mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Design of the Reasoner

Author(s):	Author names
FTW	P. Casas (editor), A. D'Alconzo
NEC	M. Dusi, S. Nikitaki, M. Ahmed
POLITO	S. Traverso, M. Mellia, D. Apiletti, L. Grimaudo, E. Baralis
ENST	D. Rossi, D. Joumblatt
TI	A. Capello, M. D'Ambrosio, F. Invernizzi, M. Ullio
FW	A. Fregosi, E. Kowallik, S. Raffaglio, A. Sannino
EURECOM	M. Milanesio
FUB	E. Tego, F. Matera
NETvisor	T. Szemethy, B. Szabó, L. Németh
ALBLF	Z. Ben Houidi
TID	G. Dimopoulos, I. Leontiadis, Y. Grunenburger, L. Baltrunas
FHA	M. Faath, R. Winter
A-LBELL	D. Papadimitriou

Document Number:	D4.2
Revision:	1.0
Revision Date:	30 Jun 2014
Deliverable Type:	RTD
Due Date of Delivery:	30 Jun 2014
Actual Date of Delivery:	25 Jul 2014
Nature of the Deliverable:	(R)eport
Dissemination Level:	Public

Abstract:

This deliverable describes the design and specification of the Reasoner system with a limited set of analysis/diagnosis rules as knowledge structure, and also evaluates the possible extensions to be included into the knowledge structure regarding learning of new rules. The deliverable presents a per-use case definition and instantiation of the Reasoner, including a first set of domain-knowledge-based analysis rules as well as the associated workflow of the iterative analysis. Some first evaluation results of the iterative process are reported. Finally, different learning techniques for extending and/or generating the knowledge structure of the Reasoner are overviewed.

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

1 Introduction

The main target of mPlane is to provide visibility on top of the complex system represented by today's Internet-like networks. The **measurement layer** provides a distributed and ubiquitous network monitoring framework to gather heterogeneous measurements from an assorted number of different vantage points. As such, the measurement layer provides the ``eyes'' of the mPlane. The **repository and large-scale data analysis layer** provides the capabilities for storing and processing the large amount of measurements coming from the measurement layer, i.e., it represents the ``muscles'' of the mPlane. The **analysis modules** provided by the supervision layer allow the mPlane to extract more elaborated and useful information from the gathered and pre-processed measurements. The multiple analysis modules provide as such different analysis capabilities to the mPlane, therefore representing the ``arms'' of the mPlane.

In this deliverable we present the final component of the mPlane, namely the **Reasoner**, which represents the ``intelligence'' of the mPlane. The Reasoner allows structured, iterative, and automated analysis of the measurements and intermediate analysis results. It orchestrates the measurements and the analysis performed by the probes, the large-scale analysis repositories and the analysis algorithms, actuating through the Supervisor to interconnect with the other mPlane components.

Starting by the **design and specification of the different components of the Reasoner in section 2**, this deliverable presents in **section 3 a per use-case description of how the Reasoner performs in the practice**, including a per use-case definition of iterative analysis and diagnosis rules composing its Knowledge Structure. The associated workflow of the iterative analysis in each use-case is also presented, as well as some first evaluation results of the iterative process.

Finally, **different learning approaches for extending and/or generating new analysis rules within the Knowledge Structure are presented in section 4**, and some considerations for building reasoning diagnosis graphs are discussed.

2 Design and Specification of the Reasoner

In this section we provide a description of the Reasoner, which represents the intelligence of the mPlane. This description includes the design and specification of the different components of the reasoning system. Fig. 1 recalls the architecture of the mPlane. The Reasoner coordinates the measurements and the analysis performed by probes and repositories, actuating through the Supervisor. It is responsible for the orchestration of the iterative analysis and the correlation of the results exposed by the analysis modules. Such a reasoning-based system is capable of generating conclusions and triggering further measurements to provide more accurate and detailed insights regarding the supported traffic monitoring and analysis applications. As such, the reasoner offers the necessary adaptability and smartness of the mPlane to find the proper high-level yet accurate explanations to the problems under analysis in the different use cases.

The Reasoner has different specific roles, depending on the use case to tackle. In the case of troubleshooting support-based use cases, the main role of the Reasoner is to drill down the measurements and interpret the analysis results provided by the analysis modules to find the most probable root causes of the associated problems. In the case of generic measurements analysis, the main role of the Reasoner is to automate the iterative measurements analysis process. In both cases, the main requirement of the Reasoner is to be able to iteratively perform different analysis tasks, taking additional analysis steps based on the results of the previous observed results. As such, the Reasoner's core structure is highly similar to that of an automatic Root Cause Analysis (RCA) system [47]. RCA is typically used as a reactive approach for identifying failure event(s) causes. Through RCA we can investigate new problems, uncover unexpected impacts, and quantify the scale and trend of different factors/events contributing to performance issues.

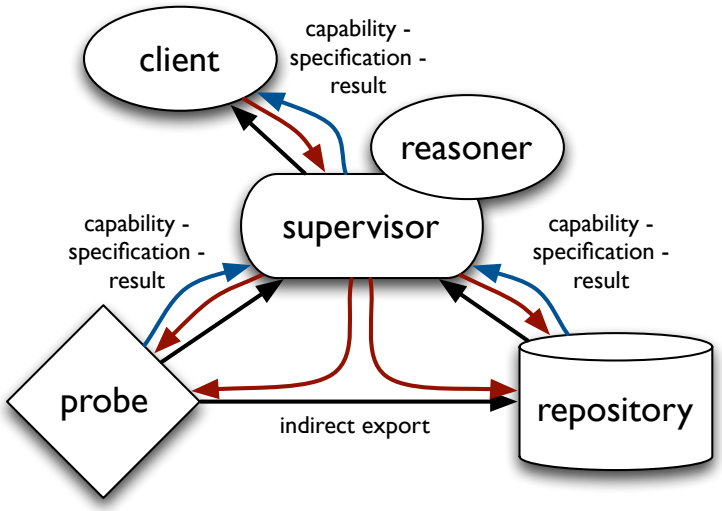


Figure 1: The mPlane architecture. The Reasoner coordinates the measurements and the analysis performed by probes and repositories, actuating through the Supervisor.

Fig. 2 depicts an overview on the complete Reasoner system. The Reasoner is divided in three major components or blocks: (1) the iterative analysis engine and diagnosis graph, which guides the automatic analysis of problems to find their root causes; (2) a knowledge structure, composed of set of domain-knowledge based analysis rules that helps to structure the diagnosis process; and

(3) the learning, exploration and automatic extraction rules engine, which allows to discover new analysis rules and to explore those events for which no root cause were automatically identified.

The size, granularity, and geo-distributed span of the measurements performed and analyzed by the mPlane at each of its layers defines a pyramidal design, in which the larger the visibility on the overall problem obtained from the measurements, the more aggregated and summarized these measurements should be so as to make the analysis feasible. For example, whereas packet level measurements are potentially performed locally at the probe side, repositories generally analyze more aggregated measurements (e.g., pre-filtered flows) from geo-distributed vantage points. Following this philosophy, the Reasoner does not work directly on top of raw data, but on the results obtained by the different analysis modules, which we shall define from now on as *events*.

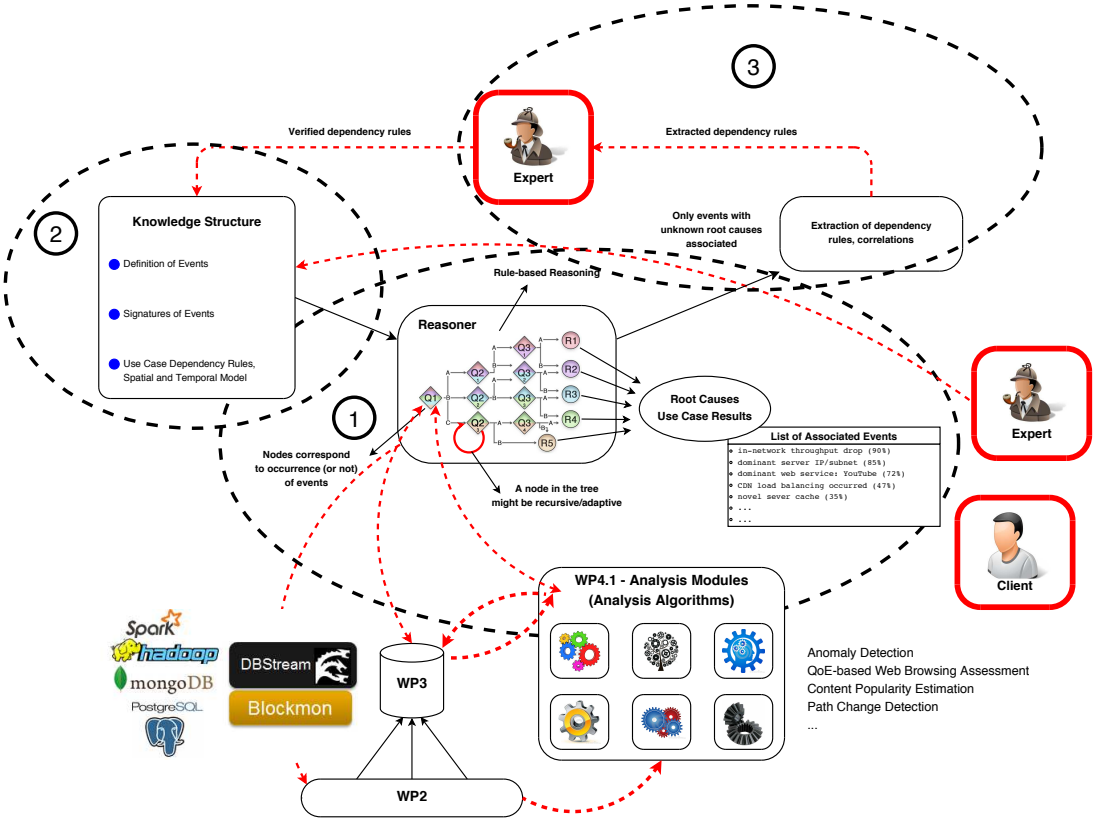


Figure 2: Automatic anomaly diagnosis and events exploration through the Reasoner.

An event captures a particular type of network condition (e.g., link congestion, YouTube throughput drop, overloaded cell, Google CDN load-balancing, anomaly detected, inter-AS routing modifications, etc.). Events are extracted from the measurements performed and analyzed at the probes and repositories through the different analysis modules, either in a continuous fashion (e.g., continuous analysis, such as anomaly detection) or in an on-demand fashion (e.g., specific reactive query, such as server reachability measurements). In general lines, events can be classified as either *symptom events* or *diagnosis events*: symptom events are the type of service problems to be analyzed (e.g., YouTube QoE degradation), whereas diagnosis events refer to the evidence of a potential root cause taking place (e.g., Google CDN load-balancing).

The iterative analysis engine of the Reasoner consists in verifying the occurrence (or not) of different events, through the analysis of different diagnosis rules, which relate problems with events and

root causes. The diagnosis of a specific issue is performed by following the analysis steps dictated by a diagnosis graph. A diagnosis graph combines a set of analysis rules in a graph-like structure, allowing for fast, structured, and automated anomalies/failures diagnosis and analysis. Events and analysis/diagnosis rules are defined in the knowledge structure of the Reasoner. This knowledge structure defines a set of basic domain-knowledge-based rules that allow the Reasoner to take decisions based on the particular monitoring application it is orchestrating, which can eventually be expanded by learning from past experiences.

The final component of the Reasoner corresponds to the exploration and automatic extraction rules engine which aims at extending the domain knowledge. Domain knowledge and operational experience can be unreliable or incomplete. Therefore, the specification of an initial diagnosis graph can be rather under-performing, both in accuracy and completeness. In this direction, the set of analysis rules is not necessarily static and can eventually be expanded by learning from past experiences, either automatically of by a manual exploration process.

The role of the exploration and automatic extraction process is to correlate all the events that occur at about the same time and which are spatially related to the failure/anomaly under investigation, in order to derive new diagnosis rules. These new rules can be derived either manually, or as we see next, automatically through machine learning algorithms. Final expert intervention is generally required to validate the identified dependencies, adding them into the knowledge structure. The target is to automate as much a possible this process, so as to minimize the final expert intervention in this validation process.

In the following sections, we provide a more detailed description of the main components of the Reasoner.

2.1 Events and Diagnosis Graph

The process of anomalies and failures diagnosis requires the exploration and correlation of relevant events. The definition of relevant events generally requires domain expert knowledge. As we explained before, network-related events capture a particular type of network condition. For example, let us consider the overloading of a specific monitored network link. In this example, a simple link throughput-tracking algorithm would monitor the used capacity of the link, and flag a relevant link overloading event when the link load attains a pre-define utilization threshold, for example, a 90% of the link capacity. An accurate definition of these events require improves the diagnosis capabilities of the Reasoner. Therefore, an event has to be described factually, including its qualitative and quantitative attributes, the type, the magnitudes, the location, and the associated time span. In the mPlane terminology, events are defined as m-tuples consisting of the following fields:

- `event name`, e.g., link overload.
- `location type`, e.g., Gn downlink interface.
- `time span`, e.g., 2013-10-21-12:30:00, 2013-10-21-12:35:00.
- `retrieval process`, e.g., Simple Link Congestion Detection Algorithm -- SLCDA(utilization threshold C_{th}).
- `additional features`, e.g., number of flows, number of bytes, list of server IPs originating the flows, etc.

The retrieval process points to the actual algorithms needed to generate/detect the occurrence of the corresponding event. Events are generated by different algorithms applied to a set of monitored KPIs (e.g., abrupt change detection, heavy hitter identification, statistical traffic/KPIs analysis, etc.). An event instance additionally specifies the initial time of the event, as well as the ending time, always related to the specific retrieval process. For example, if we consider the link overloading case, the initial time would correspond to the first time slot when the pre-defined capacity threshold is exceeded, whereas the ending time would correspond to the first time slot when the utilization drops below the threshold. Based on these definitions, an event instance would potentially look as follows:

```
<link_congestion, Gn_1, 2013-10-21-12:30:00, 2013-10-21-12:35:00,
SLCDA(90%), {1500 flows, 235.5 MB, srvIP_list,...}>
```

These events can be classified in two different classes, depending on their nature: symptom events and diagnosis events. A symptom event characterizes the problem that has to be analyzed through the mPlane. For example, if we consider the Anomaly Detection use case, a symptom event could be a major drop in the YouTube flows throughput impacting the Quality of Experience (QoE) of a large number of end users. The event could be instantiated as follows:

```
<YouTube_QoE_anomaly, ISP_A-PoP_1, 2014-02-15-20:30:00, 2014-02-15-20:35:00,
SAD(x_1, x_2, ..., x_k),
{2.3 K-flows, 6.1 K-users, srvIP_list, download_th dist, avg_QoE,...}>
```

In this case, an event reflecting an anomaly in the YouTube QoE of more than 6,000 users is generated by the Statistical Anomaly Detection module SAD, configured with the input parameters x_1, x_2, ..., x_k. The SAD analysis modules works on top of the traffic captured at one of the Points of Presence (PoP) of a certain ISP A. The additional descriptive features logged on the event instance provide more details to understand the causes for such anomaly, and can be used to create a specific signature for this specific event, out of a set of several instances. Examples of relevant diagnosis events could include the selection of a different group of servers provisioning the YouTube flows, the overload of one of the segments in the end-to-end paths from YouTube servers till the monitoring vantage point, routing changes resulting in network paths with worse performance, and so on. The definition of all these events is done within the knowledge structure, considering the specific use case to tackle. Still, many events are not specifically tied to a specific use case, and can therefore be integrated in the analysis of different use cases.

Once events are defined and the corresponding measurements and analysis modules are instantiated to track their occurrence, the key question is how does different events relate to each other, so as to understand their dependencies and verify their causal relations. These dependencies between symptom events and diagnosis events are specified through dependency or diagnosis rules. Diagnosis rules are initially defined on the basis of expert domain knowledge, but can be further evolved by analyzing the resulting measurements through learning techniques. A very simplified example of decision rule could be the following:

```
if { YouTube_QoE_anomaly :
     QoE anomaly detected in PoP_1, ISP_A
                  &
     Google_load_balancing :
     New Google cache selected in AS 15169
                  &
     ~Path_perf :
     No performance issues in inter-AS end-to-end paths
                  &
     Server_anomaly :
     Server issues detected in new servers between 20hs-23hs } → YouTube servers overloaded at peak time.
```

To limit the span of the example, this rule does not include all the corresponding events' parameters, but in the general case, all the relevant information is included for the analysis, specially regarding the temporal context, i.e., the occurrence or not of these events is verified in a similar temporal scope.

The quality of the decision rules determines the performance of the overall diagnosis analysis. Indeed, the key to proper automatic fault diagnosis is to clearly understand the dependency relationships between a symptom and diagnosis events. However, building such rules is generally very challenging and requires multiple types of information, for example:

- topological information (e.g., physical link connecting two different routers, location of base stations, locations of content caches, etc.).
- cross-layer dependency (e.g., L4 vs L7 quality/performance, L1 devices supporting L3 links, etc.),
- routing and load balancing (e.g., BGP and OSPF routing info, DNS mappings, etc.).

Identifying relevant events along a timeline of events leading to the specific problem to diagnose improves the definition of the diagnosis rules. In this direction, the most challenging part is that of relating events in a cause-effect basis. The notion of diagnosis graph allows to structure and model the dependencies among events, by exploring their temporal and spatial relationships. A diagnosis graph combines a set of analysis rules in a graph-like structure, which explores the temporal and spatial relationships between symptom and diagnosis events to find the most probable root causes. Events can be associated to the same spatial location, and also to the same temporal span, i.e., events happening ``at the same time'' of the analyzed problem. Different temporal scopes can be defined for analyzing these temporal relationshios, in orther to account for delayed causal effects.

Fig. 3 depicts an example of diagnosis graph within the mPlane framework. The nodes of the decision graph correspond to the occurrence (or not) of specific diagnosis events. The leaves of the graph represent the root causes. The links between nodes show the causal dependencies between events: for example, events are defined as causal such that if eliminated from the graph, they should cut the sequence chain of interconnected events. A node within the decision graph might also be adaptive and recursive, as it might require multiple analysis iterations to come up with an answer related to the queried event. Indeed, the reader should recall that behind each of these diagnosis graph nodes potentially lies a complex set of continuous and/or on-demand analysis modules.

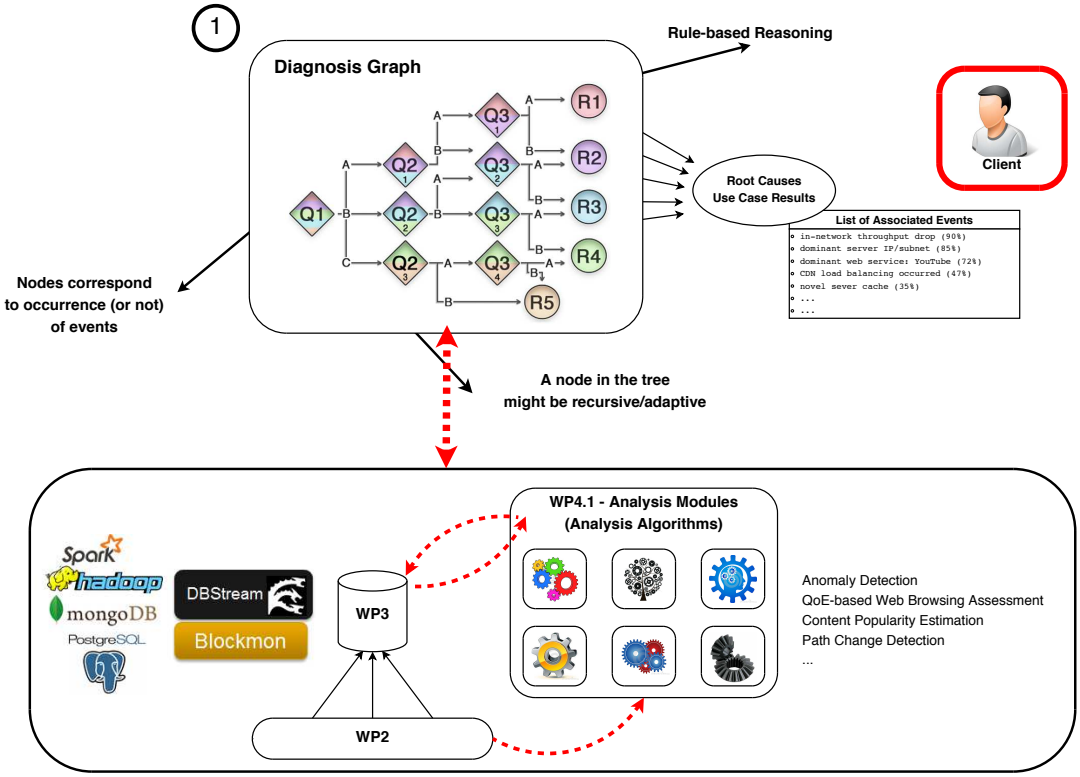


Figure 3: The diagnosis graph. A diagnosis graph combines a set of analysis rules in a graph-like structure, which explores the temporal and spatial relationships between symptom and diagnosis events to find the most probable root causes.

