, and should be chosen appropriately by a domain expert. Other more advanced clustering algorithms could be considered as well. Note that collected data have a natural temporal ordering. Statistics and measurements should be calculated with different time granularities (e.g. hour, day, week, month).

2.4.4 Output Description

The produced output is stored in the database itself, and, if required, additional structures are generated in order to improve the retrieval efficiency. Results are analyzed by a manual operator for exploratory analysis, understanding the cause of possible outages, etc., or retrieved and used by the reasoner in order to trigger alarms and require further analysis. The clustering algorithm produce a list of centroids of the observations, and the association between observations and centroids.

2.5 Mobile network performance issue cause analysis

2.5.1 Introduction

This section presents the algorithms necessary for the analysis of the performance of mobile devices, and the identification of the degradation causes, as described in Section 3.6 of the Deliverable D1.1[2].

2.5.2 Input Description

A combination of probes across various parts of the network will provide the required input in order to identify the core causes of poor user experience. The `Application probe` provides on-demand information as perceived from the application point of view. The `Mobile OS probe` offers on-demand information considering the device capabilities and the device status (CPU, Memory). Furthermore, an important aspect of this probe is to measure the cellular network conditions (associated cell tower, signal strength, bit errors, transmit characteristics, power state of the device, etc). The `Mobile ISP probe` captures, both passively and actively aggregated information such as number of associated devices, overall traffic, channel utilization, scheduling/QoS policies, etc. Moreover, information about the ISP's backbone such as capacity, transfer rates and latency can be measured using acceleration proxies and middle boxes installed in the network. Finally, the `Core Network and Service Provider probes` measure the performance of the core network and the CDN/Service provider.

The data from these probes can be returned as timestamped JavaScript Object Notation (JSON) strings and each measurement will be timestamped. Please note that for some installations it might not be possible to extract some of the described information due to proprietary drivers/hardware or luck of available APIs. For instance, some vendors of RNCs/SGSMs/GGSNs use proprietary protocols across their hardware.

We now present details about the data that are generated by each probe.

2.5.2.1 Application probe

The input from probes provides information as perceived from the application point of view.

- **Player Status** (`playerStatus`): The ability of the player to display the required video. Any errors such as inadequate CPU or memory, missing codecs, lack of caching mechanisms, poorly configured drivers, lack of hardware acceleration are logged using the `logEvent` sub-attribute.
- **Video Information** (`videoInformation`): The sub-attributes contain information about the current video `URL`, available `bitrate`(s), selected `resolution`, `videoCodec`, `audioCodec` and `duration`.
- **Request Status** (`requestStatus`): The reply(s) from the video server until the video is retrieved. The information includes information about `redirections`, and videos that were `not-found` or `denied`. Furthermore, the `IP address` of the server that returned each status code and a `timestamp` is logged. Note that in some cases this information might not be available.

- **Average Video Bitrate** (`avgVideoBitrate`): The reported average video bitrate is reported as videos can be encoded using `adaptive bitrate`. Furthermore, some video players can switch between different bitrate streams either dynamically or after instructed by the user. It can be measured by dividing the size of the played video by its elapsed duration.
- **Current Video Bitrate** (`currentVideoBitrate`): This is the average bit-rate over a 20-second sliding window. If a user-initiated bitrate change was requested the window is reset. This input is only requested on demand by the application probe when an active measurement is required.
- **Setup Time** (`setupTime`): This is the interval between the time when the player initiates a connection to start downloading the video till the time that the player starts playing the video (i.e., the initial connection and buffering interval).
- **Buffering Ratio** (`bufferingRatio`): This is the percentage of the time that the video spent in buffering and it does `not` include the initial buffering during the setup period. It is defined as: $bufferingRatio = \frac{BufferingTime}{BufferingTime+PlayingTime}$.
- **Buffering Frequency** (`bufferingFrequency`): This represents the average number of buffering events per minute of video.
- **Average Buffer Fill** (`bufferFill`): The average number of video seconds that the buffer contains during the streaming.
- **Average Transfer Rate** (`transferRate`): The average transfer rate during the video download. If the video is split into multiple chunks, then this metric only include the time when a chunk was downloaded. Note that in some configuration (e.g., HTML5 players, this information might not be possible).
- **Dropped Frames** (`droppedFrames`): The number of frames that had to be skipped due to insufficient bandwidth (e.g., for live streaming) or performance (e.g., CPU power). Note that in some players this information might not be extractable.
- **Unique Identifier** (`PID`): An identifier that is used to identify the problem across different probes.

2.5.2.2 Mobile OS probe

This probe offers passive and on-demand information considering the device capabilities and the cellular network conditions. An open-source OS may be used (e.g., FireFoxOS or Android). Notice that the amount of information that can be collected is bounded by the data that can be extracted from the cellular modem driver (i.e., Samsung Galaxy S2 devices support the extraction of such information).

- **Device Information** (`deviceInformation`): Is a string that contains the available information for this device. Sub-attributes of this field include: the `type` of the device (i.e., Samsung Galaxy S3), the `OSversion`, the `screenResolution` and the `availableMemory`.
- **Association Information** (`associationInformation`): Encodes the information about the current (instantaneous) conditions of the selected network interface: (e.g., `associatedTower`, `powerState`, `modulationType`, `RSSI`, `connectionStatus`).

- **Signal Information** (`signalInfo_10sec, signalInfo_60sec, signalInfo_300sec,`): these three attributes provide the signal information for a sliding window of 10, 60 and 300 seconds. Depending on the modem driver, the sub-attributes include:

  RSSI (`RSSI_avg, RSSI_min, RSSI_max`): the average, minimum and maximum RSSI with the associated base-station.

  Modulation Type (`modulation_avg, modulation_min, modulation_max`): an index that defines the type of the modulation used (e.g, EDGE, UMTS, LTE).

  Number of disconnections (`numDisconnect`): the number of times that the medium had been disconnected during the selected window.

  Time ratio of disconnections (`timeDisconnect`): the percentage of time during the window that the terminal was disconnected.

  Number of handovers (`numHandovers`): the number of times that the terminal has switched base-stations.

- **Transfer Rate Information** (`transferRateInfo_10sec, transferRateInfo_60sec, transferRateInfo_300sec`): these three attributes provide transfer rate statistics for the interface:

  Download rate (`downloadRate_avg, downloadRate_min, downloadRate_max`): the average, minim and maximum download rate.

  Upload rate (`uploadRate_avg, uploadRate_min, uploadRate_max`): the average, minim and maximum upload rate.

  Packet delay variation (PDV) (`pdv`): The delay is specified from the start of the packet being transmitted at the source to the end of the packet being received at the destination. Note that this information might not be available in some flows, operating systems or network drivers.

- **Unique Identifier**(`PID`): An intensifier that is used to identify the problem across different probes.