When it comes to the specific diagnosis process and the reasoning behind, the Reasoner implements Rule-based Reasoning (e.g., decision-tree like graph). Rules-based reasoning represents a simple and direct association between the diagnosed root cause and the evidence(s) for better interpretation; it is very effective in the practice, and different weights can be assigned to symptom-diagnosis links by expert knowledge so as to improve the performance of the diagnosis. Using such diagnosis graphs in a per-failure/anomaly class, the Reasoner looks for the presence of diagnostic events, and identifies the root cause as the leaf with the highest probability of occurrence.

2.2 Knowledge Structure

The Knowledge Structure of the Reasoner contains a set of manually predefined relevant events as well as a set of domain-knowledge based diagnosis rules that permits to structure the diagnosis graphs. This list of events and rules available at the Knowledge Structure of the Reasoner permits to re-use the definition of a set of events and the corresponding rules to tackle multiple different use cases, without the need of defining all the required elements from scratch every time. Indeed, some events and diagnosis rules are generic to any large-scale anomaly detection and diagnosis process in Internet-like networks, for example, when verifying the occurrence of performance degradation events in inter-AS paths connecting the remote servers to the users.

As we explained in previous section, the definition of a new event requires the specification of the retrieval process which generates such events, and these might be parametric as in the proposed

link congestion example. Having this idea in mind, one specific event definition can be reused in different diagnosis use cases which might require different parametrization of the events, for example, to improve the sensitivity of the analysis and/or to consider the normal behavior of the monitored KPI.

The knowledge structure will provide a very basic yet useful event-language specification to allow an expert user to define new parametric events and diagnosis rules, but also to construct new rules based on existing ones. As such, building new diagnosis applications would become faster and simpler. The specification of this event-language definition will follow the mPlane Reference Implementation registry and metadata definitions for interpretation of metrics and measurements.

2.3 Learning, Exploration and Automatic Rule Extraction

One of the main challenges in the automatic diagnosis of problems based on expert knowledge rules is that domain knowledge and operational experience can be unreliable or incomplete. The domain knowledge of an expert operator might be wrong, either because the relationships between events are extremely complex and not well understood, or because the system under analysis is not behaving as intended or designed to perform. This implies that the specification of a list of diagnosis rules for a specific use case offered by an expert operator, especially the initial version, can be rather poor, both in accuracy and completeness. Indeed, no expert can fully understand the entire domain, specially when considering the type of use cases targeted by mPlane.

For this reason, the Reasoner framework considers the possibility of discovering or learning new diagnosis rules out of the gathered measurements and logged events. In this sense, the set of diagnosis rules is not necessarily static and can eventually be expanded by learning from past experiences, either automatically of by a manual exploration process. Such a learning of new rules can be done by correlating the symptom event to diagnose with all the diagnostic events logged by mPlane which occur at about the same time and are spatially correlated to the problem under investigation, further selecting those presenting causal relations. As the well-known phrase in the statistics domain explains, ``correlation does not imply causation'', which forces the study to perform additional statistical tests to avoid creating incoherent rules. In addition, the correlation and further analysis can be done on top of all the available measurements which show some temporal and spatial relation to the symptom event. The main idea is that by iteratively selecting and filtering the most relevant events and/or measurements which might explain the analyzed problem, the Reasoner can extend the domain knowledge, gradually acquiring new knowledge or learning unexpected network behaviors exhibited in the data, which can ultimately be incorporated into the list of diagnosis rules and into the diagnosis graph.

The learning capabilities of the Reasoner can by no means be fully automated, specially when it comes to the extraction of new diagnosis rules. If the new learned rules are directly taken into the knowledge structure without expert validation and assertion, the diagnosis graph might rapidly get polluted by statistically relevant yet inconsistent diagnosis rules. Therefore, the extension of the knowledge structure associated to a specific use case requires final expert intervention to validate the identified dependencies. To limit the time required by the expert to verify and validate the new generated knowledge, the learning algorithms provided within the mPlane framework try to summarize as much as possible the most relevant informartion related to the generated rules, i.e., they select the most relevant features related to the specific relevant events.

Learning approaches for the Reasoner are described in section 4.

3 The Reasoner in the Practice

This section presents the Reasoner by example, detailing the role of the Reasoner in each of the use cases tackled by the mPlane project, as defined in deliverable D1.1. As we explained before, the role of the Reasoner depends on the characteristics of the specific use case. For troubleshooting support-based use cases, such as DaaS troubleshooting, Web QoE troubleshooting, and anomaly detection and Root Cause Analysis among others, the main role of the Reasoner is to drill down the measurements and interpret the analysis results provided by the analysis modules to find the most probable root causes of the detected performance issues. In the case of generic measurements-based analysis use cases, such as content popularity estimation and content curation, the main role of the Reasoner is to automate the iterative measurements analysis process.

The following subsections include in addition a per use-case definition of some iterative analysis and diagnosis rules, built from the developed experience on the definition and analysis of the measurements generated for each of the use cases. In each case, the associated workflow of the iterative analysis is also presented, as well as some first evaluation results of the iterative analysis process.

3.1 Supporting DaaS Troubleshooting

The goal of this use case is to continuously monitoring the Quality of Experience (QoE) of users accessing content using Desktop-as-a-Service solutions through thin-client connections. Whenever the users experience a poor QoE, the mPlane infrastructure, particularly the Reasoner, acts for troubleshooting its cause and iteratively responds with solutions to improve the overall users' experience.

3.1.1 The Role of the Reasoner

Fig. 4 outlines the role of each mPlane's component in this use case and highlights the input of the Reasoner, based on which troubleshooting decisions are taken.

At first, probes continuously monitor thin-client connections and passively collects IP-level features that can be accessed from the thin-client connection while it is running, 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 are collected on a per-connection basis, i.e., on a per-thin client basis, and within sliding observation time-window.

Periodically, the probe sends the features extracted from a given thin-client connection to the central repository, which stores them for the Analysis module to use. Based on these features, the Analysis module is responsible for classifying the connection, that is, inferring the application running on top of the thin-client connection during a time-window through statistical traffic classification techniques, e.g., Support Vector Machine (SVM).

By combining the information from the Analysis module with the network conditions along the path between the thin-client and the remote server, the Reasoner can eventually infer the temporal evolution of users' QoE. Note that those network conditions are collected in the first place by active (traceroute-like) mPlane probes, which periodically send them to the central repository.

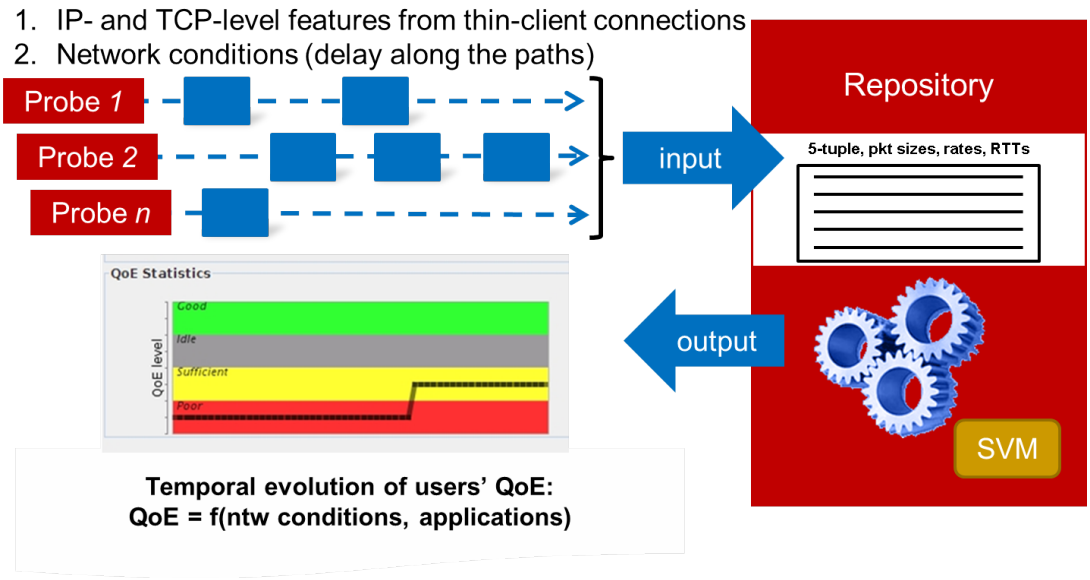


Figure 4: mPlane components for the use case ``Supporting DaaS troubleshooting''. Given a thin-client connection, the Reasoner observes as input the evolution of the user experience over time, and triggers actions based on how the quality evolves.

QoE category	Poor	Sufficient	Good
Audio	$\overline{RTT}$ ≥450ms	120ms< $\overline{RTT}$ <450ms	$\overline{RTT}$ ≤120ms
Data	$\overline{RTT}$ ≥400ms	100ms< $\overline{RTT}$ <400ms	$\overline{RTT}$ ≤100ms
Video	$\overline{RTT}$ ≥70ms	50ms< $\overline{RTT}$ <70ms	$\overline{RTT}$ ≤50ms

Table 1: Threshold values based on the average RTT of the thin-client connections within a given time-window.

3.1.2 Domain-knowledge based Iterative Rules

The Reasoner exploits two information to infer users' QoE: the application running on top of the thin-client connection, and the current delay along the path where the connection is taking place.

Different network conditions lead users to perceive different QoE when interacting with the same application. On the other side, under the same network condition, users may perceive different QoE depending on the specific application in use.

Given the class of application run by the thin-client user, the Reasoner compares the average Round Trip Time of the connection within an observation window ($\overline{RTT}$) against a set of threshold values, and returns a QoE category. Threshold values are set for each class of applications, i.e., Data, Audio, Video, and are based on latency values. To set the threshold values, we run subjective tests for quality assessment by following the Absolute Category Rating method as formalized in ITU-T Rec. P.910 with the help of fourteen people. As a result, for each class of applications we were able to identify requirements in terms of Round Trip Time (RTT) values that make the users experience a good, sufficient or bad quality of the thin-client connections. Table 1 reports on such threshold values.

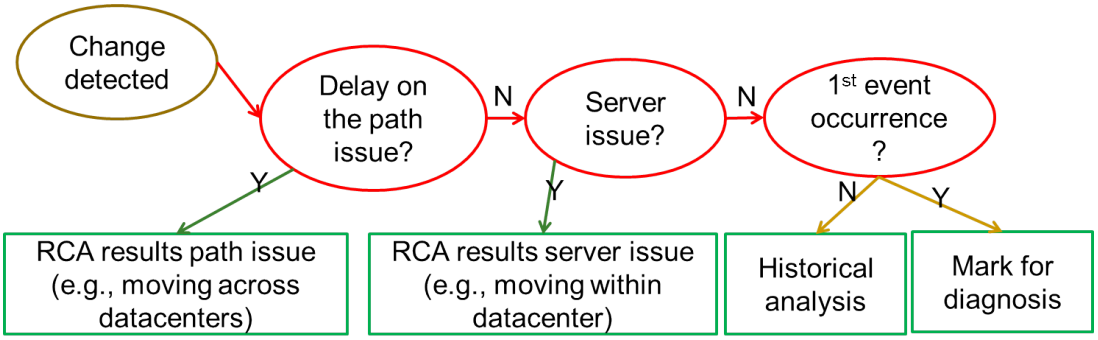


Figure 5: Desktop-as-a-Service troubleshooting: Diagnosis analysis graph. A change is detected in case of poor QoE of a user running a thin-client connection.

3.1.3 Diagnosis/Iterative Analysis Graph

Whenever the Reasoner detects a poor QoE for a user running a thin-client connection, it first tries to identify which is the node causing the bottleneck along the path (included the two end nodes of the connection). To do that, it interacts with the mPlane Supervisor to instrument probes for running latency measurements on the path between the thin-client user and the remote server. At the same time, the Reasoner can interact with the Repository to look for the same information, in case the measurements are already running on the probes.

In case the result of the measurements returns that the bottleneck is due to a node along the path, the Reasoner tries to circumvent the responsible node by migrating the remote server to another datacenter. Alternatively, if the bottleneck is due to the remote server, the Reasoner triggers the migration of such server within the same datacenter, to offload the machine where the service is currently running. While doing that, the Reasoner keeps a history of when such events occur, to find patterns and be preemptive in the future -- e.g., a given connection runs into the same issue with a known temporal periodicity.

An overall schema of the analysis graph is provided in Fig. 5.

3.1.4 First Evaluation Results

Here we describe the results in detecting the applications running on top of thin-client connections by means of statistical classification techniques. We also provide an overview of our dataset collection and testbed setup: please refer to D5.1 for more details.

We adopted a supervised approach. First, we gathered the models of our target applications by collecting a set of Remote-Desktop-Protocol (RDP) flows that served as *training set*. Then, we tested the validity of such models on another set of RDP flows that served as *testing set*. Table 2 and Table 3 report on the training and testing sets being collected, respectively.

Our testbed, depicted in Fig. 6, replicates a real DSL subscriber access thanks to the real DSL-Modem, MSAN and BRAS equipment. The client establishes RDP connections towards the server and remotely performs actions, such as watching multimedia content, viewing and editing documents, web-browsing. On the path between the Broadband Remote Access Server (BRAS) and the server we set up our monitoring probe and a machine running NetEM [17], a tool that provides net-

Category	Apps	Duration [sec]	Bytes [MB]	Network conditions down/uplink [bps]	delay [ms]	loss [%]
Audio	WMPa	1400	37	6M/1M	--	--
	VLCa	1400	35	6M/1M	--	--
Data	AdR	1400	63	6M/1M	--	--
	PPT	1400	73	6M/1M	--	--
Video	VLCv	1400	703	6M/1M	--	--
	WMPv	1400	344	6M/1M	--	--

Table 2: Training set composition. AdR stands for adobe reader, while PPT for Powerpoint presentations. VLC is marked as VLCa and VLCv when used for generating audio and multimedia content, respectively. Same consideration holds for WMP.

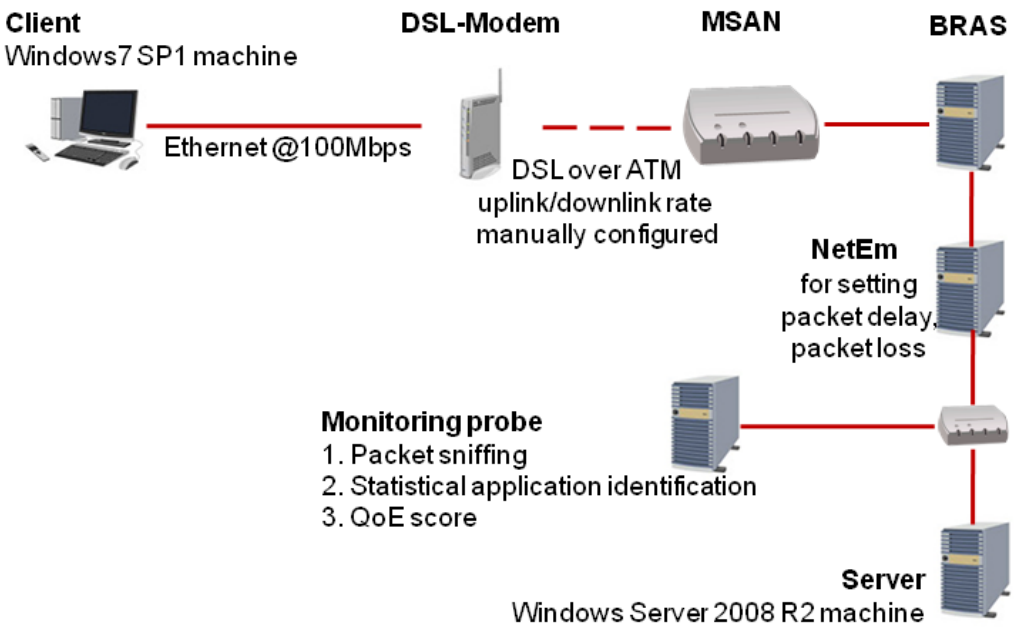


Figure 6: Overview of our testbed setup.

Category	Apps	Duration [sec]	Bytes [MB]	Network conditions down/uplink [bps]	delay [ms]	loss [%]
Audio	WMPa	980	13	6M/1M	--	--
		980	13	3M/512K	--	--
		980	13	1.5M/256K	--	--
	VLCa	980	26	6M/1M	--	--
		980	29	3M/512K	--	--
		980	26	1.5M/256K	--	--
	Skype	560	15	6M/1M	--	--
Data	AdR	980	77	6M/1M	--	--
		980	45	3M/512K	--	--
		980	26	1.5M/256K	--	--
	PPT	980	47	6M/1M	--	--
		980	39	3M/512K	--	--
		840	24	1.5M/256K	--	--
	WebB	560	83	6M/1M	--	--
Video	WebF	560	274	6M/1M	--	--
	VLCv	980	498	6M/1M	--	--
		980	277	3M/512K	--	--
		980	153	1.5M/256K	--	--
	WMPv	980	302	6M/1M	--	--
		980	265	3M/512K	--	--
		980	153	1.5M/256K	--	--
		140	26	6M/1M	10	--
		140	27	6M/1M	20	--
		140	19	6M/1M	40	--
		140	25	6M/1M	80	--
		140	23	6M/1M	120	--
		140	21	6M/1M	160	--
		140	28	6M/1M	--	1
		140	20	6M/1M	--	2
		140	7	6M/1M	--	3

Table 3: Testing set composition. WebF stands for web pages with embedded Flash videos, WebB for other kinds of web pages. VLC is marked as VLCa and VLCv when used for generating audio and multimedia content, respectively. Same consideration holds for WMP.

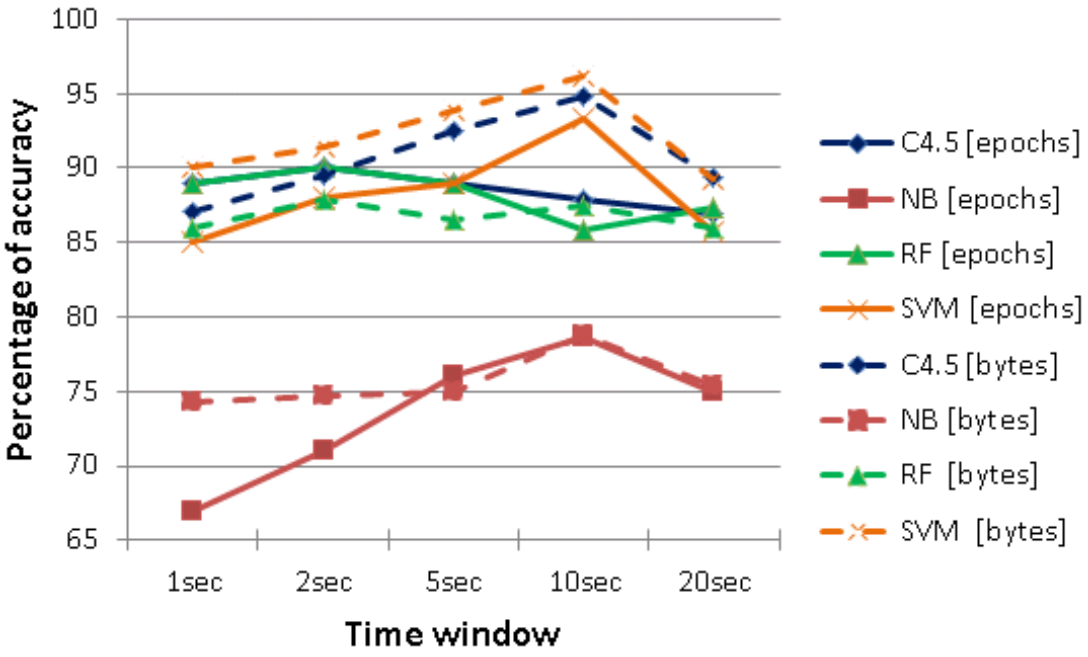


Figure 7: Accuracy by epochs (solid lines) and by bytes (dashed lines) of the statistical mechanisms: training and testing are collected under the same network conditions of bandwidth 1Mbps uplink and 6Mbps downlink, no impairments.

work emulation functionality and allows us to introduce delay and loss on the connection between the client and the server.

We started investigating how four statical techniques -- namely Support Vector Machines (SVM), Random Forest (RF), Naive Bayes (NB) and Decision Tree (C4.5) -- perform when the dataset used for training and the one used for testing include the same class of applications and are collected under the same network conditions. To this purpose, we considered as testing set the data collected with a bandwidth of 1Mbps uplink and 6Mbps downlink and no impairments on the network by running the same class of applications as the one included in the training set. Given an epoch, we observe the traffic flowing into the RDP connection, extract the features for each epoch and classify it.

Fig. 7 reports the accuracy results in terms of epochs and bytes that we obtained by considering time-windows (epochs) that ranges from 1 to 20 seconds. We achieved maximum accuracy (over 90%) when applying SVM on a time-window of 10 seconds, both in terms of epochs and bytes. Interestingly, by using a ten-second window, the accuracy by epochs of the algorithms is lower than the accuracy by bytes. Although there are exceptions, this statement also holds with other time windows, thus suggesting that the algorithms mis-classify mostly epochs that exchange few bytes.

To exemplify this, Table 4 shows the misclassification results of applying SVM to a ten-second time window, both in terms of epochs and bytes. As the table suggests, the misclassification of the applications for which the techniques have received training mostly happens when the users are reading documents (e.g., PDF files) or watching presentations (e.g., PPT files), whereas the classification of video activities, which are responsible for the biggest amount of bytes being exchanged, are correctly classified.

The table also reports the classification results for three applications for which the techniques were

%		VLCa	WMPa	AdR	PPT	VLCv	WMPv
Audio	VLCa	**100**	--	--	--	--	--
	WMPa	--	**99/95**	--	1/5	--	--
	Skype	100	--	--	--	--	--
Data	AdR	--	--	**84/93**	13/6	--	3/1
	PPT	--	1/1	22/10	**77/89**	--	--
	WebB	--	5/<1	45/35	50/65	--	--
Video	VLCv	--	--	--	--	**100**	--
	WMPv	--	--	--	--	--	**100**
	WebF	--	--	20/12	9/6	71/82	--

Table 4: Classification results (percentage) for the SVM algorithm with a ten-second time window. On the left, accuracy by epoch. On the right, accuracy by byte.

not trained for, namely Skype audio calls, web traces with and without multimedia content (WebF and WebB, respectively). In these cases, SVM still assigns the applications to the category they actually belong: for instance, 45% of the WebB epochs are classified as AdR and the remaining 50% as PPT, both belonging to the category *Data*. In fact, when considering categories only, the overall accuracy of the classification mechanisms increases, even for applications for which the techniques have been trained with (up to 97% in terms of epochs and 99% of bytes for AdR application).

3.2 Estimating Content and Service Popularity for Network Optimization

The goal of this use case is to optimize the quality of experience of the user and the network load by inferring the expected-to-be popular contents and identifying optimal objects to cache in a given portion of the network. To achieve this goal, we exploit the mPlane architecture in order to collect a large number of online traffic information requested by the users in several points in the network. The acquired information is exploited in order to predict the content popularity and suggest efficient caching replacement strategies to the Reasoner.

3.2.1 The Role of the Reasoner

Differently from use cases that include troubleshooting and where iterative reasoning is practically mandatory, the role of Reasoner is basic for the content popularity estimation use case. Practically, it orchestrates the two different analysis modules that monitor and estimate the poularity evolution of contents observed in certain portions of the network. Its only task which may require some iteration is the localization of mobile devices when we want to enable the proactive prefetching of labeled-as-popular contents.

In its current status, the Reasoner gets the list of (estimated) popular contents from the analysis modules that run continuously on the repositories, together with information about the network portion (i.e., the probe) in which such content was observed. Then, the advanced knowledge about the popular contents in terms of location, which could be very valuable for mobile users to whom we can proactively prefetch content that has a local signification with respect to the place in which they are.

3.2.2 Analysis Workflow

Fig. 8 outlines the role of each mPlane's component, and more specifically, the WP4 analysis modules and their interactions with the repository and the Reasoner.

At first, probes located in different points of the network continuously collect information about the requests of the users, and stream requests to the central repository. For each request we store the associated timestamp and the network location of the probe (**Step 1**). Based on these features, the analysis modules are responsible for predicting the future popularity of each requested contents at each part of the network (i.e., the probe location). This task is accomplished by employing two different modules: the first one, named Popularity Classifier, takes as input the popularity history for a content (**Step 2**), it generates a signature for its request arrival process using the Heterogeneous Mixture Modelling technique, and classify such content in a Hierarchical Clustering Structure, that is stored at the repository (**Step 3**). In parallel, the Online Predictor, for each observed content explores the Hierarchical Clustering Structure (**Step 4**) to find the popularity pattern which maximizes a likelihood function (**Step 5**). Once the popularity pattern has been found, we use it to predict its future popularity. Thus, if the number of future views overcomes a static threshold N, an event is triggered to the Reasoner (**Step 6**) to notify which contents are becoming popular and where they are. Then, the Reasoner may query the repository to obtain the IDs of mobile devices, base stations and caches that are located within the same area of the probe, and may be interested

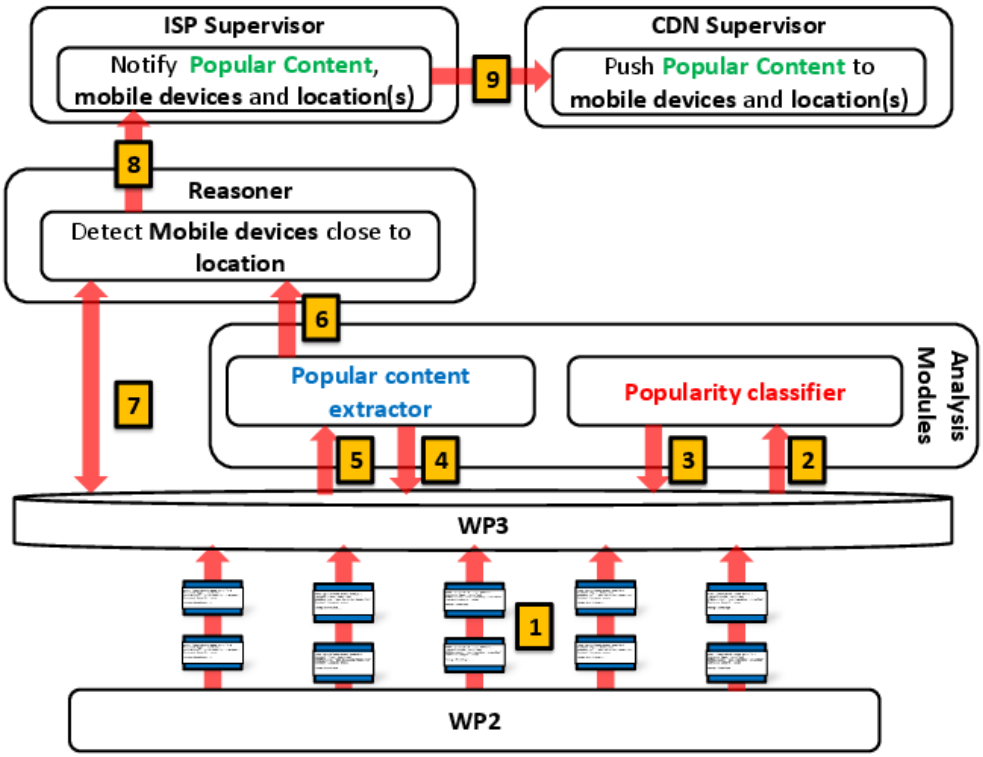


Figure 8: The analysis and reasoning modules composing the WP4 layer for the ``Estimating Content and Service Popularity for Network Optimization''. Given a set of users' request for videos, the Reasoner observes as input the evolution of the popularity of videos over time, and triggers actions based on how the popularity is predicted to evolve.

at prefetching popular contents (**Step 7**). Then, for each popular content, its ID, together with the location of the probe and, possibly, a list of devices are forwarded to the supervisor of the ISP (**Step 8**). The ISP supervisor will exchange such information with other supervisors, e.g., the supervisor of a CDN, to let it know which contents may be worthful to proactively push to its caches (**Step 9**).

3.2.3 Domain-knowledge based Iterative Rules

The Reasoner can combine the knowledge about the network topology, i.e., where caches and probes are located in the network, to identify a global list of objects to cache in a hierarchical caching scheme. Alternatively, the Reasoner can obtain a list of mobile devices in such area, and, if available, detailed information about their battery charge and connection type. With such information a service could think of delivering its contents to mobile devices by proactively pushing them when the battery charge is above a certain threshold, and the connection to Internet is provided through a WiFi access point.

Second, the Reasoner may exploit the information about which content are becoming popular and where, and combine this with the size of the cache in a given network location, for instance, to identify the list of top objects that are worth caching, according to an accuracy metric.

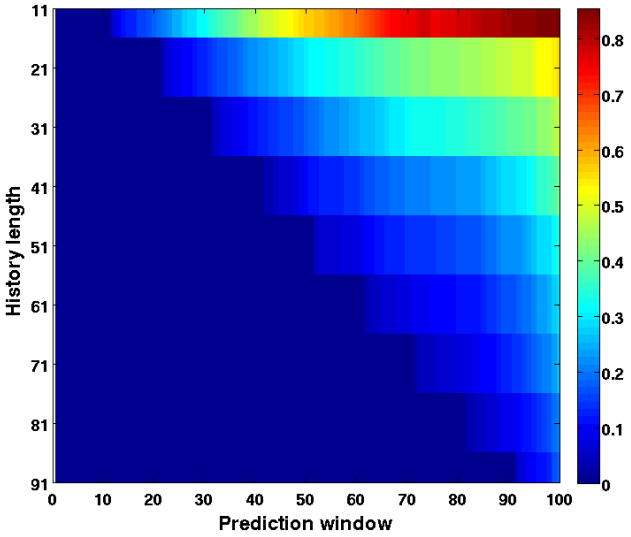


Figure 9: Mean percentage reconstruction error when predicting the future popularity of the video based on the history length. Note how the reconstruction error increases as we predict further in the future.

3.2.4 First Evaluation Results

Here we describe the results in inferring the evolution of content requests over time by means of the predictor as described in D4.1. In particular, we run our prototype on a commercial ISP anonymized trace reporting the requests to YouTube videos watched by a population of 28,000 users.

We adopted a supervised approach, based on heterogeneous mixture models. First, we gathered the models of our target applications by collecting a set of requests for YouTube videos over time (grouped by day) that served as training set. Then, we tested the validity of such models on a subset of 2,000 requests to videos available in our trace.

We started investigating the accuracy of our algorithm in predicting the future popularity of the videos, given that the algorithm has seen the first X data samples of the requests of the videos over time, where X ranges from 11 to 91, with step 5. Fig. 9 reports the results in terms of Mean Percentage Error (MPE), that is the ratio between the absolute estimation error (the difference between the estimated and the real requests) and the number of requests that the video actually gets in the data sample we are trying to predict.

As shown, the algorithm shows good accuracy (MPE is below 20%) in predicting the popularity of the objects in the short-term (e.g., up to the next ten data samples in the future), whereas the accuracy degrades the more we try to predict long-term, especially when the known history is very little (e.g., over 80% of error when the algorithm tries to predict up to 80 data samples ahead in the future, based on the knowledge of the first 11 data samples).

3.3 Passive Content Curation

There is more content on the web today than what users can individually discover and consume. This fact gave birth to a plethora of *content curation* tools and services. We refer to *content curation* as the process of identifying and organizing online content so that users can easily focus on what is relevant and interesting. A promising family of content curation tools relies on crowdsourcing with Reddit and Digg being prominent examples. For instance, in Reddit, users submit a link to their favorite content (e.g., a video or a news article), and the ``crowd'' of the other users rate it. The higher the rate, the higher the chance that Reddit shows the link on their homepage. This results in a platform that tracks the most relevant content on the web according to the reddit community.

This use case takes a different approach to content curation and demonstrates how mPlane can be used to provide such a service based on the only passive observation of network content traffic. Instead of relying on users to actively rate content, we infer users interests from clicks. We beleive that clicks are a good measure of interest because users often have an idea about what they are about to click on (because they saw a preview, a friend recommended the link etc). Once the clicks are inferred, we track the evolution of their timeseries to compile a digest of the web URLs that are likely to attract the attention of the crowd at a given moment in time. This digest of URLs is then presented in a nice web interface to the final users.

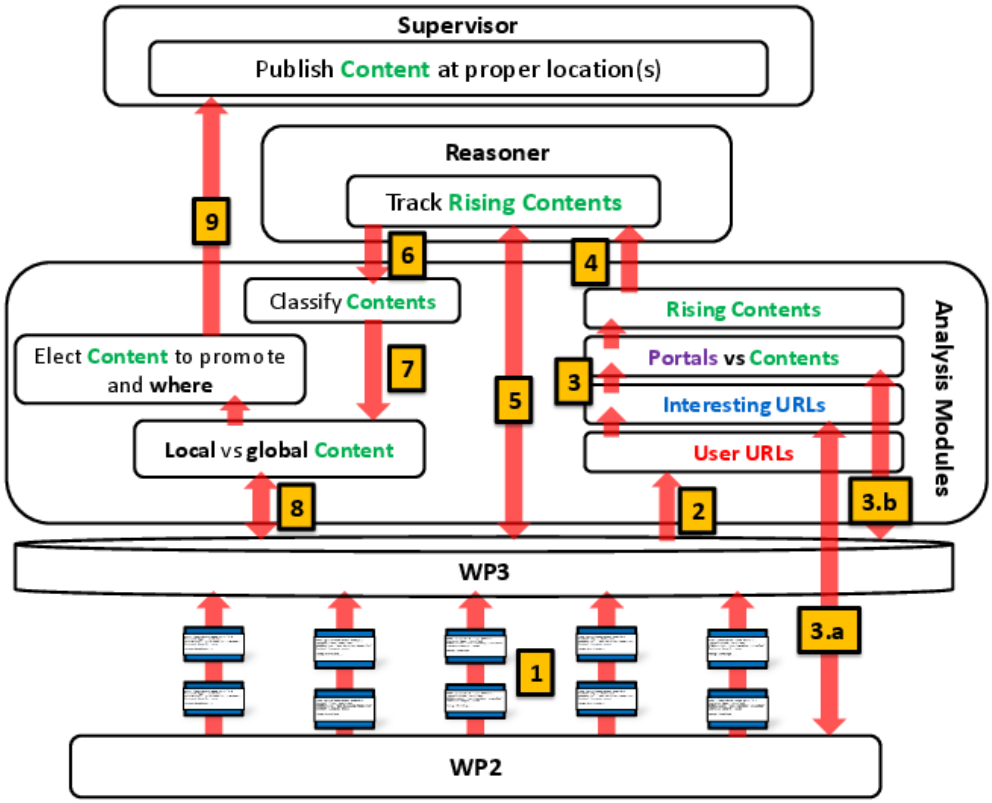


Figure 10: The analysis and reasoning modules composing the WP4 layer for the ``Passive Content Curation'' use-case, and the workflow involving them.

3.3.1 The Role of the Reasoner

This use case is different from use cases that include troubleshooting and where iterative reasoning is more than a must. At this point of the project, the role of Reasoner is still basic for the content curation use case. It consists so far mainly in orchestrating the different analysis modules to track rising contents and take decisions about which content URLs to promote to users.

In its current status, the reasonnr gets the list of interesting URLs that are getting the attention of the crowd from the analysis modules that run continuously on the repositories. It starts then to follow them more closely. The goal is to elect among these URLs the ones that will make it to the final promotion web-page. For the followed URLs, the Reasoner ask the repository to get their popularity and timeseries. Our promotion algorithms are basic so far, we leave their enhancement for future work. For now, we elect a content as soon as it gets more than N visits, with N being a number statically defined at the Reasoner.

Advantageously, the Reasoner infers advanced knowledge about the promoted URLs in terms of location. For instance, it is valuable to guess what content is local to the region (e.g., nearby shops and restaurants, or local news) as opposed to contents that are valuable nationwide. In order to get this information, the Reasoner must couple the historical information about content visits with their geographical source. For instance, this geographical information could be very valuable for mobile users to whom we can provide suggestions of content that has a local signification with respect to the place in which they are.

In the future, the Reasoner should request the analysis module of the content popularity estimation use case, to get more information about the future evolution of a content URL that it is following.

3.3.2 Analysis Workflow

We despict the workflow for this use case in Fig. 10. The probes extract HTTP logs from the network and stream them to the repositories (**Step 1**) where online analysis modules run continuously to exctract *user* clicks (**Step 2**), infer *interesting* clicks (clicks that are likely to attract other users' attention) and clicks that pertain to single *content* items (as opposed to *portals* that aggregate multiple content items) (**Step 3**). Iteresting rising contents are then notified to the Reasoner through events (**Step 4**). The Reasoner rebuilds the popularity evolution of each content by querying the repository (**Step 5**), and launches the classification analysis module to group contents by categories, e.g., to distinguish videos from news (**Step 6**). Each popular content is then pushed to the module in charge of understanding how spatially spread is its popularity (**Step 7**). This action is performed by querying the repository, i.e., by asking at how many different network spots (probes) such content was seen (**Step 8**). Finally, the interesting contents together with their locations are notified to the supervisor (**Step 9**).

Note that some of our analysis modules contain some basic reasoning. This is the case for the interesting URLs module as well as the ``content versus portal'' module. The first module takes as an input user URL visits (user clicks) as elected by the user URLs filtering module. Upon the event of the election of a user URL, we use an active probe (a web scraper, as represented in Fig. 10 by **Step 3.a**), to query the page looking for social meta data, in order to understand whether this page is potentially interesting for other users or not The second modules analyzes the interesting URLs as well as their timeseries (**Step 3.b**) and learns in order to detect which URLs pertain to content aggregation URLs (e.g. the front page of an online news website that compiles a lot of news) and

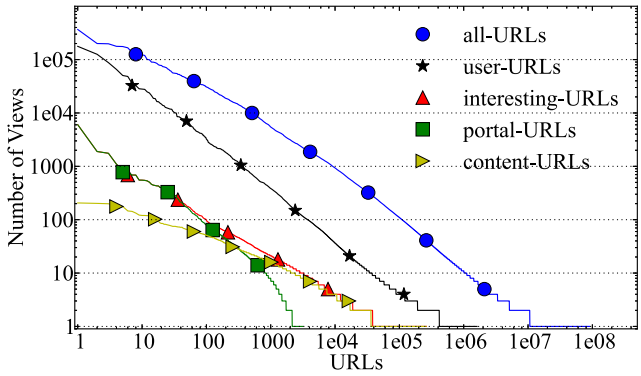


Figure 11: URL popularity before and after filtering (3 days of HTTP logs).

which ones relate to a single content item (e.g., a particular news). However, since these modules can work continuously on the repository and do not need to use other sources of data, we decided finally to include them as analysis modules. They will be introduced in details in the next deliverable D4.3.