2.5.2.3 Mobile ISP probe

The mobile ISP probe, captures, both passively and actively, information about each mobile terminal and base-station. Notice that due to the fact that most ISPs use a variety of hardware configurations that support different statistics about their operation, the information described below might be only partially available.

Passively generated for each terminal:

- **Association Information** (`associationInformation`): Encodes the base-station that the terminal is associated and de-associated. If available, sub-attributes contain the terminal's `powerState`, the `modulationType` and the `IPinformation`.

At each RNC, passively generated every t seconds:

- **Load Information** (`loadInformation`). Information about the load of each sector over of sliding window of the last t seconds:

  Number of associated clients (`numClients`).

Number of active clients (`numActiveClients`): the number of clients that are exchanging traffic at high-power mode.

Channel utilisation (`channelUtilisation`): the percentage that the medium is busy transmitting/receiving.

At each backbone controller: Since a variety of links might be used (e.g., satellite, microwave, fiber), information about the load of the links is returned.

- **Link Load Information** (`linkLoadInformation`) for each interface:

  Download load (`downloadLoad`): the aggregate download load.

  Upload load (`downloadLoad`): the aggregate upload load.

- **Dropped packets** (`droppedPackets`): packets that had to be dropped due to the buffers being full or due to CPU overload (if available).

2.5.2.4 Core Network and Service Provider probes

These probes provide information such as core network performance, peering utilization, service uptime, number of served clients, network congestion, CDN selection at the service provider. Furthermore, the CDN provider can also measure the caching hit-rates, service latencies and report any issues at the distribution network.

2.5.3 Algorithms Description

In our scenario we adopt an application-initiated probing model. More specifically, as shown in Figure 3, following the detection of a problem, each probe is incrementally consulted depending on the collected data. We will now describe in detail the resulting actions.

Application Probe: The main task of the application probe is to identify a potential quality of experience problem and initiate the probing process. More specifically, based on the input described in the previous section, two main categories of issues can be identified: i) unable to play due to application miss-configuration and ii) quality of service issues. If the problem does is not related to application miss-configuration, the *phone probe* is consulted to further investigate the root of the problem.

Phone Probe: After receiving a request from the application probe, the phone probe examines the OS/phone state to detect any local issues. More specifically the phone probe will identify problems related to poor resources and signal/bandwidth issues. If both the application and the phone probes cannot identify the cause of the problem their data are forwarded to the mobile ISP probe for further investigation.

Mobile ISP probe: After receiving a request from a device, this probe is responsible to identify any issues within the ISPs network. Note that it is not necessary to trigger a new on-demand measurement for each request: recent measurements concerning the same user, users associated with the same cellular location, or about the same service or CDN might cached to enhance performance. When an issue is found, a warning is generated within the ISP in order to troubleshoot problematic configuration of the network. Furthermore, automatic actions can be potentially made to load-balance the load. Finally, the application can be notified about the cause of the problem depending on the operator's policy.

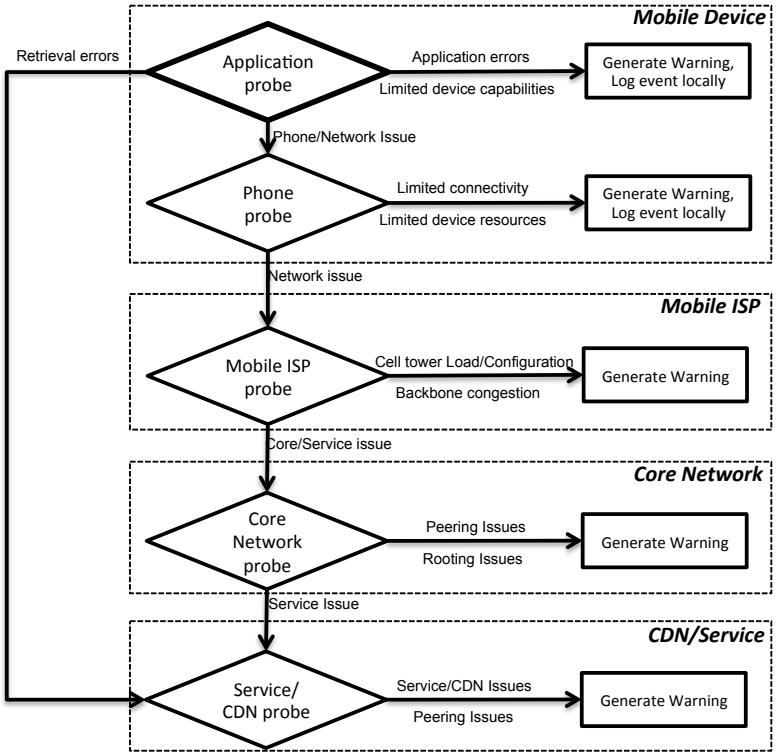

Figure 3: Workflow of identifying the causes of mobile video performance issues.

Core Network an CDN probes: If the problem is not found between the user's mobile device and the ISPs peering point, the service provider and the content distribution network have to be probed. Information concerning the nature of the problem (e.g., URL/IP of the service, the type of the problem and the PID) is forwarded and further actions are taken.

2.5.4 Output Description

The output of each stage of the investigation is a warning that includes the <PID, Location, UID, ResourceID, ErrorCode, DetectedCause, ProbeData>. Where required, the data are anonymised and the association is stored locally at the repository of each party. The output is either forwarded to the next probe (for further investigation) or to the reasoner in order to get a historical overview of the issue-cause associations and perform passive analysis. Furthermore, the error code can be also returned to the application in order to display a warning to the user.

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

2.6.1 Introduction

This section presents the basic network data analysis techniques necessary for the detection and Root Cause Analysis (RCA) of traffic anomalies in large-scale networks, corresponding to the use case presented in Section 3.7 of the deliverable D1.1[2]. The description is performed in terms of the basic algorithms that are employed in the use case, the required input data coming from the measurements layer and from other repositories, and the outputs of the analysis, which will then be available for further processing.

The main driver of the anomaly detection and RCA use case is to track and understand anomalous changes in traffic patterns, specially in the light of the complex current Internet traffic scenario, dominated by the popularity of content providers that leverage Content Delivery Networks (CDNs) to serve end-users. Indeed, a major portion of today's Internet traffic is served by CDNs. Anomaly detection and RCA is a fundamental task for ISPs, network administrators as well as for CDN providers, to understand how Internet services use the network (e.g., at specific time of the day there is a traffic shift due to load balancing), to characterize the behavior of the users (e.g., due to low performance users stop to use a service), and to optimize or troubleshoot their systems when the detected changes disrupt the normal operation QoS of their network.