3.3.3 First Evaluation Results

We build a preliminary prototype of our use case. We present in this section some early results that this prototype allowed.

In particular, we run our prototype on a commercial ISP anonymized HTTP log. We show in this section how the different modules process the raw data till we get to the URLs that our system would promote. We also characterize how different is the view that our media curation system has compared it with that of the top two influencers of Internet visits today: Google and Facebook. We also run our prototype live in a large campus network. A very preliminary version of the deployment and its output can be watched here: http://youtu.be/aCOIUcrlA8E.

3.3.3.1 Applying our system on HTTP logs

We apply the first module on three days of the commercial ISP http trace to extract user-URLs. Our second analysis modules runs on top to obtain interesting-URLs. Finally, the content vs portal module splits these URLs into portal-URLs and content-URLs.

Fig. 11 shows the popularity of all-URLs, user-URLs, interesting URLs, content-URLs and portal-URLs. The x-axis reports the rank of URLs, while the y-axis reports the actual number of visits of each URL, i.e., its popularity. These results refer to a subset of 3 days of our trace. Both axes are log-scale. Observe that the distribution follows a typical heavy-tail law, as many URLs show a low popularity: only 10,000 out of millions URLs have a popularity larger than 1000. The user-URLs, i.e., the URLs intentionally clicked by users, represent a tiny fraction of all the observed URLs in the trace. Our method detects that among the 190 million URL requests in this portion of our trace, only 3.4% correspond to visits to actual user-URLs. Among these user-URL visits, only around 6.5% of them were pointing to interesting-URLs, corresponding to 0.22% of the total number of visits.

The 3,191 detected portal-URLs cumulate 57,794 visits (18.1 visits on average over 3 days), and the

260,327 classified as content-URLs cumulated 373,147 visits (1.43 visits on average). Fig. 11 shows that the portal-URLs are represented by a very steep curve, meaning that a few portals attract a large fraction of views, leading to a short tail. Content-URLs instead have much lower popularity in general (as the most popular content-URL has 200 views), and a much heavier tail. Finally, out of the 90M distinct all-URLs, only 0.28% of them correspond to content-URLs. This shows the importance of our heuristics to find such a ``needle in the haystack''.

3.3.3.2 How diverse is the content that gets promoted

Our use case, by design, aims at capturing rich and diverse content in order to present it to users. We now study how diverse is the content that it promotes and put it in perspective with that promoted by Google and Facebook, both in terms of the number of distinct websites to which they generate visits, as well as in terms of content categories. For this analysis, we focus on around 260,000 interesting URLs extracted from 3 days of our HTTP logs.

To group these URLs in different categories, our categorization module employs the Alexa categorization APIs provided by Amazon.[1] These APIs provide for each URL a list of categories that Alexa has gathered though its crawling and web analysis. Employing data mining techniques (e.g., analyzing the links on the crawled pages to find related sites), Alexa continuously updates a database of web categories based on Open Directory.[2] We also considered using other solutions, such as AlchemyAPI[3] and BlueCoat[4], but we had questions about Alchemy's accuracy and Blue offers no APIs.

We apply the Alexa categorization API on the interesting URLs set shown in Fig. 11. The API successfully categorized only around 45% of them. To study the sites promoted by Google and Facebook, we extract two subsets from these interesting URLs. The first pertains to URLs whose referer was Google, the second to URLs whose referer was Facebook.

Table 5 summarizes our results. The table shows (1) the number of distinct URLs, (2) their mean popularity, (3) the number of distinct hostnames (websites) (4) the number of URLs that the Alexa API successfully categorized, and finally (5) the number of distinct categories for different URL subsets: *All interesting URLs*, those promoted by *Google* and those promoted by *Facbeook*. In addition, we report the numbers for 4 other subsets that express what we would have promoted:

Top 10,000: We assume that we promote the most popular 10,000 content-URLs over the 3 days. The least popular URL in this set cumulated around 4 visits.

Top 3,000 daily: We assume that we promote, each day, the most popular 3,000 content-URLs only within that day. Since some pieces of content remain popular from one day to the other, this set contains all in all 7,144 distinct content-URLs. Each day, the least popular content-URLs in this set got exactly 6 visits.

Pop n where n=2,3,5: We promote a content-URL as soon as it cumulates more than n visits. This gave us a total number of 7,683 URLs for n=5, 23,599 for n=3 and 61,964 URLs for n=2.

First in terms of discovered hostnames, the table shows that the interesting URLs that we see cover as large as 36,501 distinct websites and 3736 distinct categories. The table also shows that Google

[1] `http://aws.amazon.com/awis/`
[2] `http://www.dmoz.org/`
[3] `http://www.alchemyapi.com/`
[4] `https://sitereview.bluecoat.com/`

URLs	Interesting URLs	Google Search	Facebook	Top 10,000	Top 3000 daily	Pop 5	Pop 3	Pop 2
URLs	264,134	57,191	11,193	10,000	7144	7,683	23,599	61,964
Mean popularity	1.632	2.077	4.136	8.672	12.780	10.082	5.51	3.337
Hostnames	36,591	19,339	4,011	1,126	797	920	2,068	3,979
Categorized	116,029	26,494	4,179	5,199	3,849	4,056	11,608	28,399
Categories	3,736	2,397	581	607	445	500	1,048	1,828

Table 5: Observed categories (3 days).

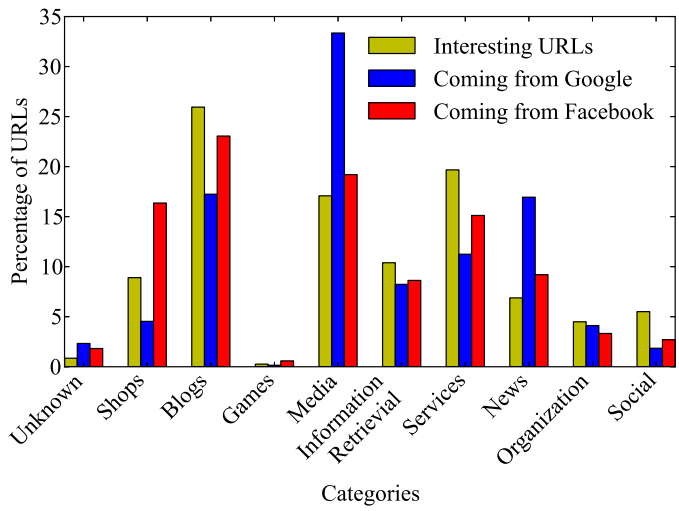


Figure 12: Interesting URLs categories (3 days of our trace).

and Facebook have an extremely large diversity. Although Google-promoted URLs are only 20% of the total URLs, they cover as many as 55% of the total observed hostnames. This holds as well for Facebook for which almost each second facebook-promoted URL belongs to a different website.

However, our service is aimed at a different niche compared to Google and Facebook. Its goal is to promote general purpose popular content that is likely to attract the crowd's attention, while Google and Facebook's content is likely more tailored to an individual user's needs. This is to an extent confirmed by looking at the mean popularity of both Facebook and Google-promoted URLs. Indeed, Google-promoted URLs have a mean popularity of 2 and those of Facebook received 4 visits on average.

Now if we look at what our service promotes, we find that although it focuses only on popular content-URLs, it still has a wide diversity. The top 10,000 URLs, for instance, have as many content categories as their equivalent number of Facebook URLs, and come from as different as 1,100 different websites. This is about more than the double of websites that Google news uses to build an entire news search engine for the country of the HTTP logs.

To better visualize the Alexa API categories which are too much detailed, we assign each of the found categories to one of these main content types: Shops, Blogs, Games, Media (video and streaming), Information Retrieval (e.g. search engines, wikipedia, stackoverflow), Services (e.g. travel agencies), News, Organizations and Social networks. Fig. 12 shows the distribution of these categories for Interesting URLs observed by our service, Google and Facebook URLs. The figure shows that three sets have a large diversity, with services, news, media, blogs and shops being the most domi-

nant categories. Compared to the subsets promoted by Google and Facebook, the interesting URLs have a larger portion of blogs, and Google is promoting a larger portion of media. The most dominant Facebook category is also Blogs.

We conclude that the nature of the content that our use case promotes is diverse as we expected.

3.4 Measurements for Multimedia Content Delivery

3.4.1 The Role of the Reasoner

In this scenario, the Reasoner is working in a troubleshooting/diagnostic role. The process is triggered by measured quality impairments, and the output of the diagnostics is the estimation of one or more causes responsible for the impairment. The Reasoner outlined here is used by the network operator (ISP) providing access for clients requesting multimedia (streaming) content. The operator is primarily concerned about:

1. Most importantly, if the bottleneck or problem is within the operator's network or not. If not, then the operator may notify and assist the content provider, and inform concerned customers.

2. If within, then what causes the problem - in this case, the precision of the diagnostics depends on the auxiliary information and measurements that is provided to augment content-delivery specific measurements. Network operators usually have diagnostics tools and procedures already in place to deal with internal network problems. Thus, in many cases the mPlane Reasoner just triggers this process by proving that a capacity or other problem is present in the network, and points at a location or component for starting the investigation.

In the following sections, we describe a basic setup that does not leverage on existing auxiliary measurements, carefully noting the points where the precision and usefulness (for the operator) of the reasoning process can be enhanced by providing additional measuements and data. Such data is very likely to be available, in a form specific to the operator's existing monitoring systems.

3.4.2 Probes, measurements, pre-processing

The scenario uses measurements from both active and passive probes, in the following roles:

Active content probes simulating end-user transactions scheduled by the Reasoner. They are deployed at different locations within the ISP's network, trying to achieve a good coverage of network paths between subscribers and the content provider. The probes are configured to request and download multimedia content (e.g. YouTube or OTT videos) and evaluate network, QoS (and partially QoE) characteristics. Their ``playlist''s are synchronized, so all probes are requesting the same content at the same time. Metrics include DNS resolution and server delay, best achievable bitrate, number of video stalls etc. Importantly, these probes also provide the names and IP addresses of content servers providing the requested media clips. Some of such probes are configured into ``crawling'' mode to query for different media clips in order to expand the list of known content servers.

Active "line" probes are used by the operator for performance testing of internal links (e.g. by transferring static HTTP content from known internal servers, or performing other simple measurements such as ping to evaulate line performance). These probes operate against known **responders** (or *reference servers*), also operated by the provider to help identify internal bottlenecks. The concept is to facilitate testing the performance of internal links or paths, and provide auxiliary measurements for evaluating the stream probes' results. Obviously, if the provider has other means

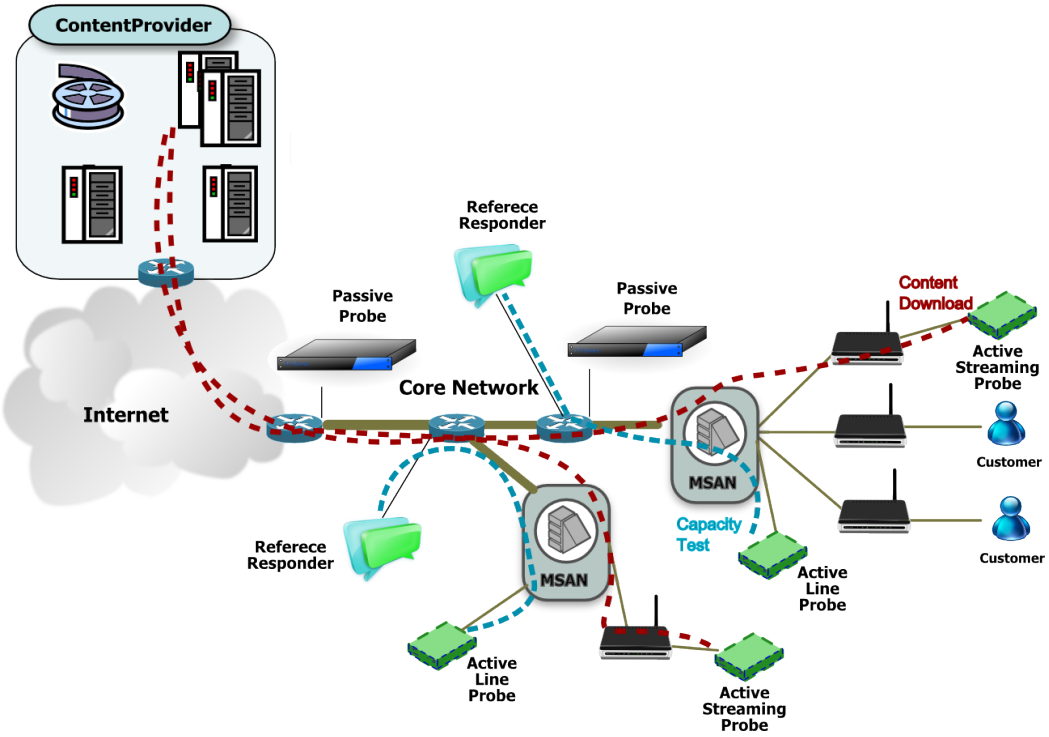


Figure 13: Measurements for the multimedia content delivery case

to obtain equivalent data (e.g. interface counters monitoring on routers), or a link utilization monitoring system, those measurements are also taken into account by the reasoning process in a similar fashion.

passive tapping probes (e.g. Tstat, DATI) are deployed to central locations where they observe as much subscriber - content provider traffic as possible, and measure basic TCP transmission properties for each connection (e.g. average bitrate, volume transmitted, delay, retransmits etc.)

Using the server addresses obtained from active measurements, mPlane builds a list of *known content servers*, which is subsequently used to filter the passive probes' logs for transactions. It must be noted that certain content providers (e.g. Google/YouTube) do not use dedicated servers for multimedia content, and this traffic is interspersed with other Web or Cloud traffic. Nevertheless, filtering these transactions by server IP is useful to detect overload issues caused by e.g. sudden spikes in demand.

It must also be noted that this scenario assumes that apart from IP addresses, port numbers and number of bytes transferred, no deeper information can be extracted by the passive probes. This models the case when the content is transferred over HTTPS or the passive probes' DPI modules do not support the streaming protocol used.

Measurement readings are stored into *Repositories*, that perform the following functions:

- All measurement results are stored historically (along with metadata such as probe location and the media ID queried) for baselining
- Using the filtered list from passive logs, the Reasoner computes aggregated statistics characterizing traffic between the servers and subscribers served by the ISP. Such statistics include number of queries, approximates of total bandwidth served and min/max/avg server delay

etc. for given intervals of time, and the computation is off-loaded to the repositories as much as possible.

In cases when subscriber addresses can be tied to geographical regions or network locations, (e.g. in the case of regional DHCP servers or address pools), the filtering and aggregation can be futher refined by client (subscriber) address data, and a subsequent filtering pass can be made over the transfer logs provided by passive probes.

3.4.3 Analysis Workflow

The analysis is triggered by one or more active probes reporting performance degradation. The following steps are taken:

1) knowing the location of the probe(s) in question, the Reasoner considers

   a) relevant link/path/interface utiliztation or *active line probes* measurements that reside on paths between the streaming probe and the peering gateway towards the content provider. If these measurements show significant overload, packet loss or performance degradation from baseline values, then the diagnosis concludes. In this case, the the accuracy of the diagnosis direcly depends on the level of detail these auxiliary measurements provide.

   b) passive probe transfer logs are filtered with client (subscriber) IP addresses whose Internet traffic is routed via the same path the streaming probe uses. These passive measurements are aggregated for metrics that indicate link congestion (e.g. retransmits), and the result is compared against baseline values.

   Network topology modeling follows established practices with the provider. ISP internal networks are usually built following a simple, uniform topology. The *core* is either a logical ring, star or mesh with a small number of nodes, with regional PoPs (aggregation centers) attached to it in one or two layers (sec. 3.5.2 explains diagnosing typical ISP networks in detail). Identifying paths from a device located in a PoP to an Internet exchange point is usually a trivial task for the ISP.

2) if no internal network bottleneck can be identified, the diagnosis proceeds as follows:

   a) session properties relevant to the server (hop count and RTT) are compared to other servers' similar metrics providing the same content item. If one or more server addressess can be pinpointed with excessive hop count or RTT values (compared to the others), the diagnosis concludes by pointing at improper cache selection on the content provider's behalf.

   b) the Reasoner proceeds by filtering passive probe logs with the list of known server addresses. Traffic-related metrics (e.g. number of TCP sessions, overall traffic volume) are aggregated and compared against baseline. If there's a marked increase in overall traffic, then the degradation was caused by the increased demand for content from the media provider.

3.4.4 Experimental Results

The analysis outlined above was influenced by Fastweb's earlier work and experiences with the IQM VoD monitoring tool.

The results of IQM VoD testing reveal that the actual download speed needed to have a smooth video streaming experience should be slightly higher than what YouTube recommends. Fastweb's practical experience also shows that in cases when this condition is not met, the first step of the analysis should be the checking of bandwidth/utilization of links being used, as poor video streaming experience is often caused by link congestion. Practical experience also underlines that different congestion patterns can be observed during different periods of time (e.g. during business hours). These network problems could take place both in internal and external networks.

In conclusion, it is highly expected that the algorithm, by highlighting key metrics such as routes, hop counts, RTTs etc, would provide important insights for providers in diagnosing and optimising their networks.

3.5 Service Provider Decision Tree for troubleshooting use cases

In the context of a fixed Service Provider's (SP) business, an interesting application of the mPlane infrastructure would be to monitor the Quality of Experience perceived by its customers for some key services (f.i. video services) and, in case the QoE level degrades below a predefined threshold, to start an automatic process that analyzes data and triggers new measurements in order to find the root cause of the problem.

The reference scenario, as depicted in Fig. 14, is based on a typical SP's infrastructure. The SP has built and runs a completely private instantiation of the mPlane infrastructure. This means that all the probes, the repository, the supervisor and the reasoner are under the exclusive control of the SP.

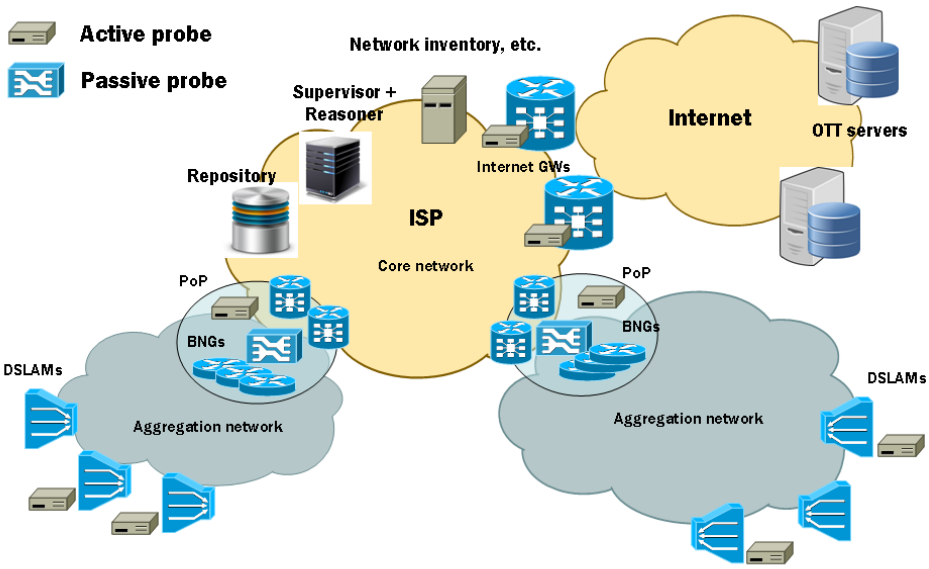


Figure 14: Reference scenario

In order to evaluate the QoE of a specific service (f.i. video streaming) a continuous monitoring infrastructure built upon a set of passive probes is considered the preferable choice. This because passive probes can estimate QoE directly from users' traffic flows, so the information is directly related to what is perceived by the customers. In case the traffic being analyzed uses HTTPS, then the approach outlined in Section 3.4 can be used: it primarily uses active probes to perform measurements and eventually it uses passive probes to extract additional information. Passive probes are placed in multiple vantage points and extract from traffic flows data useful to estimate the QoE. For instance, the average throughput per flow or the Round Trip Time can be usefully used to build performance parameters associated to the monitored service. The passive probes will intercept only a subset of the total traffic, so it's worth underlying that the analysis will have to rely on statistical inference techniques. Besides the set of passive probes, the reference scenario also considers the availability of a set of active probes. The active probes are located directly behind a subset of access nodes (e.g. DSLAMs, OLTs, etc.), within the PoP site and next to the Internet Gateways. These

probes are able to perform active measurements: for instance, Ping/Traceroute or actively requesting video contents and taking measures from Over The Top services (e.g. YouTube). The active probes work on demand, following triggers coming from the Reasoner (through the Supervisor). Their role is to perform measurements on specific network path/segments to help the Reasoner to pinpoint the cause of the problem.

3.5.1 The Role of the Reasoner

The role of the Reasoner is to implement an automatic process to detect a degradation of the QoE perceived by the customers and to pinpoint the cause of the problem, leveraging the data measured by the probes and collected on a distributed Repository infrastructure. The Reasoner will interwork with the other mPlane elements, by means of the Supervisor, to fetch data measured by the probes and to trigger new measurements on-demand. The detail of the analysis performed by the Reasoner can in principle be tuned on the basis of the amount of data it can access and use. As a first step it is considered valuable to be able to decide if the origin of the problem resides inside the SP's network (fault, congestion, etc.) or is due to an external source (origin server fault, upstream provider's fault, CDN policy change, etc.). In case the problem is internal, it's worth considering a further analysis step to identify the affected portion of the network: core network, aggregation network or access network. This can be achieved by considering the distribution of degradation events within the network and the active measurements results triggered by the Reasoner.

An analysis module runs continuously on the repository: it performs a per-DSLAM aggregation of data measured by the passive probes, and compares the results with predefined performance thresholds. After each run, the analysis module sends results to the Reasoner process (through the Supervisor):

- "OK" means that all data are above the threshold; additional information can be sent to the Reasoner: statistical parameters (f.i. mean, variance, percentiles, etc.) could be used by the Reasoner to build trends and refine the thresholds.
- "NOT OK" means that some data are below the threshold; the list of impacted DSLAMs is sent to the Reasoner together with the statistical parameters of the data

When a "NOT OK" is received, the Reasoner is triggered to start the analysis. Being the data aggregated per DSLAM, the Reasoner is triggered only in case of significant degradation episodes, having statistical relevance at the DSLAM level.

The Reasoner also has access to the network inventory to extract:

- Network map
- DSLAM-BRAS relationships
- Active probes availability and location
- Other DSLAM properties (e.g. uplink capacity, etc.)

3.5.2 Diagnosis/Iterative Analysis Graph

Fig. 15 depicts the Diagnosis Analysis Graph.

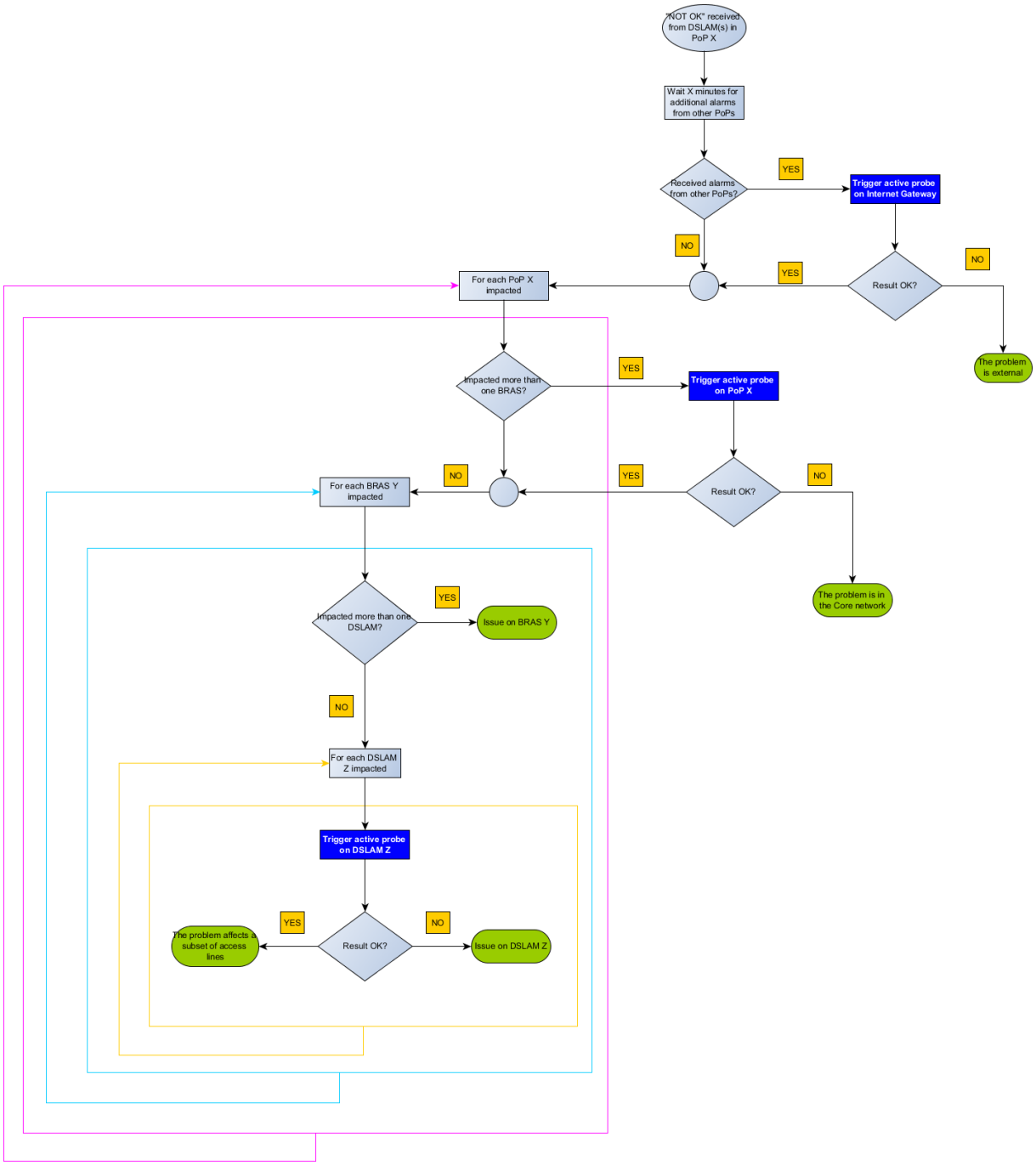


Figure 15: Diagnosis Analysis Graph

The graph describes the actions performed by the Reasoner to find the cause of the problem. The analysis follows a top-down approach, evaluating the possible causes from the most general to the most specific. It is structured in four major blocks, corresponding to the main steps of the analysis:

- Internal or external issue: it uses data collected by the passive probes and it triggers an on-demand measurements on the active probe placed near to the Internet Gateway
- Issue in the Core network: it uses data collected by the passive probes and it triggers an on-demand measurements on the active probes placed in the PoP (these probes take inter-PoPs measures)
- Issue on the BRAS: only data collected by the passive probes are used
- Issue on the DSLAM: both data from passive probes and active probes (connected to the DSLAM) triggered on-demand are used.

The first step verifies if the cause of the problem is internal to the SP's domain or not. If the cause lies outside, we expect to see the problem widespread allover the network. As a consequence the Reasoner initially waits for additional alarms coming from all the installed passive probes (one in each PoP). If multiple alarms are actually received from every PoP, the Reasoner performs an additional test to verify if the problem is really external. If the problem is internal to SP's domain, the next step aims at verifying if the issue is in the Core network (f.i. a congestion event on the geographical links between the PoPs). For this condition to be true, all the measurements taken within the PoP should be affected by the problem: being the data aggregated per-DSLAM, the Reasoner expects to see a QoE degradation for all the DSLAMs under test. If this condition is satisfied, the Reasoner performs an additional verification by triggering an active probe. If the problem is not related to the PoP, the next step is to verify if there's an issue at the BRAS level. Similarly to the previous steps, the Reasoner should verify if all the DSLAMs served by the same BRAS are impacted: if the result is positive then the problem is probably on the BRAS, otherwise it means that single DSLAMs are impacted. The last step verifies if the problem affects the whole DSLAM or only a subset of access lines. The active probe directly connected to the DSLAM is used to perform this control.

3.5.3 Workflow of the Iterative Analysis

The overall workflow is represented in Fig. 16.

After the initial phase, where all the probes and the repository register themselves to the Supervisor, the Supervisor instructs all the passive probes to start measuring. The content type to be monitored (f.i. YouTube), the frequency of reporting, and the location of the Repository are also communicated by the Supervisor to the probes. The passive probes start measuring and storing the values to the Repository. An analysis module periodically reads data from the repository, aggregates them per-DSLAM, and compares the aggregated QoE values to some pre-defined thresholds. This is done PoP by PoP. After the single PoP has been evaluated, data are sent to the Reasoner process via the Supervisor: "OK" or "NOT OK" and the measured values aggregated per-DSLAM. A received "NOT OK" triggers the Reasoner process to start its diagnosis algorithm, described in the previous section.

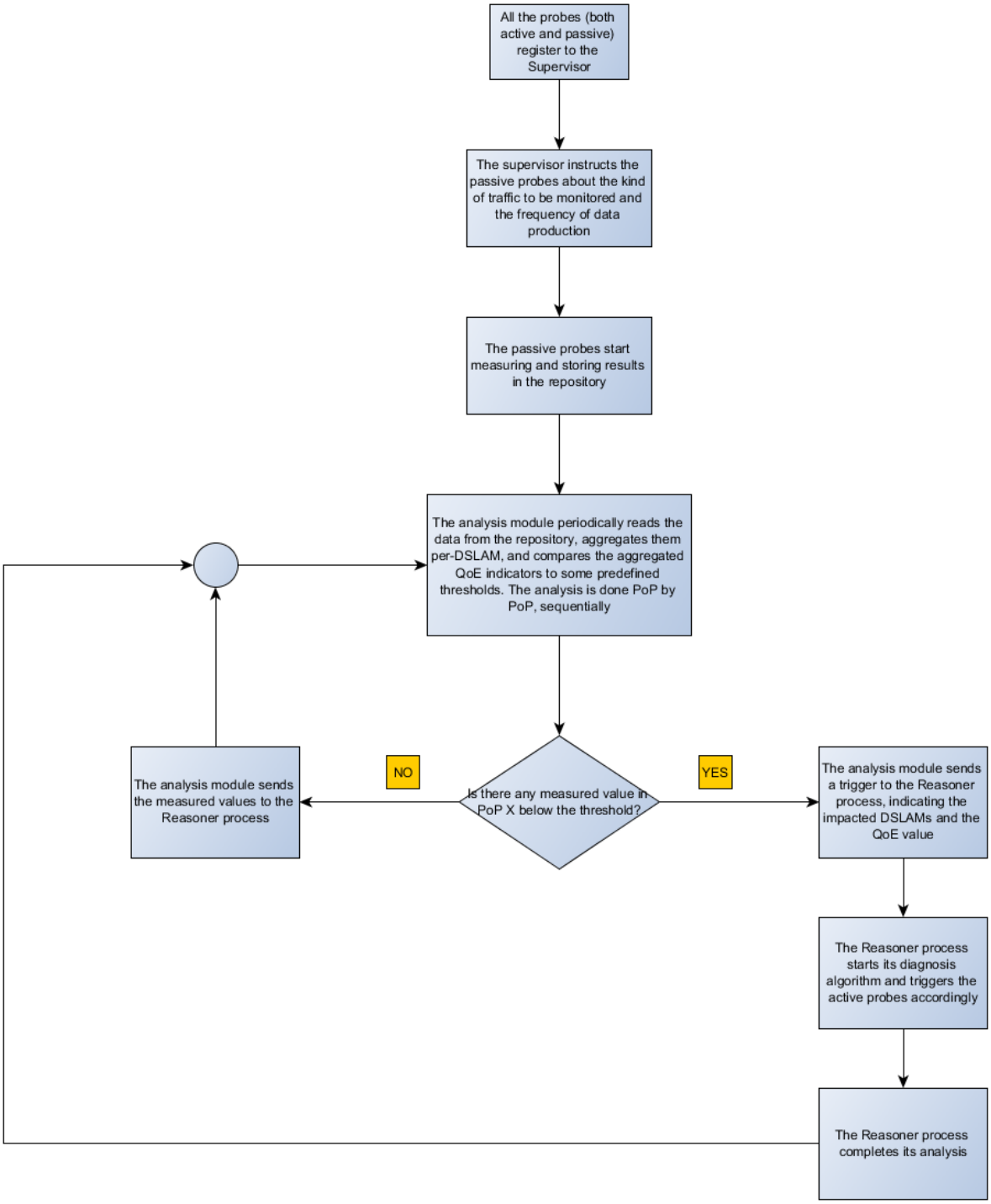


Figure 16: Workflow

3.5.4 First Evaluation Results

Currently, part of the analysis module has been implemented. Specifically, the software that aggregates the QoE parameters measured by the passive probes is available. The aggregation is performed on a per-DSLAM basis, in order to have a sufficient granularity to trigger the Reasoner process. The QoE parameters taken into account are the bandwidth and the Round Trip Time (RTT). The passive probes calculate these quantities for each session and store the values into the Repository. The analysis module takes these values, aggregates them per-DSLAM, and calculates the average. The obtained result is compared to a pre-defined threshold: if the result is below the threshold, the analysis module interprets it as a QoE degradation and triggers the Reasoner process to find the cause of the problem. An example of the average bandwidth per-DSLAM is shown in Fig. 17, where the value is calculated every 10 minutes. After each calculation, the analysis module sends a feedback to the Reasoner process: "OK", if all the DSLAMs are over the threshold, "NOT OK", if one or more DSLAMs are below the threshold. In both cases, the values are sent to the Reasoner.

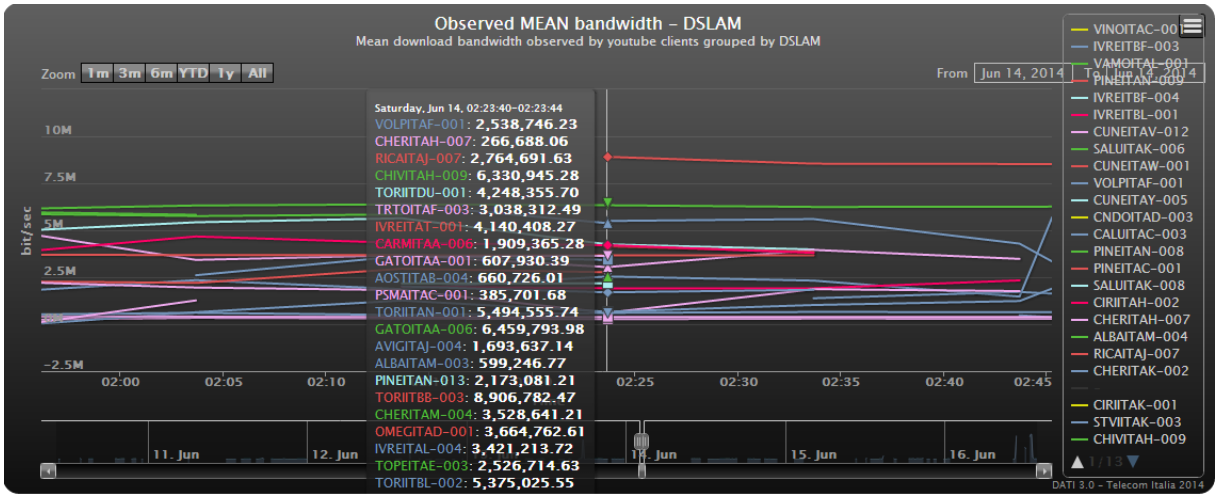


Figure 17: Average bandwidth aggregated per-DSLAM

Fig. 18 shows an example of RTT calculated from the sessions of a specific DSLAM. In this case the values are classified also on the basis of the profile of the access line. This option can be enabled also for the bandwidth calculation.

The RTT is primarily useful to detect rerouting events that move the traffic on longer paths. These events can be within the SP's network (f.i. a link fault on the primary path) or can be due to the Content Provider that starts serving the content from a different site. The bandwidth gives information that partially overlaps with RTT, but it additionally gives hints on packet loss events, that can be due, for example, to network congestion.

3.5.5 Cooperation between different mPlane instancies

The mPlane architecture offers the possibility to envision some forms of cooperation between different mPlane instances. One form of cooperation implies the possibility to ask a different Supervisor for measurement data. For example, an mPlane instance managed by a Service Provider A could run a Reasoner algorithm, as described in Section 3.5.2, to pinpoint the root cause of a QoE

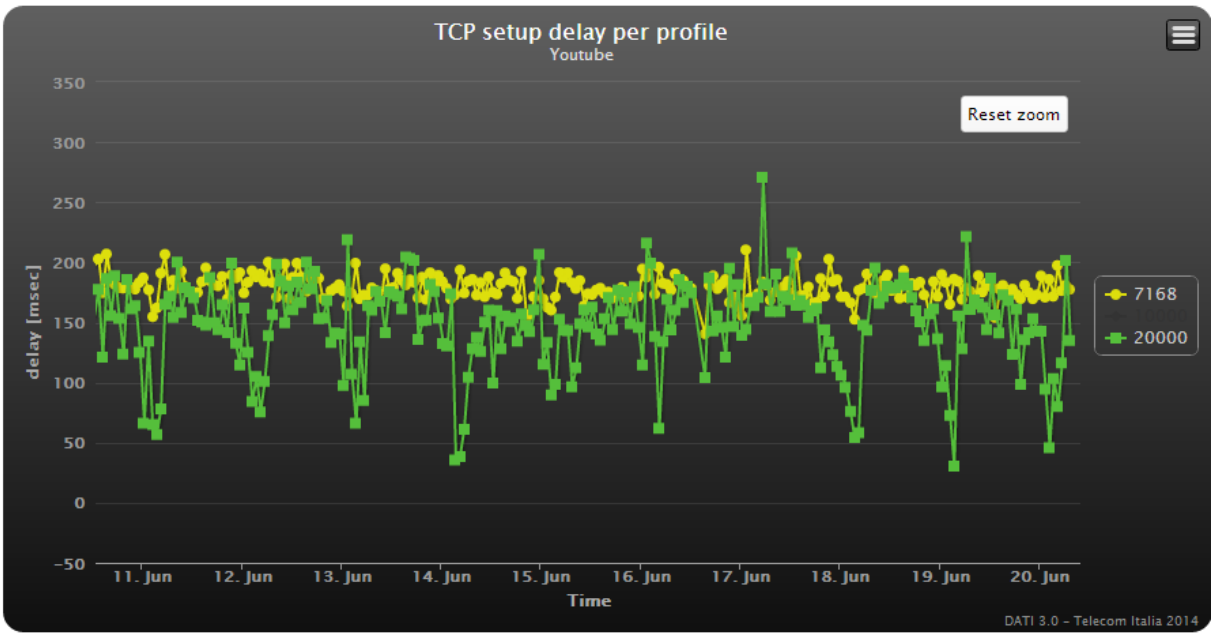


Figure 18: Average RTT aggregated per-DSLAM

degradation for traffic from Content Provider X. In case the algorithm suggests that the problem lies outside the SP's network, the Reasoner (through its Supervisor) could ask the Upstream Provider's Supervisor if there's any issue between the Upstream Provider's network and Content Provider X. From a business point, of view this kind of interworking is essentially based on a customer-provider relationship. On the other hand, SP A could also have in place an mPlane relationship with Content Provider X and directly interact with CP's Supervisor. A third kind of mPlane cooperation is between two Service Providers in competition with each other. From a business perspective this could seem unlikely to happen, but it could be figured out a sort of relationship similar to a private peering agreement for BGP route exchange: a sort of "do ut des" relationship where both players have benefits without revealing business sensitive information. Considering the example described above, the Reasoner managed by SP A could benefit from knowing if other "peer" SPs are also experiencing the same problem, given that it has an "external" cause. In any of the previous cases, the relationship between two Supervisors is exactly the same that exists between a probe and a Supervisor. Supervisor B will advertise one or more capabilities to Supervisor A: those capabilities correspond to measurements that Supervisor A can ask for to Supervisor B. These measurements could be "real" measurements triggered on specific probes or they could result from aggregation of data in the Repository. In any case, Supervisor B can mask all sensitive details to Supervisor A and just expose the result it can deliver to the peer. Considering the QoE degradation problem, the capability exposed by Supervisor B to Supervisor A could be the average RTT to the Content Provider, measured every 5 minutes for the last 30 minutes. This could help (the Reasoner linked to) Supervisor A understand if the problem affects also other SPs or not. Fig. 19 shows how the cooperation with different Supervisors could fit in the flow chart of Fig. 15.