In the general scenario of this use case, mPlane is continuously collecting predefined measurements over time and from multiple vantage points. Both active and passive measurements can be considered. Measurements are collected at the measurements layer and stored in the repository where, at predefined time scales, are processed to detect and diagnose anomalous traffic behaviors (i.e., identify the root causes for such unexpected behaviors).

Some examples of the changes mPlane detects and diagnoses include changes in RTT to specific content, changes in the average download speed from certain servers, service unavailability, changes in the traffic volume distribution, changes in the number of traffic flows, etc. Once an anomaly is detected, alarms are raised and could eventually automatically trigger the intelligent reasoner analysis and the corresponding iterative measurements. In the final and complete approach, when an anomaly is detected, the corresponding alarms are logged and presented to the mPlane user, and the most plausible causes for that anomaly are displayed.

It is is important to recall that the kind of analysis techniques that we depict in the following sections are *simple* and are meant to be performed at the repositories and analysis layer of the mPlane architecture. The term simple refers to the fact that the corresponding operations have to run in potentially huge amounts of data coming from the measurements layer, and as a consequence, they cannot represent (in principle) complex analytics.

2.6.2 Input Description

Given the heterogeneity of services to monitor and types of devices to serve as probes, there is a huge set of input metrics that could be considered as valid examples for the anomaly detection process. However, in order to provide a concrete description of the problems and analysis techniques being addressed, we shall focus on three specific scenarios for anomaly detection and RCA: (i) **CDN**

traffic and CDN network analysis, (ii) **tracking end-user traffic-share evolution**, and (iii) **detecting unknown behaviors in end-user traffic**. In the following description, we do not specify the temporal granularity of input data. Such granularity depends on the specific scenario requirements. As a general rule, the temporal granularity to use depends on the temporal behavior of the events to be detected and characterized.

- **CDN traffic and CDN network analysis**: multiple traffic and network features can be considered as input for analyzing CDNs and detecting anomalies. The following is a non-exhaustive list of the metrics considered as input for the basic data analysis techniques employed in this scenario:
  - Network distance from a vantage point to a group of CDN servers: for example, RTT, hop count, and AS count. Analyzing distance-based data of CDN servers permits to track network architectural changes (e.g., deployment of a new data-center, failure of an already established data-center). Also, it allows for load balancing analysis and detection of traffic shifts at different CDNs, e.g., by identifying time variant load balancing policies under different load conditions (including flashcrowd events).
  - Per flow download throughput and RTT from different CDN servers: throughput and delay are related to performance, and are useful to identify traffic shifts and congestion events. For example, a drop in the download throughput or a increase in the RTT at peak hours can be due to policies redirecting the clients requests to ``far'' data centers, or to congestion events in the network.
  - IP ranges of CDN servers, and organizations owning those ranges: knowing the specific IPs which correspond to specific CDNs allows for per-CDN analysis, inter-CDN comparison, and cross-CDN detection of anomalies.
- **Tracking traffic share evolutions**: tracking the share of services used by large groups of end-users and detecting novel trends and anomalous behaviors is useful for many purposes, from improving network provisioning techniques and operations to supporting marketing decisions. For example, we can derive trends on the evolution of Internet services such as YouTube and Facebook, considering their share of volume or their popularity across the users. Strong variations in the share of traffic which is not classified can also reveal interesting behaviors, for example, sharp changes in the amount of generic HTTP traffic can hide the presence of a new application that is becoming popular. Depending on the capabilities of the probes, some of the input metrics include:
  - If the network probes have a traffic classification module, then the input data metrics include (all on a per-application basis): up/down bytes, number of flows, flow timestamps (start and end of the flow), and number of users.
  - If the probes do not have traffic classification capabilities, input data additionally includes HTTP requests with the headers and DNS resolution queries. We specifically focus on HTTP-based traffic as it represents the top source of end-user traffic.
- **Detecting unknown behaviors in end-user traffic**: detecting previously unseen events and patterns in user traffic is achieved through the following input data:
  - Performance metrics: per flow throughput, inter-packet arrival times, per flow TCP retransmissions, RTT.

- Protocol-based metrics: number of TCP flag packets (SYN, FIN, RST, etc.), number of initiated/concluded TCP sessions.
- Connection-based metrics: number of different IPs source/destination, number of different source/destination ports.

2.6.3 Algorithms Description

The kind of algorithms which analyze the previous input data can be separated into those running at the repositories and analysis layer of mPlane, and those running on top of the results provided by the analysis layer. As we said before, we usually expect that the algorithms running at the repositories and analysis layer are less complex in terms of number of operations w.r.t. those running on top of their results, as they should be able to process large amounts of data. The following lines describe the analysis required by each of the aforementioned scenarios:

- **Algorithms for CDN traffic and network analysis:**
  - **Search and filter CDN flows:** the first step to analyze CDN traffic is to isolate those flows coming from specific CDNs. An easy way to perform this filtering is by origin IP address: many public databases exist today (e.g., MaxMind, http://www.maxmind.com) where the organization owning an IP address can be retrieved. Search and filtering can be achieved in the large scale by efficient indexing; we may think on a database-like repository where all the flows captured at different vantage points are imported every ΔT minutes and indexed by timestamp and IP address.
  - **CDF of average throughput and RTT per CDN**: a characterization of CDN traffic performance can be achieved by tracking the average throughput and the RTT of the flows served by the corresponding CDN. The anomaly detection algorithm we propose works by detecting differences in a time series of the empirical distributions on different indicators such as throughput or delay. The CDF computation involves sorting, summing, and thresholding basic operations.
  - **CDF of minimum RTT per CDN**: the minimum RTT of the flows coming from specific CDNs is directly correlated to the distance from the vantage point to the servers/data centers. As before, changes in the CDF reveal changes in the distribution of the CDN servers selected for the specific traffic, evidencing for example load balancing policies at the CDN.

  The anomaly detection algorithm works slightly as follows: every ΔT minutes, the flows served by different CDNs are isolated. The CDF of the average per flow throughput and RTT are computed, and similar for the CDF of the minimum per flow RTT. The algorithm assumes the existence of a base CDF reflecting the past normal behavior of the system in terms of the corresponding indicators. If the distance between the current CDF and the based CDF is above certain detection threshold, the anomaly detection algorithm flags an anomaly; on the contrary, if the distance is below the threshold, the base CDF is updated by integrating the current CDF as part of the normal behavior.

- **Tracking traffic share evolutions:**

- **Pattern matching on HTTP headers:** as we said before, the traffic shares analysis is based on HTTP traffic flows, which currently carry the majority of the end-user traffic. Every new HTTP flow is parsed and the contacted host name is compared against a set of defined regular expressions or patterns describing different services and applications. If a matching pattern is found, the flow is assigned to the corresponding service.
- **Total HTTP traffic volume**: every ΔT minutes, the total volume of HTTP flows is computed and stored back in the repository, indexed by time stamp. In this way, a time series of HTTP traffic volume is directly available for further analysis.
- **HTTP traffic volume variance computation**: every ΔT minutes, the variance of the total HTTP traffic volume is computed from the past 10 contiguous HTTP volume measurements. This indicator is used to compute the anomaly detection threshold w.r.t. abrupt changes in the overall HTTP traffic volume, which may evidence the presence of a new HTTP application which is becoming popular, or even the event of a flashcrowd.
- **Per service volume counting**: every ΔT minutes, the downloaded volume per service is computed and kept back in the repository, indexed by time stamp. Tracking the volume per service allows for detection of trends and usage behavior of HTTP services.
- **Per service flow counting**: every ΔT minutes, the number of flows per service is computed and kept back in the repository, indexed by time stamp. Similar to the previous case, tracking the number of flows per service allows for analysis of traffic share behaviors.
- **Per service number of users counting**: every ΔT minutes, the number of unique users per service is computed and stored back in the repository, index by time stamp.
- **Inter-service distance computation**: a fast approach for characterizing and comparing the traffic shares of a predefined set of services is by doing clustering analysis. For each service of the predefined set, and for each time stamp (ΔT granularity), we define a vector characterizing the service at the given time. Such a vector includes the up-link and down-link transmitted volume, the number of packets and flows, the number of users, the average flow size and flow duration, the average and median minimum RTT measured on top of the flows, among others. The analysis step computes the inter-vector distances (i.e., Euclidean distance) and stores them back in the repository. These distances are then used for clustering purposes.

The HTTP traffic shares tracking and analysis algorithm works slightly as follows: every ΔT minutes, the HTTP headers of the flows are matched against a set of predefined regular expressions describing a set of services (e.g., YouTube, Facebook, Netflix, etc.). The aforementioned set of metrics is computed. For each metric, a different detection threshold is computed using the variance of the previous non-anomalous measurements. In addition, the service vectors are built and the distance among them are computed and stored back in the repository. These distances are finally used by a clustering algorithm, which runs outside the repositories and analysis layer.

- **Detecting unknown behaviors in end-user traffic:**
  - **Counting of number of flows, bytes, specific packet types (TCP SYN, FIN, RST, TCP retransmissions, HTTP cancellations), different source/destination IPs and ports:** every ΔT minutes, all these metrics are computed for all the traffic flows coming from a single or fixed set of vantage points. The computations done at the repositories and

analysis layer are based on filtering and counting, and we assume that all the specific metrics are already computed at the measurements layer.

- **Computation of average per-flow throughput and average RTT:** every ΔT minutes, the average per flow download throughput and the average per flow RTT are additionally computed, using simple filtering and counting.
- **Inter-time-slot distance computation:** for each ΔT time slot, we define a vector characterizing the behavior of the traffic at the given time. Such a vector includes all the previously computed metrics. The analysis step computes the inter-vector distances (i.e., Euclidean distance) and stores them back in the repository. These distances are then used for clustering purposes.

The algorithm for detecting unknown behaviors in end-user traffic works as follows: the first step is the learning phase, in which every time slot ΔT is mapped to an n-dimensional vector, during a period of 24 h. The inter-vectors distance among these learning time-slots are computed and stored back into the repository, which will be then used by a clustering algorithm identifying groupings and structure. In the analysis phase, every new time slot is mapped to an n-dimensional vector, and its distance towards the centroids of the clusters identified at learning time is computed. If this distance is below a certain detection threshold, then the new time slot is added to the learning time slots; on the contrary, if the distance is above certain detection threshold, an anomaly is flagged. The specific dimensions in which the new time slots deviates more from the learning time slots are identified, which are then used for understanding the causes of the deviation.

2.6.4 Output Description

Given that the basic analysis algorithms are very atomic, their output is straightforward. For example, in the case of CDN traffic and network analysis, the main output of the algorithms are selective per CDN flows and time-series for various CDFs (i.e., sorted vectors) for different analysis features on top of these flows. In the case of tracking traffic share evolutions, the main output of the analysis algorithms are time series on HTTP traffic shares and per service characterization features (e.g., number of users, of flows, of bytes). In addition, time series of inter service-based-vectors distances are provided as output, which are then used for clustering purposes. Finally, in the case of detecting unknown behaviors in end-user traffic, the main output are the time series of inter time-slot distances, describing temporal traffic behavior and performance.

Note that in the three presented scenarios, the **outputs generated by the basic analysis algorithms** are meant to be **post-processed** by specific structure analysis algorithms (i.e., **clustering**), change-detection algorithms (i.e., **anomaly detection**), and **outliers-detection** algorithms.

2.7 Anomaly detection and root cause analysis in large-scale networks: data mining algorithms

2.7.1 Introduction

This Section addresses the application of data mining techniques to the mPlane network traffic traces, for the detection and Root Cause Analysis (RCA) of traffic anomalies in large-scale networks, corresponding to the use case presented in Section 3.7 of the deliverable D1.1[2], with a particular focus on distributed and scalable cloud-based association rule extraction.

Data mining focuses on studying effective and efficient algorithms to transform huge amounts of data into useful and actionable knowledge. Industries are attracted by the business opportunities arising from the exploitation of the extracted knowledge, while researchers are interested in the challenging issues coming from the application of data mining techniques to new scenarios and growing datasets, leading into the field of Big Data Analytics.

Different data analytics tools rely on data mining algorithms to gain interesting insights from large volumes of semi-structured or unstructured data. Data mining techniques allow extracting previously unknown interesting patterns such as groups of data objects (cluster analysis), unusual patterns (anomaly detection), correlations and dependencies (association rule mining) [14].

When dealing with huge data collections, the computational cost of the data mining process (and in some cases the feasibility of the process itself) can potentially become a critical bottleneck. Parallel and distributed approaches can be adopted to increase the mining efficiency and improve algorithm scalability.

Association rule mining is a two-step process: (i) Frequent itemset extraction and (ii) association rule generation from frequent itemsets [3]. Since the first phase represents the most computationally intensive knowledge extraction task, effective solutions have been widely investigated to parallelize the itemset mining process both on multi-core processors [17, 16, 11, 8] and with a distributed architecture [15, 6, 9, 18]. However, when a large set of frequent itemsets is extracted, the generation of association rules from this set becomes a critical task.

In the context of the mPlane project, we plan to design, develop and apply a horizontally-scalable approach, which consists of a series of distributed jobs run in the cloud. Each job receives as input the result of one or more preceding jobs and performs one of the steps required for the data mining task. As a reference, we expect each job to be performed by one or more MapReduce tasks run on a Hadoop-like cluster.