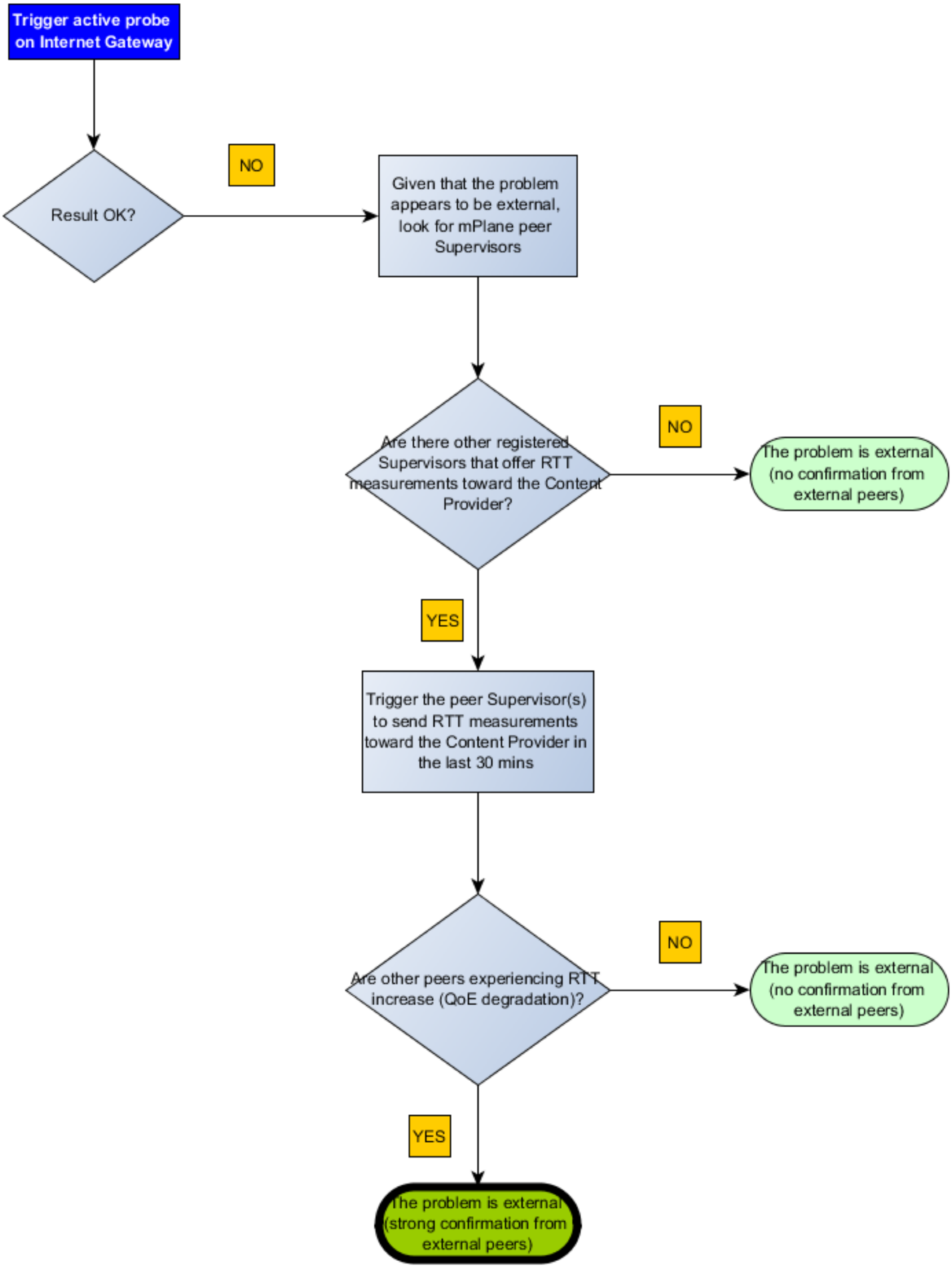


Figure 19: Cooperation between different Supervisors to enhance the troubleshooting algorithm

3.6 Quality of Experience for Web browsing

This use case demonstrates how the mPlane can be used to monitor and find the root causes of end-user quality of experience degradation in Web browsing. To achieve this goal, we browse selected web pages from several vantage points in the network. Each probe sends to the repository aggregated data carrying also its own ``vision'' of the network, and the Reasoner pinpoints the cause of performance degradation on specific targets (i.e., web pages) by exploiting data from different mPlane probes.

3.6.1 The Role of the Reasoner

Probes located in different points of the network browse continuously a set of selected web pages. Collected data are stored at the probe's side, and aggregated results are stored in the Repository. The Reasoner starts its analysis when one of the following events is fired by a Firelog probe:

- page load time above a certain threshold (headless browser probes), or
- annoy button pressed by a human user (instrumented browser probes)

Whenever one of the two triggering events is fired, the Reasoner collects data from the repository, and tries to locate the root cause of a bad performance in web browsing.

Both, the headless browser-based probe and the instrumented browser-based probe send data about:

- browser data (i.e., page load time, http downloading time, number of objects, ...)
- probe status (i.e., cpu / memory usage)
- probe network (i.e., LAN, gateway, ...)
- ping and traceroute to destination

Exploiting data from different Firelog probes, the Reasoner can distinguish between different LANs, different gateways and different paths toward the same destination (see Fig. 20).

As different mPlane probes collect data and send them to the Repository (Fig. 21), the Reasoner can further analyze paths and hops toward a destination, in different times, building an historical view on how a certain web site was served and perceived by users.

3.6.2 Domain-knowledge based Iterative Rules

The Reasoner exploits the information about the browsing session in different parts of the Internet (Fig. 20) toward the same destinations.

Furthermore, it can combine traces from different probes to identify problematic paths and/or hops.

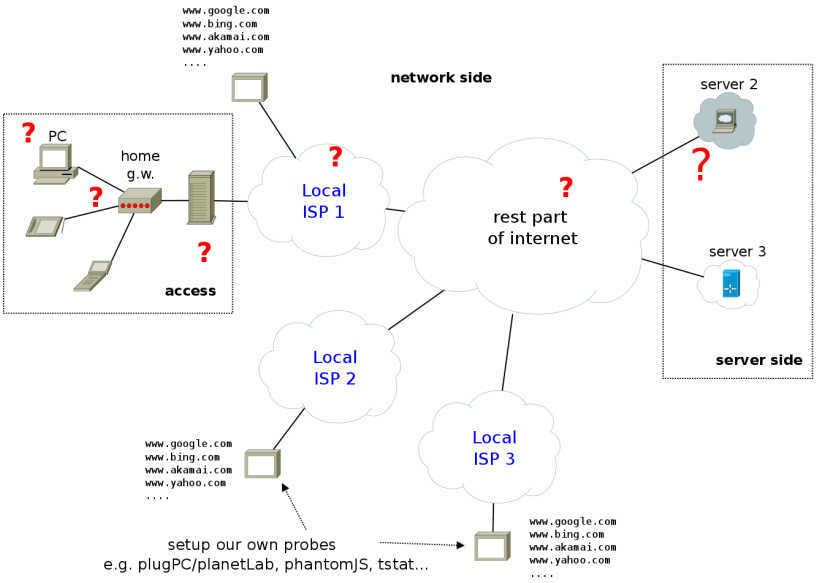


Figure 20: QoE Use case overview: Reasoner point of view

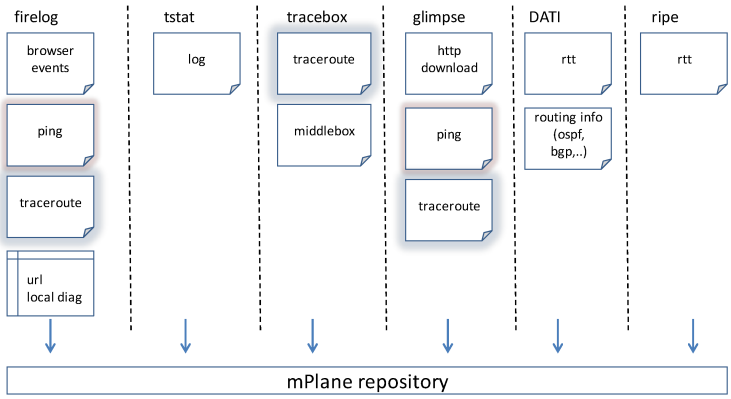


Figure 21: Data ingested by different mPlane probes can be exploited within this Use Case.

3.6.3 Diagnosis/Iterative Analysis Workflow

Fig. 22 depicts the different mPlane components involved in the diagnosis process. There are two types of diagnosis: one, locally performed at the probe side, consists in the probe's point of view on

the executed browsing sessions; the second, more detailed, is run from the Reasoner point of view, and exploits different probes in different parts of the Internet (see Fig. 20).

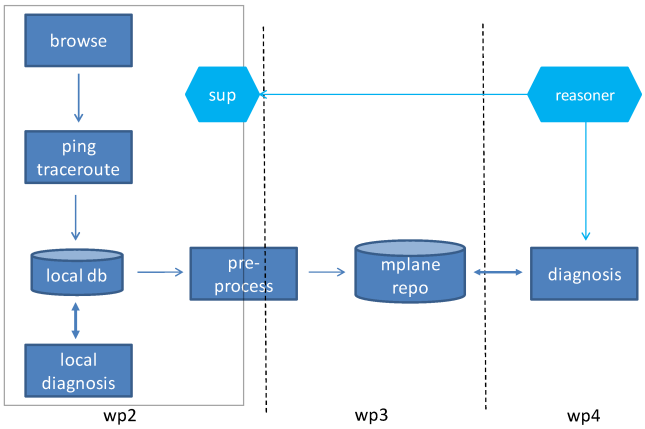


Figure 22: Interdependencies between different mPlane components.

At first, probes located in different points of the Internet continuously browse a selected set of Web pages. For each browsing session, (a) a page is loaded and browser times are computed; (b) a ping to the destination IP addresses is performed; (c) a traceroute to the detination IP addresses is performed; (d) a first diagnosis is locally done on the probe.

Data collected in such a way are stored in the local database of the probe. Aggregated data are transmitted then and stored in the mPlane repository.

If, during a browsing session of a given web page P, the page load time reaches a certain threshold, an event is triggered for the Reasoner to performe a diagnosis.

The Reasoner will then:

- Collect data on P in the mPlane repository;
- Check for other Firelog probes local diagnosis for P (i.e, check if other probes in other parts of the Internet have experienced problems with P);
- Discriminate between different LAN / ISP / paths;
- Select common portions in the path toward the IP addresses serving P;
- Inspect traces toward the destination.

At the end of the process, results are stored in the mPlane repository to collect historical diagnosis data on the selected web page P.

The workflow in this process is given in Fig. 23. If all tests pass, then the Reasoner has to check other mPlane probes' data looking for IP addresses matching the inspected paths, and look for time series in the RTTs in every hop.

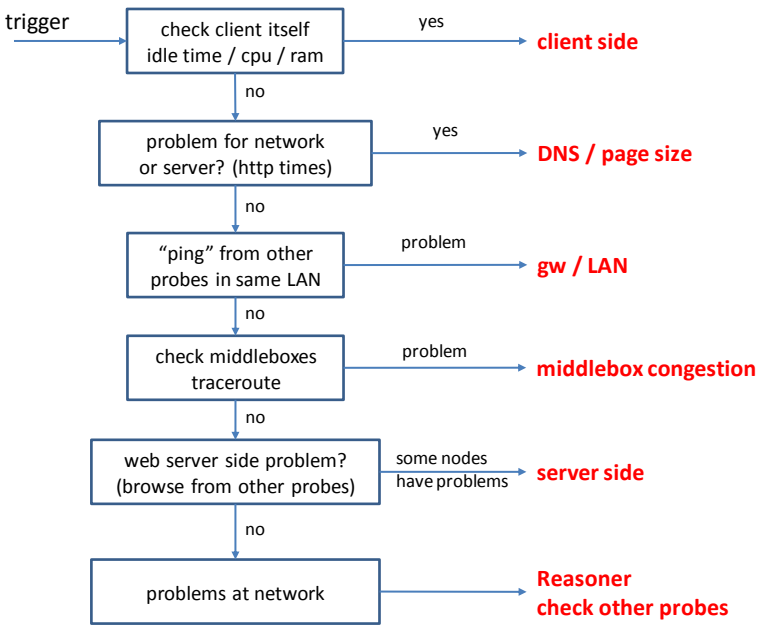


Figure 23: Workflow of the Iterative Analysis

3.6.4 First Evaluation Results

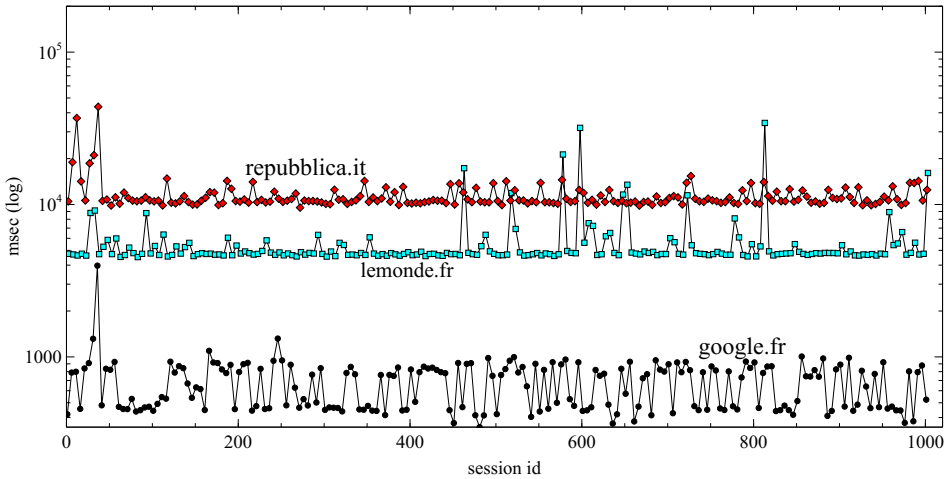
We describe here the preliminary results of the diagnosis algorithm. We collected data from 1005 browsing sessions from a single probe, to underline clearly what can (and what can not) be diagnosed from a single vantage point. In the experiment (started on Monday, June 16 at 17.30 and lasting 24 hours), the analysis algorithm was able to spot the following situation.

#	result	count	%
1	no problem	604	60
2	gw (cusum on gw)	142	14
3	local client: cpu_perc > 50.00	12	1
4	local client: t_idle/t_tot > 0.75	198	20
5	web server too far: t_tcp > 60.00	49	5

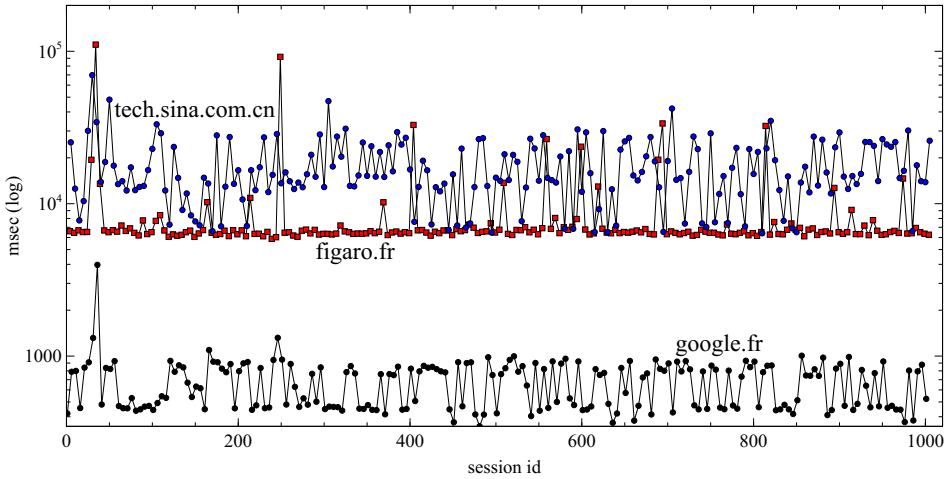
More in details:

- Case n.1: no problem was experienced by the probe.
- Case n.2: the gateway was overloaded. We compute the cumulative sum (cusum) on the RTT to the gateway (i.e., first hop) looking for deviations from the mean.
- Case n.3: client was overloaded. CPU usage was more than 50%.
- Case n.4: browsing application was taking too much time in rendering the page.
- Case n.5: time between SYN packet and SYN-ACK packet was too long.

Time series of page load time metrics computed on selected web pages is presented in Fig. 24, in which we can discriminate between (usually) *good* sites and *bad* sites. Significants deviations from the mean in the page load times (i.e., peaks) are interpreted as poor quality of experience in the browsing session.



(a)



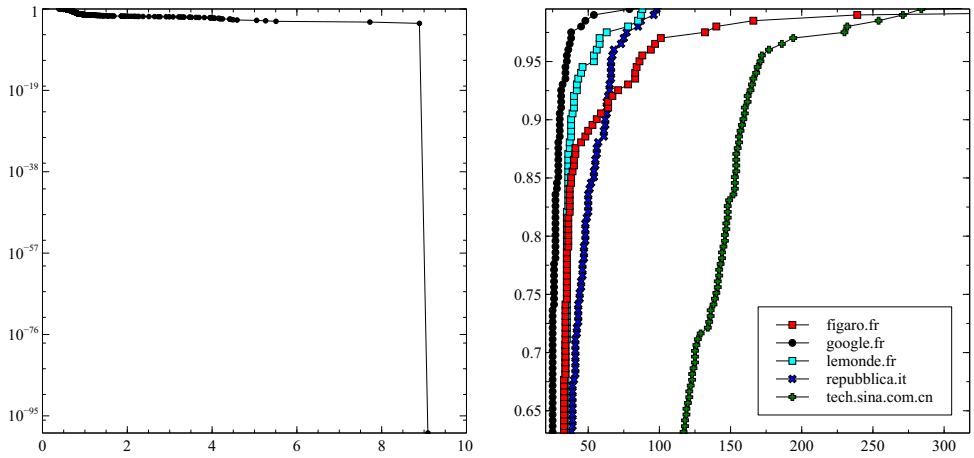
(b)

Figure 24: Page load times of web sites.

A problem on the gateway is detected by inspecting the deviation from the mean in the first hop RTT. In Fig. 25(a) the CDF of such RTT is given. As it can be noted, more than 90% of RTTs are below 1 second.

We can infer a problem in a generic portion of the network by computing the relative distances between subsequent hops in a trace to a given target. For example, consider the trace given in Fig. 26, from which we can infer that the path between intermediate hop 141.136.109.141 and 77.67.72.170 is congested.

We can also distinguish differences in HTTP times, being those the time between the first GET message and the first byte of data received from the web server. In Fig. 25(b) the difference can be noted between *close* Web servers and *far* Web servers.



(a) 1st hop CCDF (b) CDF of http times.