The algorithm architecture consists of the following steps:

- Pre-processing
- Item frequency
- Rule mining
- Rule ranking

2.7.2 Task definition and expected output

in this Section, a formal definition of the data mining task is presented and the outputs of the algorithm as a result of its application to mPlane data are formally defined.

Let $\mathcal{D}$ be a dataset whose a generic record r is a set of features. Each feature, also called *item*, is a couple *(attribute, value)*. Since we are interested in analyzing statistical features computed on traffic flows, each feature models a measurement describing the network flow (e.g., Round-Trip-Time (RTT), number of hops).

An *itemset* is a set of features. The *support count* of an itemsets I is the number of records containing I. The *support* $s(I)$ of an itemset I is the percentage of records containing I. An itemset is *frequent* when its support is greater than, or equal to, a minimum support threshold $MinSup$. *Association rules* identify collections of itemsets (i.e., set of features) that are statistically related (i.e., frequent) in the underlying dataset. Association rules are usually represented in the form $X \rightarrow Y$, where X (also called rule antecedent) and Y (also called rule consequent) are disjoint itemsets (i.e., disjoint conjunctions of features). Rule quality is usually measured by rule support and confidence. *Rule support* is the percentage of records containing both X and Y. It represents the prior probability of $X \cup Y$ (i.e., its observed frequency) in the dataset. *Rule confidence* is the conditional probability of finding Y given X. It describes the strength of the implication and is given by $\mathrm{c}(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$ [14].

Given a dataset $\mathcal{D}$, a support threshold $MinSup$, and a confidence threshold $MinConf$, the mining process discovers all association rules with support and confidence greater than, or equal to, $MinSup$ and $MinConf$, respectively.

To rank the most interesting rules, quality indexes can be exploit, such as the rule support, confidence, and lift measures [14], In particular, lift is a good candidate since it measures the (symmetric) correlation between antecedent and consequent of the extracted rules. The lift of an association rule $X \rightarrow Y$ is defined as [14]

$$\mathrm{lift(X, Y)} = \frac{\mathrm{c}(X \rightarrow Y)}{\mathrm{s(Y)}} = \frac{\mathrm{s}(X \rightarrow Y)}{\mathrm{s(X)s(Y)}} \tag{2.1}$$

where $\mathrm{s}(X \rightarrow Y)$ and $\mathrm{c}(X \rightarrow Y)$ are respectively the rule support and confidence, and $\mathrm{s}(X)$ and $\mathrm{s}(Y)$ are the supports of the rule antecedent and consequent. If lift(X,Y)=1, itemsets *X* and *Y* are not correlated, i.e., they are statistically independent. Lift values below 1 show a negative correlation between itemsets X and Y, while values above 1 indicate a positive correlation. The interest of rules having a lift value close to 1 may be marginal. We expect the mined rules to be ranked according to their lift value to focus on the subset of the most (positively or negatively) correlated rules.

2.7.3 Input data

Input data for the proposed approach are traffic traces collected by passive probes, e.g. Tstat [7, 13].

Tstat is part of mPlane network data collection tools. The probe rebuilds each TCP connection by matching incoming and outgoing segments. Thus, a flow-level analysis can be performed [13].

Our approach focuses on a subset of measurements describing the traffic flow among the many provided by passive probes. Any other probe or network data collection tool that can provide the

same data can be used as input source for the proposed approach. The most meaningful features we plan to include in the analysis are detailed in the following:

- the *Round-Trip-Time* (RTT) observed on a TCP flow, i.e., the minimum time lag between the observation of a TCP segment and the observation of the corresponding ACK. RTT is strongly related to the distance between the two nodes
- the *number of hops* (Hop) from the remote node to the vantage point observed on packets belonging to the TCP flow, as computed by reconstructing the IP Time-To-Live
- the *flow reordering probability* ($P\{reord\}$), which can be useful to distinguish different paths
- the *flow duplicate probability* ($P\{dup\}$), that can highlight a destination served by multiple paths[1]
- the total *number of packets* ($NumPkt$), the total *number of data packets* ($DataPkt$), and the total *number of bytes* ($DataBytes$) sent from both the client and the server, separately (the client is the host starting the TCP flow)
- the minimum ($WinMin$), maximum ($WinMax$), and scale ($WinScale$) values of the *TCP congestion window* for both the client and the server, separately
- the *TCP port* of the server ($Port$)
- the *class of service* ($Class$), as defined by Tstat, e.g., HTTP, video, VoIP, SMTP, etc.

Based on measurements listed above, an input data record is defined by the following features: RTT, Hop, $P\{reord\}$, $P\{dup\}$, $NumPkt$, $DataPkt$, $DataBytes$, $WinMax$, $WinMin$, $WinScale$, $Port$, $Class$.

2.7.4 Pre-processing

This step performs the following two activities: (i) filtering, (i) value discretization, and (ii) format conversion.

Filtering is used to discard irrelevant flows with respect to the analysis task. For instance, if the number of packets in the flow is below a given threshold (e.g., 10 packets), then flow statistics may be considered unreliable and hence the flow could be excluded from the analysis.

Associaton rule mining requires a transactional dataset of categorical values, hence the discretization step converts continuously valued measurements into categorical bins, and the format conversion step addresses the requirement of a transactional dataset in the form of attribute = value.

Automatic discretization approaches can exploit state-of-the-art techniques to select appropriate bins depending on data distribution. Otherwise, domain-expert knwoledge can be exploited to select meaningful static values for the bins.

This task entails an inherently parallel elaboration, considering that can be applied independently to each record.

[1] $P\{reord\}$ and $P\{dup\}$ are computed by observing the TCP sequence and acknowledgement numbers carried by segments of a given flow. We refer the reader to [13] for more details.

2.7.5 Item frequency

This step computes the item frequency from the transactions emitted by the pre-processing phase. Its output is a $(key, value)$ pair for each item in the transaction, where the key is the item itself (e.g., RTT=5-10), and the $value$ is its count, i.e., always 1. An aggregation function is then executed to sum all the values for each key, hence computing the support count of each item. This is a typical group-by query performed as a distributed job.

2.7.6 Rule mining

This step performs (i) the itemset mining and (ii) the rule extration. It may require multiple jobs to be run.

For the itemset mining task, we plan to exploit existing state-of-the-art techniques such as the parallel FP-growth algorithm, as described in [10].

The rule extraction step, instead, is going to be a novel contribution as a distributed cloud-based algorithm, since there seems to be no publicily available approaches to be exploited in the context of the project. Rule extraction may target a subset of the whole association rules that could be extracted, based on the specific interest of the domain, e.g., rules correlating network service types or discriminating different network performance metrics.

2.7.7 Rule ranking

A final step is executed to sort and aggregate the rules according to the consequent and the quality index measure. Lift is a good candidate as a rule quality measure. Sorting and aggregating on the consequent helps in analyzing the extracted rules for finding significant correlations.

2.8 Verification and certification of service level agreements

2.8.1 Introduction

This Section discusses the algorithms required for the verifiction of Service Level Agreements (SLAs), as described in Section 3.8 of the Deliverable D1.1[2] As reported in D1.1 the verification of an SLAs is technically equivalent to the verification of the implicit guarantees of service made by a provider offering Internet service. Therefore one of the fundamental task of mPlane will be a group of procedures to check and verify SLAs between providers and customers of Internet services, defining the minimum level of service provided in terms of one or more measurable parameters. Currently SLAs are tested in terms of some network performance parameters as ``bandwidth'' (generally expressed in terms of raw throughput). However, with the evolution of the applications, SLAs will regard aspects more and more related to user perception. Therefore we first need to define new metrics that take into account Quality of Experience (QoE), and investigate on the introduction of SLAs based thereon. In this scenario, each user may have SLAs with different providers (e.g. ISP, VoIP, IPTV); we therefore need to consider correlation among different SLAs. This will permit both the verification of SLAs between providers and customers from either the provider or customer end, as well as the customer-end verification of advertised throughput for comparison and regulatory purposes. In this scenario, we look at both the verification of SLAs between providers and customers from either the provider or customer end, as well as the customer-end verification of advertised throughput for comparison and regulatory purposes. Details on the SLA procedures can be found in D1.1 chapter 3.8, and according to such a document let us to characterize mPlane data procedures in terms of SLA operation, in particular how to proceed in terms of SLA contract definition: basic schema with the QoS and/or QoE parameters, SLA negotiation, SLA monitoring, and SLA enforcement, according to defined policies. Furthermore it has to be pointed out that SLA verification is generally a continuous process. Therefore, the mPlane infrastructure of the service provider, or of an enterprise customer, is set up to monitor a given SLA at the time the SLA is committed; the supervisor then instructs the probes to periodically perform measurements, either passively or actively, and compares measured parameters to stored SLA parameters, until such time as the SLA is discontinued. Periodically generated results or reports are then either forwarded by the supervisor to the appropriate administrator or regulatory authority, or stored in a repository for later retrieval. SLA violations can also be alerted by the supervisor; the exact reporting and alerting behavior is subject to the terms of the SLA. In general the metrics required for SLA analysis are those which are used to specify the level of service required; for the scenarios envisioned:

1. simple availability of a service (i.e., a time-series of yes/no states),
2. throughput between two endpoints given application-unlimited demand,
3. throughput between two endpoints given constant-rate demand,
4. latency between two endpoints given a load profile, and/or
5. application-specific metrics for QoE measurement.

These measurements should be taken in the presence of other normal background load at both endpoints. With the service evolution, the trend is to shift the verification towards applications. This implies that metrics and measurements have to be applied at the application layer; suitable

SLA parameters must be defined for specific Web services such as YouTube, specified in terms of application-level measurable parameters (e.g. measured video stall rate). This implies measurement method that is able to exploit all the capacity of the line; the most suitable method here is an active probe. Since dedicated hardware devices for measurement are prohibitively expensive for home gateway applications, this active probe should be based on a software tool that a user can download and run on their home PC, laptop, or tablet. On the other end ISPs have the necessity to control and monitor the network traffic in each segment of their own network in order to avoid (or to take under control) SLA problems. Such a network monitoring will be obtained by means of a passive probe network, that will be used also for ISP network management and traffic engineering procedures.

The supervisor is required to control the measurements, to generate periodic reports in the format required by a regulator, to alert the administrator in case an SLA is not met. Furthermore SLA verification is an inherently cross-domain operation, as there are always at least two parties to such an agreement. However, prior work on SLA verification has focused on verification from the point of view of a single party. In the case of QoE-based SLA, however, measurements must necessarily be taken at or close to the client; in this case, an mPlane infrastructure at a customer network could take measurements of controlled video transmission (e.g., with the ISP actively downloading videos emulating user behavior), and report the results back to the provider network to inform it of SLA-relevant parameters; the provider could then compare customer-visible measurements to provider-network measurements in verifying its own compliance to an SLA. Therefore a fundamental role is for repositories for storing SLA verification information that must be able to store the metrics listed above as well acceptance criteria for SLA comparison.

2.8.2 Input Description

Each user will have at disposal a probe to verify his service and experience. Such a probe has to measure a long list of parameters also depending on the kind of physical access at disposal (ADSL, Fiber, 3G, 4G, Wi-Fi,...). For practice reasons the probe is assumed as a software agent that downloading a file (or a list of files) is able to test the network quality (active probe). So far such a procedure has been based operating at OSI Layer 4 making measurement of throughput, delay, jitter and data loss. Therefore we can say that periodically the probe simply sends four parameters, but that can be also more, up to some dimes in the case of wireless environment as we will show below. With the progress in terms of access capacities (higher and higher than 10 Mb/s) and services (HD TV, 3D, games,...) measurements at layer 4 could result unsuitable to characterize both ISP SLA and user perception. Measurements based on TCP permit to quantify the user perception and therefore are related to MOS (or Quality of Experience). Therefore according to these results we can conclude that a probe, to give a reliable contest for the user, has to make measurement both with UDP (verifying the ISP SLA) and TCP (to test the user perception) approach. If we wish to include further characteristics in terms of QoE for applications for each application running on a device other parameters related to MOS should be considered. For an instance by looking at You Tube applications MOS could be measured in terms of stalling (see Pedro) both for number and duration.

Wireline environment

Here we report the data treatment procedure experience for SLA verification carried out in the framework of the Italian monitoring network “Misura Internet” by means of the Nemesys agent tool [12]. The environment that was taken into account referred to the Italian wireline broadband access that mainly consists of ADSL2+ technology with a download bandwidth lower than 20 Mb/s.

These bandwidth delay values allows us to validate the TCP method since the difference among the nominal and measured bandwidth is small in the current scenario. To better illustrate such a consideration we report the bandwidth (as measurement ratio=measured bandwidth/nominal one) vs the bandwidth delay product (BDP) for the tests performed in Italy in the framework of AGCOM tests. As reported in [12], 96% of the tests have a BDP lower than 30000 that assures a good reliability in the measurement carried out. The Misura Internet architecture [1] is based on two different methodologies regarding SLA:

1. a monitoring network consisting of 20 test points distributed along Italy and specifically placed in major Italian cities (i.e., one test point for each region) to monitoring the ISP performance and to provide the users information regarding the quality of the ISP network.
2. End user measurements, concerning throughput and QoS, that are carried out directly at the user home, using a open source software Ne.Me.Sys (Network Measurement System) that is developed specifically for this project by Fondazione Ugo Bordoni with the aim of verify the SLA. When a consumer measures its own fixed line connection performances, a personal approach is defined. Furthermore, the consumer could compare the obtained results with benchmark and statistic values.