Figure 25: CCDF of the 1st hop RTT and CDF of http times.

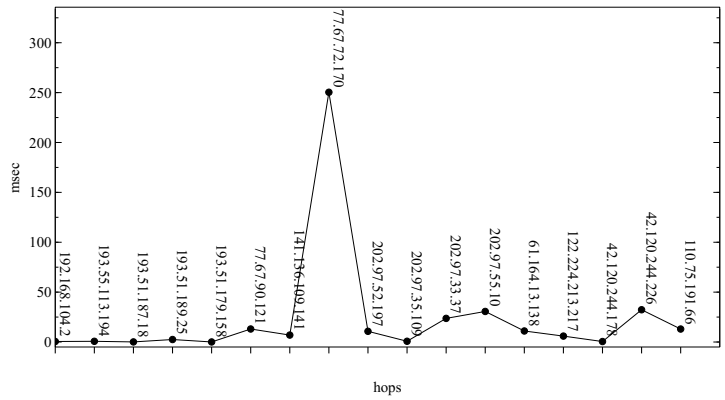


Figure 26: Interhop distance measured as difference between two consecutive RTTs in a trace. The hop between 141.136.109.141 and 77.67.72.170 is congested.

By integrating different measurements from other mPlane probes distributed in different parts of the Internet we can further investigate common paths towards the same destination IP addresses and provide a better understanding on what are the root causes of a poor quality of experience in Web browsing.

3.7 Mobile Network Performance Issue Cause Analysis

The main objective of this scenario is to monitor and discover problems related to poor end-user quality of experience of mobile connectivity. On the one hand, the Reasoner should be able to exploit the information coming from a combination of probes that monitor various parts of the network to identify the key causes of quality degradation in mobile video streaming. On the other hand, another tool implemented by the project solely relies on end-host probe information and targets more general connectivity problems. The latter is briefly described at the very end of this section. The focus of this section is related to mobile video streaming.

3.7.1 The Role of the Reasoner

Triggering an event
In our scenario there are two ways of starting the reasoning process:

- **User Initiated:** The users may self-report the fact that they are experiencing problems with their video playback through a form on their mobile devices.
- **Automatically:** For the video playback scenario, there are certain features that can indicate poor quality of experience: i) long starting time, ii) frequent pauses re-buffering events, iii) in adaptive video quality settings the ability to only display videos of lower resolution. Any of these cases can be easily detected on the mobile device itself. For this specific case a simplified version of the Reasoner, installed on the user's phone, is analyzing these parameters. Furthermore, it is possible to detect these scenarios from a Reasoner deeper in the network (e.g., on the service provider or the ISP).

When any of these triggers is initiated then the troubleshooting iteration begins.

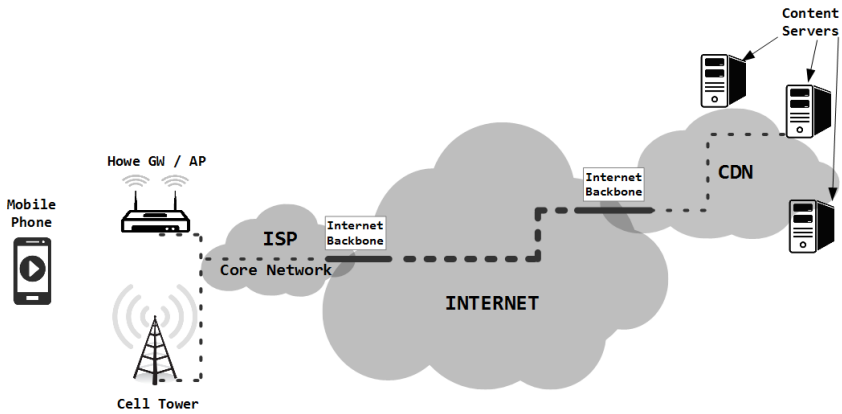


Figure 27: The iterative process of consulting with with reasoners across different entities.

Identifying the root cause of poor connectivity on a mobile device might not a trivial task because there are multiple involved parties along the path of the information (Fig. 27). More specifically, the fault could be found within the user's device, the local connectivity (wiFi/cellular), the connection point (i.e., the access point or cellular tower), the ISP's internal network, peering points with the core network, the CDN or even the servers of the actual service (e.g., Youtube). Therefore, the

Reasoner has to be able to access and analyze the data from multiple probes that lie on different entities.

To this extend two approaches are used i) a simpler centralized approach where the Reasoner has access to all the data and ii) an iterative approach that is more realistic where each entity (user, ISP, CDN, service provider) has its own instances of domain-specific reasoners.

3.7.2 Diagnosis of the Centralized approach

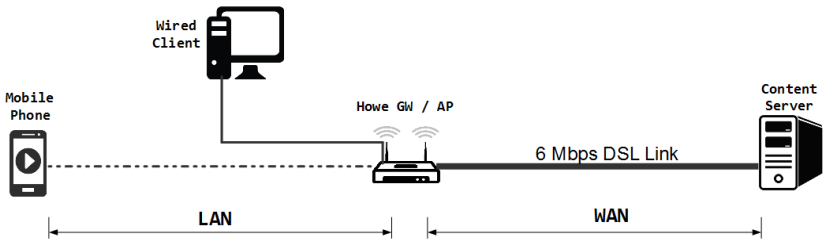


Figure 28: The iterative process of consulting with with reasoners across different entities.

Assuming that the Reasoner has access to the data from all the probes, then the troubleshooting algorithms can be directly applied to this collection of measurements.

Initially the Reasoner has to request access from all the probes that could monitor the flow across the network path and then combine these measurements into a single feature space.

Afterwards, we use machine learning to identify correlations between these nature of the problem and the collected features. For the data processing and analysis we currently use Weka. More specifically, we apply Decision Trees to perform the classification of each instances.

To train the algorithms, we collect a ground-truth dataset following controlled experiments (Fig. 28). These allow us to establish the ground truth not only for problematic sessions but also for the type of problem that occurred. We afterwards use it for training the problem detection algorithm and evaluate it with the dataset from the real-world measurements.

3.7.3 Workflow of the Iterative Analysis

Due to the fact that data is generated within the network of various legal entities (e.g., the user, the ISP providers, core network providers)it is usually the case that these cannot freely exchange the collected information (e.g., YouTube might not share probe data with a user or an ISP). Therefore, a distributed approach in terms of data collection and analysis should be performed.

In our scenario each entity (e.g., user, ISP, CDN) is running an instance of the Reasoner, however, each of these instances are considered as a “black box” for the others.

- Data is collected and owned separately by each involved entity (e.g., the user's device, the mobile ISP, etc.).
- Each entity runs its own instance of a troubleshooting agent that can only access the internal data to identify any possible causes within the organization. Domain knowledge can be used within each entity to identify an issue and trigger the troubleshooting process.

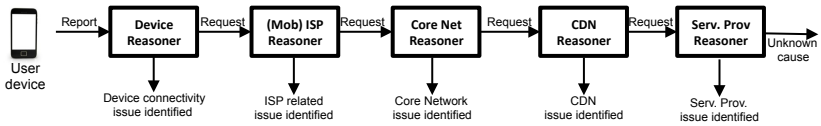


Figure 29: The iterative process of consulting with with reasoners across different entities.

- Finally, the agents across different organizations are using the proposed architecture to collaborate in order to identify the exact cause of a problem. In that process only the abstracted information is revealed between the involved parties. The mPlane reference implementation is used to initiate this collaboration across different reasoners.

- A query to identify an issue is only delegated to another entity only when the local Reasoner indicates that there is no local problem.

For instance, Fig. 29 shows an example of a user reported connectivity problem. In this example:

- The user reports an issue with connectivity or quality of experience (notice that the request to troubleshoot an issue can also be automatically generated).

- The application and device probes use the collected data to identify if the reason is within the device (e.g., poor signal strength, missing codecs, not enough memory, other applications are using the bandwidth). Only information that is collected and owned by the user's device is used. If the reason is identified the issue is considered solved and the user and/or the service provider are notified. If the reason is not identified then the local troubleshooting probe generates a request for further investigation is forwarded to the instance running at the ISP (mobile or fixed) that provides the connectivity. Only the required information such as the timestamp, the objects that caused the issue is shared to help the ISP identify the flow within its own network.

- Similarly, when the probe of the ISP receives a request from a mobile user, it uses its own repository to identify if the problem lies within its own network. Information owned by the ISP such as base station load is used to identify any problems at the specific time/location of the user. Similarly, collected information from the backbone and middle-boxes are also used to identify any causes there. As with the device probe, if no issue is detected within the ISP a request if forwarded further towards the core network that served this request. Notice that the troubleshooting engine of the ISP can also detect a problem (even when initiated by a user's device). As before a request is forwarded to the corresponding probes to further investigate the root cause of the issue.

- In a similar manner, if the issue is not identified, further requests can be further forwarded all the way to the service provider (e.g., web host or video provider). Therefore, in our architecture this sand-boxed iterative process addresses all the aforementioned data sharing issues while providing the ability to track problems across different entities.

At each site, the same machine learning as in the centralized case is used but with two differences: i) only the features from the local probes are used and ii) only the problems that can rely within the entity are considered.

3.7.4 First Evaluation Results

As a starting step, we use the controlled environment of the lab to simulate a range of problematic scenarios in different segments of the data path that potentially cause interruptions in the playback and QoE degradation.

For the controlled experiments, we set-up a simple testbed with a video server, a router/AP (Access Point) and an Android phone. The mobile is connected to the Wireless LAN of the AP and the server is in turn connected via an Ethernet cable to one of router's ports. A wired client in also connected to the router as part of the LAN to act as a traffic generator during the experiments. We use `tc` and `netem` to simulate a DSL link by shaping the downstream of the link between the server and the router to 6Mbps based on the average connection speed reported in Akamai's Q4 2013 state of the Internet Report for Spain and introducing variable delay of 30±10ms with 25% correlation of the next delay value according to our Europe-based ADSL vantage points (Fig. 28).

The video server operates on a Linux distribution with the Apache HTTP server installed. All the videos from the top 100 most viewed list that have been previously downloaded from YouTube to the server in either Standard or High Definition to ensure the diversity of the collection. For the router/AP we used a Netgear WNDR3800 running the OpenWRT operating system. The access point of the device was configured to work on the 5GHz band in order to minimize interference from any surrounding sources. For the mobile client, we used a Samsung Galaxy S II running Android 4.4.2 with `pcap` library installed and root account enabled so that we have sufficient permissions to monitor traffic on the active network interface. The mobile application that we developed is responsible for performing HTTP requests to the server and open the returned video stream using the default Android media player. As soon as the playback finishes the application repeats the process by launching another random video from the list.

In order to create scenarios for the controlled experiments that represent real-world problems, we compile a list of common faults that we will simulate to cause stalling during the video playback. These scenarios can be grouped in three basic categories, networking, device hardware and wireless medium issues.

Shaping and Congestion. In the first category we have the LAN or WAN congestion and LAN or WAN shaping scenarios. These cases correspond to real-world conditions where the resources of the network are limited due to increased traffic, or due to bottlenecks such as slow links or bandwidth caps.

To simulate LAN congestion, we use multiple `iperf` instances to transmit UDP traffic from the wired LAN client to the router, while for the WAN congestion we generate the traffic with the same method but from the server to the router.

Mobile Load. In the second category we examine cases where the high load of the device's hardware does not allow the proper decoding and playback of the video. The problems of this type are more common on handheld devices that come with limited hardware capabilities. The application on the mobile logs these events as error messages from the video decoder. The hardware load simulation is performed with the Linux workload generator tool `stress` that allows to generate CPU, I/O, memory and disk workloads in order to stress-test the host system. For the Galaxy SII, after experimenting with different workload combinations, we observed that one workload for the CPU and one for the I/O operations is enough to create stalls and skipped frames.

Low RSSI The last category deals with the simulation of two faults common in 802.11 networks. In the first scenario (low RSSI) we simulate poor signal reception by placing the phone far from the AP

and blocking the line-of-sight with physical objects. As a result, there is degradation in the wireless link's SNR and the available data rate. More specifically, during the experiments we noticed high frequency re-buffering with RSSI values of -89dB and lower and link speeds less or equal to 2Mbps.

WiFi Interference. The second scenario titled WiFi interference, involves creating interference on the wireless channel from external sources. In real use cases, such interference can be caused by adjacent devices transmitting or receiving on the same frequency range. For our experiments we create interference by generating large traffic workloads on a second WLAN where the AP operates on the same channel as the AP we use for our measurements.

Background Variations. To generate the dataset for the controlled experiments, for each scenario we perform multiple iterations with random videos and gradually increment the intensity of the problem until we observe frequent re-buffering events on the smartphone's system log. At the same time, attempting to create more realistic network conditions, we introduce variance to the system by adding synthetic traffic workloads of different patterns and at random intervals. More specifically, we again use `iperf` to create combinations of long or short UDP workloads with high or low bandwidth. We ensure that the added traffic is significant ranging from 32Kbps to 1Mbps, but does not affect the outcome of the experiments by causing stalls in the playback.

3.7.4.1 Datasets

All the collected metrics that correspond to a single video session are aggregated to one instance in the dataset. Each instance in the controlled experiments dataset is comprised of 343 metrics out of which there are 113 network metrics for each of the three vantage points, the total number of re-buffering events and from the hardware measurements of the mobile we get the maximum observed CPU utilization, the minimum amount of free memory and the minimum value of the RSSI.

Before performing the analysis, the instances in the data need to be labelled appropriately in order to be identified and evaluated by the classifier. Specifically, we remove the re-buffering events from the instances in the controlled experiments data and label non problematic instances as `good'. The problematic sessions are labelled as either `lan shaped', `lan congested', `wan shaped', `wan congested', `low rssi', `wifi interference' or `mobile load' according to the simulated scenario they correspond to. In the real-world dataset however we have no knowledge of the type of problem that caused re-buffering at the player side so we are only able to mark the problematic instances as `bad'.

Detecting Problems: Firstly, we want to examine weather it is possible to identify problematic video instances through each one of the vantage points or the combination of them. We prepare the data by merging all the labels from problematic instances to a single label `bad', while preserving all good labels. We consecutively evaluate for every vantage point separately and finally with all the points combined. The overall accuracy for the mobile and router is 78.8%, for the server 74.4% and for the combination of all 80.3%. Although the server vantage point is performing worse than the other two when used separately, there is significant improvement when we take measurements from all probes combined.

In Table 1 we present the performance of the algorithm per vantage point in terms of Precision (P) and Recall (R). We observe that the worse performance of the server derives from the lower accuracy when identifying bad instances. The intuition behind this observation is that most of the problems in our dataset occur far from the server where there is not enough information to correctly identify instances as bad.

	Mobile		Router		Server		Combined	
	P	R	P	R	P	R	P	R
good	0.84	0.82	0.81	0.87	0.79	0.81	0.83	0.87
bad	0.7	0.73	0.73	0.65	0.66	0.62	0.75	0.72
W. Avg.	0.79	0.79	0.78	0.79	0.74	0.74	0.8	0.8

Table 6: Accuracies for problem detection in controlled experiments.

Detecting the Location of Problems: In the next step we aim in verifying the algorithm's accuracy when identifying in which part of the data path the problem has occurred. For this purpose we create three new labels `wan', `lan' and `mobile' based on the locality of the problem. In label `wan' we merge wan congestion and wan shaping problems, `lan' contains instances from lan congestion, lan shaping, wifi interference and low rssi scenarios and finally in the `mobile' we place the problematic instances that correspond to mobile load.

The percentage of the correctly classified instances drops to 75.95% in this evaluation case. As expected the accuracy for identifying good instances remains approximately the same as for good/bad classification. However in the related accuracies in Table 7, we see that mobile device problems are detected with higher accuracy. This is attributed to the stronger correlation of the hardware metrics with the particular problem.

	Precision	Recall
good	0.84	0.87
lan	0.51	0.51
wan	0.6	0.43
mobile	0.9	0.82
Weighted Avg.	0.75	0.76

Table 7: Accuracies for localization detection in controlled experiments.

Detecting the Exact Problem: In the following part of the analysis of the controlled experiments, we train and evaluate the algorithm using all the labels of problematic scenarios that are available in our dataset. In this way we assess the accuracy with which the classifier can detect the root cause behind the problem experienced by the user.

From the output of the classifier we get 73.7% correctly identified instances while different labels are classified with different accuracies as seen in table 3.7.4.1. In more detail, we observe low performance for lan congestion and 802.11 related problems but much higher for wan shaping and mobile load.

Our next step involves a per-vantage-point evaluation to examine which vantage point is performing better when identifying each problem type. For this evaluation it is necessary to separate the measurements from each point to a different dataset. After the separation, we train and evaluate the classifier for each of the three new datasets.

From the results of each classification we compile Table 9 where we compare the precision and recall measures of each vantage point separately and with their combination. From the table we can observe that the vantage point on the mobile is able to detect with higher accuracy problems in the LAN segment and issues of the wireless medium. The router performs well when detecting wan congestion and lan shaping while the only problem the server can identify with better accuracy than the other vantage points is wan shaping. In terms of overall accuracy, the mobile is better than

	Precision	Recall
good	0.84	0.83
lan congested	0.24	0.36
lan shaped	0.67	0.6
wan congested	0.71	0.62
wan shaped	1	0.7
low rssi	0.46	0.6
wifi interference	0.44	0.36
mobile load	0.9	0.82
Weighted Avg.	0.76	0.73

Table 8: Accuracies for root-cause detection in controlled experiments.

the router vantage point which in turn is better than the server, with respective accuracies 73.77%, 69.94% and 68.85%.

The improved accuracy of the mobile, is a strong motivation for instrumenting users' devices. With a single probe collecting measurements from the mobile, the user is able to verify if the problem occurs locally or in a remote part of the network. In the case of a local problem, the algorithm can help the user troubleshoot by providing information about its root cause. If the issue occurs remotely, the user is able to report the problem to the respective network administrator.

Another interesting finding from the results in this section is that any vantage point in our system can tell with good accuracy if a video did not suffer any problems. Based on this insight, middle entities such as service providers can use TCP-driven detectors to detect problems without having to instrument the client or the server.

Finally, we conclude that the usage of a combination of vantage points in a distributed manner, helps to increase the accuracy of the system. Strategically placing more probes on devices along the data path such as edge routers, will not only improve the detection of problems but add knowledge to the system about the location and the nature of the problem.

	MOBILE		ROUTER		SERVER		COMBINED	
	P	R	P	R	P	R	P	R
good	0.83	0.88	0.81	0.85	0.79	0.85	**0.84**	0.83
lan congested	**0.45**	**0.36**	0.31	0.29	0.29	0.29	0.24	0.36
lan shaped	**0.67**	**0.57**	0.4	**0.57**	0.5	0.29	0.67	0.6
wan congested	0.4	0.5	**0.83**	**0.62**	**0.8**	0.5	0.71	0.62
wan shaped	0.5	0.33	0.57	**0.67**	**0.67**	**1**	**0.7**	0.33
mobile load	**1**	**0.73**	0.5	0.27	0.75	0.27	**0.9**	0.82
wifi interference	**0.5**	**0.64**	0.45	0.45	0.43	0.54	0.44	0.36
low rssi	**1**	**1**	**0.4**	0.29	0.25	0.29	0.46	0.6
Weighted Avg.	0.72	0.74	0.69	0.7	0.69	0.76	**0.73**	**0.74**

Table 9: Accuracy comparison for all vantage points and labels

3.7.5 End-Host-based Performance Problem Troubleshooting

As already mentioned at the very beginning of this section, another tool (GLIMPSE) has been developed to troubleshoot more general network performance problems. There is a lot of overlap

with the general descriptions of the problem and solution space so far (e.g. user-initiated and automated/supervisor-initiated measurements both potential triggers for iterative analysis). In this section we highlight a few differences to the reasoning approach described above.