The server of the couple has been located into a IXP (Internet eXchange Point), also called NAPs (Neutral Access Points), and the client could be the end user personal computer or a probe located within one of the government building. For such measurements the client has to (down/up)load a pseudorandom file to evaluate throughput. The file size is a very important parameter; indeed it affects the TCP (Transmission Control Protocol) slow start mechanism. To have a reliable description of the user performance to be compared with SLA several measurements are necessary along the time. For such an aim several measurements would be necessary along the day to take into account all the possible variations due to environmental conditions and traffic distribution, but only one day could be not enough since the analysis should require also the week behavior, but also the month and the year ones. To make a measure, each pseudorandom file is downloaded (uploaded) 20 times consecutively. More specifically one single test is composed by a single file download/upload, 20 tests constitute a measure. At least one measure per hour during the day is needed. As far as latency is concerned, 10 PING test constitute a measure, also in this case at least one latency measure per hour is need. All measures are statistically elaborated, the 5Th (called minimum bandwidth) and the 95th (called maximum bandwidth) percentile, as well as the average value and the standard deviation are calculated for each test. Starting from tests, also Packet Loss Ratio and Unsuccessful Data Transmission rate are calculated. Packet Loss Ratio represents the ratio between the number of not replied PING commands and the total number of sent PING commands. Unsuccessful Data Transmission rate is the ratio between unsuccessful data transmissions and the total number of data transmission attempt in a specified time period. The method adopted by Ne.me.sys allows the user to monitor his wireline broadband access by means of its PC. Therefore it is necessary a method to avoid that other devices (PCs, tablet, smart TV) connected to the broadband access disturb the measurement.

One of the most stringent constraint concerning the alien traffic; indeed, in the first release, the user web-surfing was forbidden, but sometimes some applications automatically connected themselves to Internet, without the user permission. In these case Ne.Me.Sys. did not allow the measure. In the second release a small amount of alien traffic is permitted; in this way the end user can complete more easily the measure cycle and the measures are always valid. In particular a threshold has been introduced: the alien traffic has to be less than 10

In terms of data treatment each user transmits to the measurement server a group of n data (n=4 in this case: throughput, RTT delay, jitter, data loss) for each measurement test (every 20 minutes for a maximum of three days). It has to be pointed out that the server has to be able to simultaneously treats thousands of downloading and uploading of files for tests and manage repository of final data measurements. For future measurements to take into account higher bandwidth networks (>20 Mb/s) a novel approach is necessary to consider the differences that can be present between physical layer and application layer as shown in fig. x.1. Therefore we assume that we will need at least data for Layer 1 and Layer 4. The topic is under investigation in mPlane WP2 and currently we can say that the number of the data is mxn, where m is the number of OSI Layer considered.

Wireless environment
Wireless environment is much more complex than the wireline one since much more information is necessary to characterize the user quality due to the random behavior of the radio channel. Currently FUB is investigating about a new measurement approach of Misura Internet in wireless environment in the framework of the new AGCOM resolution DEL154/12/CONS and here we report the measurement approach that is experimentally running. To characterize a wireless broadband access together with QoS parameters typical of wireline environment, further information are necessary corresponding to the characteristics of radio channel as degrading effects due to the signal propagation and to the cell occupation. We decided to consider all the radio aspects in terms of "measurement failure" . Therefore this is the list of user measurement to characterize its QoS mobile broadband access:

1. download-upload throughput. As in wireline case with TCP approach; it is defined as the ratio between the file dimension and the time transmission duration. The transmission duration is the period that starts from the instant when the wireless network has received all the information necessary to start the download/upload transmission up to the last bit of the file test is received.
2. Transmission data failure (down/upload). Percentage among data transmission failure and the transmission data attempts. Failure can occur when the test file is either not totally transmitted or without error within a fixed time-out.
3. Packet loss.
4. Transmission delay in terms Round Trip Time. It is obtained by means of Echo Request/Reply (Ping) in agreement with protocol ICMP (RFC 792: "Internet Control Message Protocol")
5. Delay variation (jitter).

2.8.3 Algorithm description

All the log measurement have to contain all Layer 3 information that each probe (meant also as software agent downloaded on mobile device) exchange with the network. Furthermore log measurement have to contain georefential information to show the measurement location.

The measurement number strongly depends on the statistical approach. According to the FUB experience on mobile environment it was decided that to have a full wireless environment for Italy measurements have to be carried out in 20 cities, one for each Italian Region. To have reasonable statistical information each city was subdivided in several areas 250000 m^2 wide, called pixel. For each city the number of the pixels depend on the city dimension. To have an idea for 20 cities we

have a total of 16562 pixels. In order to have a more limited number of measurements among all the pixels we selected 1013 ones as representative. Assuming 25 minutes for each measurement 422 hours should be necessary to obtain measurements on all the selected pixels. Taking into account the time for traveling with a car measurement test we need at least 600 hours, and assuming 8 hours for day we need 75 days (15 weeks) for a complete characterization.

Customer Experience for these field measurements is based on the following services:

1. http downloading
2. FTP uploading
3. http browsing
4. PING packet loss, RTT as average and variance (Jitter).

According to these consideration some Key Performance Indicator (KPI) have been identified.

Cycle test description
For each test we defined the following file dimensions:

1. http downloading: 3MB (mp3 file)
2. FTP uploading: 1 MB (mp3 file)
3. http browsing: ETSI Kepler Web page (800 kB)
4. PING: from 10 to 40 packets for each test.

KPI list
In the following we list the KPI defined for wireless environment:

1. FTP uploading unsucces ratio
2. FTP uploading throughput
3. http download unsuccess ratio
4. http download throughput
5. http browsing unsucces ratio
6. http browsing session time
7. RTT
8. packet loss
9. jitter

2.8.4 Output description

Looking at the list of KPI used as representative to characterize the QoS and QoE for a mobile user we can point out some considerations:

1. each measurement has characterized by a long sequence of data considering all the KPI values but including also the information on L3 and Geo information;
2. The measurement architecture has to be able to simultaneously manage thousands of measurements;
3. The KPI introduced in this environment allows us to characterize the user from different layers point of view (from physical to app), but with the advent of higher capacities mobile environment (i.e. LTE) we should introduce novel measurement to distinguish the channel capacity from the TCP degradation as explained in Sect. X.1 for wireline environment. Furthermore to better characterize the user perception in terms video service performance KPI should be introduced concerning typical web services (i.e. You Tube) with authomatic Mean Opinion Score (MOS) methods; it means much more data from measurements;
4. This measurement architecture can allow SLA treatment at different levels. The user can have a full characterization of his device, but it has to be defined the wireless contest where to carry out the measurement (home, particular locations,...) in order to define measurement procedure with relative duration. Furthermore different SLA can be defined, for an instance with the ISP but also with some service providers;
5. From the collection of the data coming from the user probe each ISP can check the state of its own network verifying the mantainance of all its SLA. In particular two different solutions are possible for an ISP: a) to use a third party architecture that collect the measurement of all the ISPs, where each ISP can access to the data belonging to its network, b) use an own monitoring network.

The QoS measurement architectures carried out by FUB in the framework of the AGCOM activities to monitoring the internet accesses, with particular interest for SLA certification have illustrated some characteristics in terms of data treatment and mining. First of all the architectures are based on file download and therefore particular attention has to be given to server sizes that have to be able to simultaneously answer to user requests. Particular investigations have been necessary to define the size of such files and also the characteristics (HTTP, FTP, PING,...). Each measurement output can consists of several data, that can be a few as in the wireline case and very much in the wireless case (if we include also Layer 3 and Geo). Such data from the probe located in the user device have to be processed to obtain a certificate (PDF) describing the user characteristics, valid for the SLA comparison. The architecture processor (Reasoner in terms of mPlane) has to be able to simultaneously elaborate thousands of such data. This architecture can be used also by the ISP to analyze to state of its network; in particular the data coming from the user can be useful to understand the levels of network performance and in particular the conditions to satisfy the SLA with the users. This architectures are supposed as third party operation, but could be also implemented by the ISP to have a complete monitoring of the network. It is clear that such a monitoring network dedicated to an ISP has a cost that can be small if it is considered only in terms of software agent. It has to be investigated if this kind of architecture can be considered as sufficient for an ISP to characterize all its network not only in terms of SLA satisfation; in fact an ISP is also interested to

monitoring the congestion levels in the different segments of its network and in particular to answer to user demand both in terms of bandwidth and service performance (for an instance in terms of Class of Services also from the business point of view). For the ISP point of view it is necessary to analyze other architectures based on passive probes located in traffic aggregation points (Base station, GGSN, router,...) and in particular to verify the relationship among measurements carried out in the traffic aggregation points and the ones carried out by users on their IP devices.

3 Conclusions

This document describes and maps basic algorithms to perform analytic tasks in WP3, which corresponds to different use cases addressed in mPlane, defined in WP1.

The WP3 objective is to work on systems to store and pre-process data collected by the mPlane probes, developed within WP2 context. Essentially, we define such algorithms as black boxes, focusing on input and output data.

Each scenario has different requirements, in terms of format, volume and velocity of the available data to process; in terms of computational power needed to analyze the data (CPU or IO bound algorithms); in terms of analysis frequency (hourly, daily, weekly or occasional) and deadline requirements for the output (delay tolerant, real time); in terms of output to produce (synchronous, asynchronous, etc.). In the listed algorithms, two main classes of algorithms and requirements can be identified: real time (or nearly real-time) algorithms and batch algorithms. The former aims to produce output with strict constraints on the system response, and process data as it comes in, typically without buffering delays; incoming data is typically processed in memory, without using intermediate storage. In contrast, batch algorithms typically involve very large amounts of data, very difficult or impossible to be analyzed in real time; data is previously stored and, at a later stage, analyzed. While many scenarios have a strong connotation on one or the other aspect of the analysis, some of them requires both real time and batch analysis on the available data.

While many of the considered algorithms are common and their sequential implementations are well known, their parallel counterparts need to handle a large amount of data and they are not trivial. Indeed, while certain tasks lend themselves to parallelization because they require that a large number of independent computations (think for example to low level image processing or to Bitcon mining), many algorithms cannot be easily split up into parallel portions; in addition, sometimes naive and simple approaches tend to be extremely inefficient. From the description of the algorithms, it emerges that there are many commons algorithms across all scenarios: clustering algorithms, first order statistics, rank statistics are recurrently appearing, making these algorithms good candidates for belonging to a common set that can be used by most of the use cases.

References

[1] Misurainternet.it. http://www.misurainternet.it.

[2] Use Case Elaboration and Requirements Specification. mPlane Deliverable D1.1, Jan. 2013.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94*, pages 487--499, 1994.

[4] H. Cui and E. Biersack. Distributed troubleshooting of web sessions using clustering. In *TMA 2012, 4th International Workshop on Traffic Monitoring and Analysis, March 12-14, 2012, Vienna, Austria / Also published in LNCS, 2012, Volume 7189/2012*, Vienna, AUSTRIA, 03 2012.

[5] H. Cui and E. W. Biersack. Trouble shooting interactive web sessions in a home environment. In *SIGCOMM HomeNets 2011, ACM SIGCOMM Workshop on Home Networks, August, 15-19, 2011, Toronto, Canada*, Toronto, CANADA, 08 2011.

[6] M. El-Hajj and O. R. Zaïane. Parallel bifold: Large-scale parallel pattern mining with constraints. *Distributed and Parallel Databases*, 20(3):225--243, 2006.

[7] A. Finamore, M. Mellia, M. Meo, M. Munafò, and D. Rossi. Experiences of internet traffic monitoring with tstat. *IEEE Network*, 25(3):8--14, 2011.

[8] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y.-K. Chen, and P. Dubey. Cache-conscious frequent pattern mining on modern and emerging processors. *The VLDB Journal*, 16(1):77--96, 2007.

[9] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: parallel fp-growth for query recommendation. In *RecSys*, pages 107--114, 2008.

[10] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 107--114, New York, NY, USA, 2008. ACM.

[11] L. Liu, E. Li, Y. Zhang, and Z. Tang. Optimization of frequent itemset mining on multiple-core processor. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 1275--1285, 2007.

[12] E. M. L.Rea. Italian qos monitoring network: Impact on sla control. In *Proceedings of the 2012 International Telecommunications Network Strategy and Planning Symposium*, NETWORKS '12, pages 1--6. IEEE, 2012.

[13] M. Mellia, M. Meo, L. Muscariello, and D. Rossi. Passive analysis of tcp anomalies. *Computer Networks*, 52(14):2663--2676, 2008.

[14] Pang-Ning T. and Steinbach M. and Kumar V. *Introduction to Data Mining*. Addison-Wesley, 2006.

[15] I. Pramudiono and M. Kitsuregawa. Tree structure based parallel frequent pattern mining on pc cluster. In *DEXA*, pages 537--547, 2003.

[16] O. R. Zaïane, M. El-Hajj, and P. Lu. Fast parallel association rule mining without candidacy generation. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 665--668, 2001.

[17] M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency*, 7(4):14--25, Oct. 1999.

[18] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Huang, and S. Feng. Balanced parallel fp-growth with mapreduce. In *2010 IEEE Youth Conference on Information Computing and Telecommunications (YC-ICT)*, pages 243 -- 246, 2010.