The Reasoner is responsible for the supervisor-initiated measurements. These are scheduled for two different reasons. Either they are part of a continuous measurement campaign (e.g. a campaign to measure the multi-path topology of the Internet using Paris Traceroute or a measurement schedule to monitor the average delay to Google or Akamai caches - ground truth measurments in general) or they are triggered based on observed anomalies. The latter initiates an iterative analysis of the anomaly. The detected anomaly could be due to a user-initiated or supervisor-initiated measurement. An anomaly as of now is a reachability problem or a significant deviation of an expected behavior or metric (determined by a long running measurement campaign). E.g. when a user initiates a test of his or her Internet connectivity because of a perceived performance problem, the GLIMPSE probe will start a number of pre-defined test procedures and will send the results back to the repository. The Reasoner will inspect the result and conclude whether the performance problem is local to the user (based e.g. on SNR values, Wifi channel overlap and other factors) or whether it is somewhere deeper within the network. If it is the latter case, the Reasoner will trigger additional measurements (probes in the same geography, same service provider etc.) to narrow down the location of the problem. The Reasoner continues this iterative measurement until enough data is collected to eventually conclude on a likely problem location.

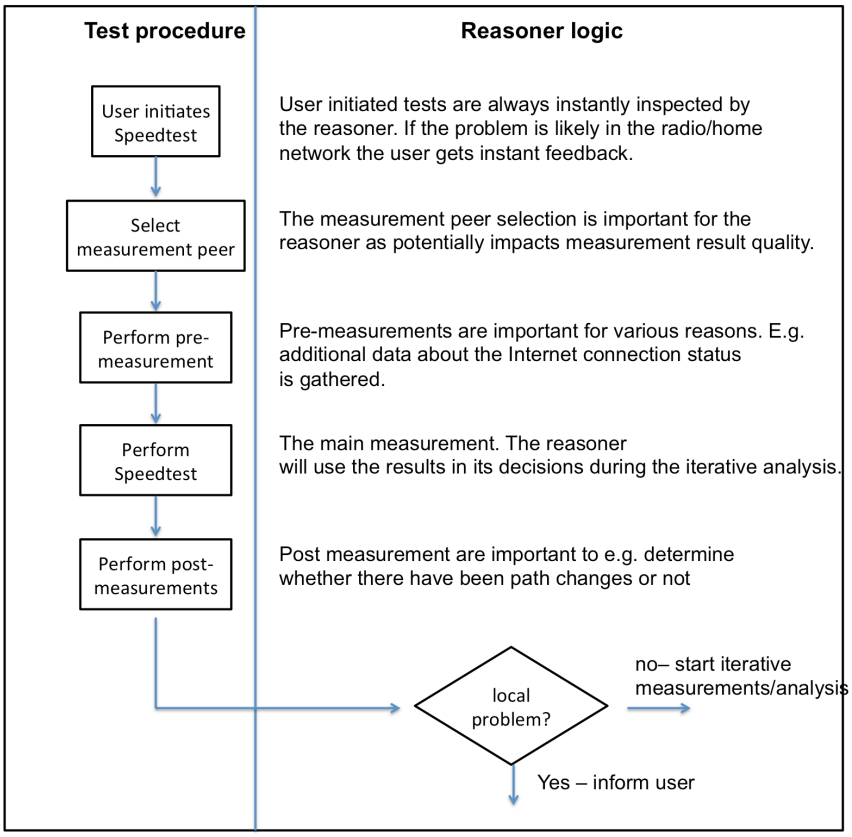


Figure 30: GLIMPSE iterative reasoning process example

The GLIMPSE probes provide two sets of information. They provide probe-local information such as Wifi quality and if available e.g. information from the home gateway via UPnP and they also

provide their measurement results (ping, traceroute, speedtests and others). This information is needed by the Reasoner as to determine whether the perceived problem is located in the mobile/home network or beyond. Only if the problem cause is likely outside the radio/home network, the Reasoner will determine the next measurements to be performed by other GLIMPSE probes. These additional measurements are currently pre-defined but will – over time and given experience with the system – be changed to dynamically generated test instructions. In other words, the iterative reasoning is not a chain of reasoners that are triggered one by one until a root cause can be determined but a set of iteratively triggered measurements all evaluated at a central Reasoner instance.

An example is illustrated in 30, where a user-initiated speedtest serves as a potential trigger for the Reasoner to initiate additional measurements.

3.8 Anomaly Detection and Root Cause Analysis in Large-scale Networks

This use case targets the continuous monitoring of large-scale network traffic, aiming at detecting and diagnosis anomalies potentially impacting a large number of users. The use case particularly focuses on the most popular web-based services (e.g., YouTube, Facebook, Google Services. etc.), delivered by complex network infrastructures maintained by omnipresent Over The Top (OTT) content providers and major Content Delivery Networks (CDNs) such as Google, Akamai, Limelight, SofLayer, etc.. Detecting and diagnosing anomalies in such scenarios is extremely complex, due to the number of involved components or players in the end-to-end traffic delivery: the Content Provider, the CDN provider, the intermediate Autonomous Systems (ASes) of the transit Internet Service Providers (ISPs), the access ISP, and the terminals of the end-users. This high complexity motivates the usage of mPlane to improve the visibility on the traffic and on all the intermediate components. And more specifically, the diagnosis of the detected anomalies requires the coordinated guidance of the mPlane Reasoner, which shall decide the specific measurements and deeper analysis to perform, once an anomalous is detected.

As an example of the application of the Reasoner in this use case, we present two different application scenarios, aiming at detecting and diagnosing different relevant anomalies. The first scenario corresponds to the detection and diagnosis of major anomalies in the delivery of YouTube videos impacting the Quality of Experience (QoE) of a large number of users. This scenario is used next as a basis for describing the complete work-flow of the mPlane Reasoner. The second scenario consists on the detection and diagnosis of load balancing policies in the Akamai CDN, specially targeting the selection of preferred caches for content distribution and the detection of deviations from these preferred caches. This second scenario was presented in [46].

3.8.1 The Role of the Reasoner

The role of the Reasoner in the monitoring, detection, and diagnosis of QoE-based anomalies in YouTube is 3-fold: firstly, the Reasoner has to coordinate the analysis of the measurements and results provided by the different analysis modules which participate of the specific use case. These algorithms includes anomaly detection, tracking of inter-domain path changes, QoE-based measurements at end devices (if available), or QoE-based YouTube monitoring at the flow level, identification of path-congestion from single vantage-point measurements, and the instantiation of active measurements through geo-distributed measurement platforms such as RIPE Atlas [40]. Secondly, the Reasoner is in-charged of guiding the drilling down of any detected anomaly to find out it root causes, once an anomaly-detected triggering event is raised by the anomaly detection algorithm. For doing so, the Reasoner follows the steps dictated by its diagnosis graph. Finally, in case no root causes are correctly identified, the Reasoner triggers additional algorithms to discover new diagnosis rules to enrich the set of diagnosis rules.

Fig. 31 depicts the configuration of the different components involved in the detection and diagnosis of large-scale anomalies in the delivery of YouTube videos impacting the QoE of a large number of users. Traffic is passively monitored at the access ISP, in one or multiple Points of Presence (PoPs) aggregating a large number of customers, using one or multiple mPlane-Tstat passive monitoring probe(s). Traffic is monitored at the flow-level, generating a large set of flow-statistics for all the downlink and uplink traffic. Using Tstat flow filtering and traffic classification capabilities, only

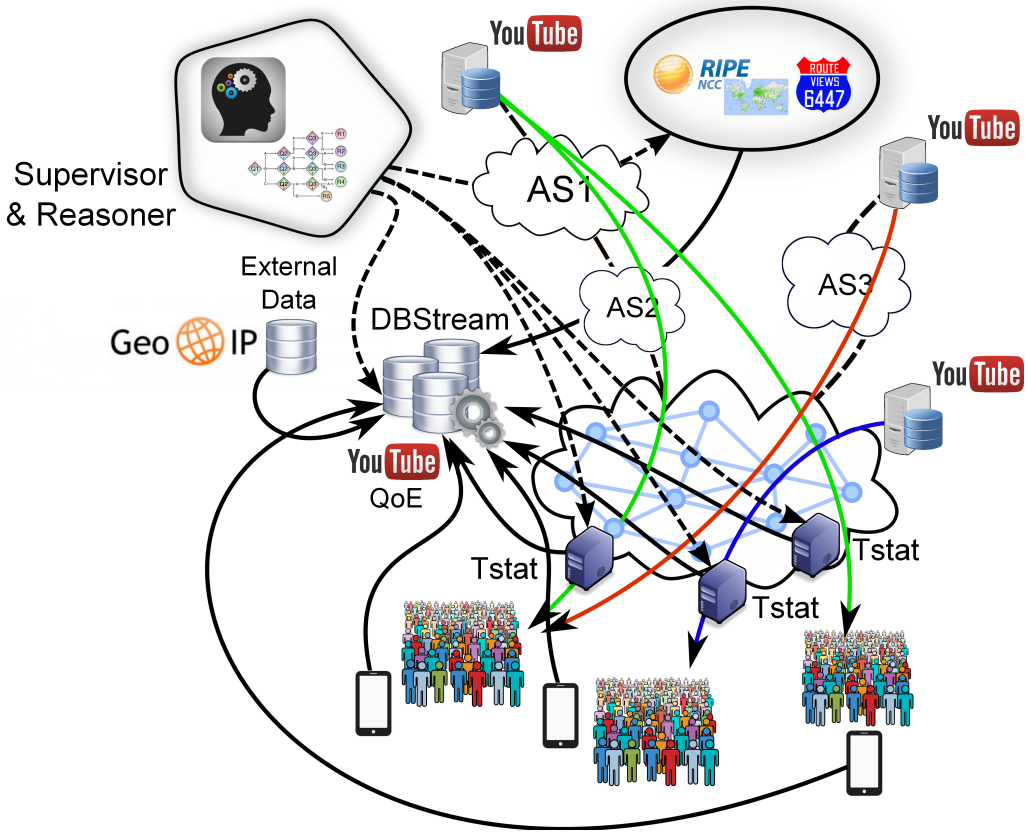


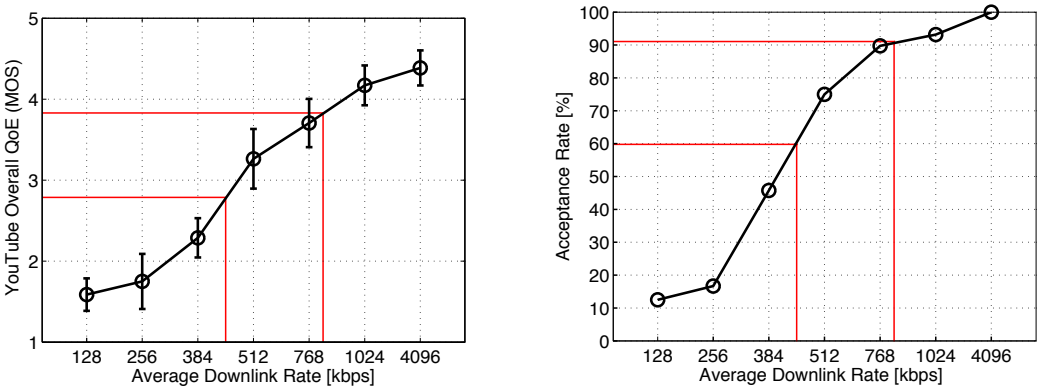
Figure 31: Detection and diagnosis of QoE-based large-scale anomalies in YouTube. The term large-scale reflects those anomalies which impact a large number of customers. QoE-based measurements at the end-devices are potentially used for diagnosing device-problems. However, as we target large-scale anomalies, end-device issues are generally discarded from the beginning.

flows related to YouTube videos are retained for further analysis. Some of these per-flow statistics include: flow size, flow duration, average download throughput, video bitrate, server IP, RTT, etc..

Flows captured at the passive probes are periodically exported to one of the mPlane's repositories, DBStream, which is in-charged of performing the large scale analysis of the combined measurements, which is described next. Tstat flow measurements are combined with two other types of measurements: (i) external data coming from geo-localization services such as MaxMind[5] and IP-address analysis services such as Team Cymru Community Services[6], and (ii) inter-AS path performance measurements and routing, generated through the combined usage of the geo-distributed active measurements framework provided by RIPE Atlas [40] and BGP measurements coming from RouteViews[7].

One important note on this use case is that we perform the YouTube monitoring from passive measurements at the access ISP and not at the end-terminals, which might highly reduce the visibility on some of the features of the video flows in case of HTTPS usage. Some other mPlane use cases show how similar monitoring can be performed directly from end-device measurements, avoiding

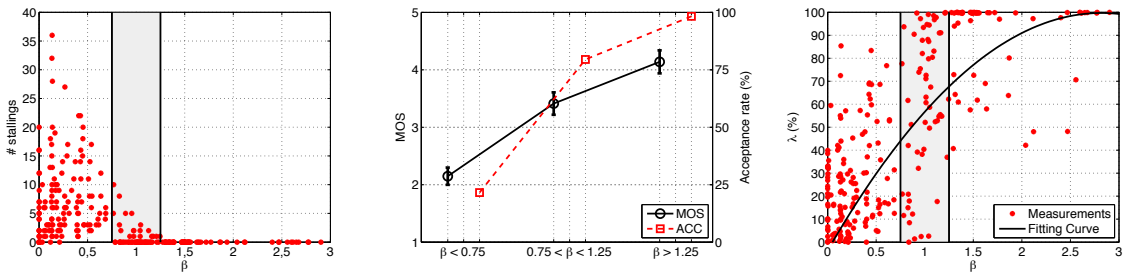
[5] https://www.maxmind.com
[6] https://www.team-cymru.org/
[7] http://routeviews.org/



(a) YouTube overall QoE vs. downlink rate.

(b) YouTube acceptability vs. downlink rate.

Figure 32: YouTube overall QoE and acceptability in terms of average downlink rate. The curves correspond to a best-case scenario, in which only 360p videos were considered. In a more general case with higher resolution videos (e.g., 1080p HD), the download rate has an even stronger effect on the user experience. The Figs. are taken from the study performed at [3].



(a) # stallings vs. β.

(b) MOS and acceptance vs. β.

(c) User engagement (λ) vs. β.

Figure 33: β = ADT/VBR as a metric reflecting user experience and engagement. Users have a much better experience and watch videos for longer time when $\beta > 1.25$, corresponding to ADT = 750 kbps in 360p videos.

traffic encryption and obfuscation issues at the flow transport. This particular use case considers the additional usage of end-device measurements for diagnosing device-problems. However, as we target large-scale anomalies impacting a large number of users, end-device issues are generally discarded from the beginning. Indeed, issues at the end device would rarely cause service disruptions and performance degradation at the large-scale, except from limited situations linked to corrupted software updates and or compromised terminals (i.e., terminals belonging to a botnet).

The main triggering event of this use case is the detection of a QoE-based degradation event impacting a large number of users. In the case of YouTube, the download throughput is the main network KPI that influences the experience of the user. Even so, our studies [4, 3] have shown that the main impairment affecting the QoE of the end-users watching HTTP video-streaming videos are playback stallings, i.e., the events when the player stops the playback. One or two stalling events are enough to heavily impact the experience of the end user. Given that the Tstat flow measurements report the average flow download throughput as one of the monitoring KPIs, we rely on our previous results to better understand how download throughput relates to QoE and stallings in YouTube.

Fig. 32 reports the overall QoE and the acceptance rate as declared by users watching YouTube

videos during a field trial test conducted and reported in [3], both as a function of the average download rate. During this one-month long field trial test, about 40 users regularly reported their experience on surfing their preferred YouTube videos under changing network conditions, artificially modified through traffic shaping at the core of the network. Fig. 32(a) shows the overall QoE as a function of the average download rate, using a 5-points MOS scale, where 1 corresponds to very bad QoE and 5 to optimal. The Fig. clearly shows that the overall QoE drops from a MOS score close to 4 at 800 kbps to a MOS score below 3 at 470 kbps. A MOS score of 4 corresponds to good QoE, whereas a MOS score below 3 already represents poor quality. The same happens with the service acceptance rate, as reported in Fig. 32(b). In the first results of the use case presented in section 3.8.5, we shall consider the thresholds $T_{h_1} = 400$ kbps and $T_{h_2} = 800$ kbps as the throughput values splitting by bad, fair, and good QoE. Both curves correspond to a best-case scenario, in which only 360p videos were watched by the users. As we see next, both 360p videos and videos with higher resolutions are present in the dataset analyzed with mPlane as example, thus QoE degradations are potentially worse than those reported.

To improve the QoE monitoring capabilities of this specific use case, we introduce a simple yet effective QoE-based KPI to monitor the QoE of YouTube videos from network measurements. In [4] we have already devised an approach to estimate stallings in YouTube from passive measurements at the core network, but the used techniques can not be applied when YouTube flows are carried over HTTPS. Therefore, using the same measurements of the field trial, we introduce a new approach. Intuitively, when the average download throughput (ADT) is lower than the corresponding video bit rate (VBR), the player buffer becomes gradually empty, ultimately leading to the stalling of the playback. We define β = ADT/VBR as a metric reflecting QoE. Fig. 33 reports (a) the measured number of stallings events and (b) the QoE user feedbacks as a function of β. In particular, no stallings are observed for $\beta > 1.25$, and user experience is rather optimal (MOS $>$ 4). As a direct application of these results, if we consider standard 360p YouTube videos, which have an average VBR = 600 kbps [13], an ADT = 750 kbps would result in a rather high user QoE, which is the value recommended by video providers in case of 360p videos [24]. Fig. 33(c) additionally shows how the fraction λ = VPT/VD (video played time and duration) of the video time actually viewed by the end users actually increases when β increases, specially above the β = 1.25 threshold.

3.8.2 Domain-knowledge based Iterative Rules

As explained in chapter 2, the Reasoner guides the iterative analysis through a set of diagnosing rules to verify the occurrence (or not) of specific signatures explaining the detected issues or symptoms. These rules are initially defined by an expert operator based on his domain knowledge and operational experience. Given a specific symptom event to diagnose -- in this specific use case, an important QoE-degradation impacting a large number of users watching YouTube -- each of the rules checks for a predefined signature characterizing the diagnostic events which might explain the symptom, i.e., the root causes.

In order to define the set of knowledge based rules to diagnose a problem, the first step is to identify which are the possible root causes of such problems, and where could the origins be located. The large number of possible root causes coupled with the generally much lower number of vantage points providing information about the symptoms makes the enumeration of the root causes and their location a complex task. The approach we take is a coarse one, in which we drill down the detected anomaly to find out the main part of the end-to-end service delivery responsible for it (e.g., device, access ISP, Internet, CDN, content provider), rather than the specific network element

(e.g., interconnection router, link failure, routing table, etc.). In the specific case of Internet-scale services like YouTube, which are hosted and delivered by highly distributed and omnipresent CDNs, the origins of the problems could be potentially located at:

- **the end terminals**: potential issues in the end-terminal are multiple, from software to hardware issues, as well as connectivity and signal strength among others. However, as we said before, this use case considers QoE impacts in a large number of users, and thus individual buggy terminal events are out of the scope of the diagnosis analysis. Only problems simultaneously affecting a large number of terminals are potentially considered, for example, issues related to software updates affecting a whole category of devices (i.e., iOS smartphones, Windows 8 OS, etc.).
- **the home network**: similar to previous observations for end terminal issues, the home network could be a potential issue only in case of problems affecting for example a whole category of home gateway devices. However, in this specific case, firmware updates are much less frequent than OS and software updates, and therefore we exclude the home network from the analysis.
- **the access network**: diagnosing issues at the access network heavily depends on the type of access network considered (cellular, WiFi, FTTH, ADSLx). Download throughput problems at the access can be caused by multiple issues, from congestion events to equipment outages and misconfigurations.
- **the core network of the ISP**: problems at the ISP providing the Internet access to the users are generally the most common ones. These are various, including intra-AS routing, router outages and equipment failures, misconfigurations, etc.. The usage of virtualization and software-defined technologies (both the the access and core networks) adds additional sources of potential performance issues.
- **the Internet**: depending on the location of the YouTube content and on the cache selection policies used by Google to answer users' requests, the YouTube flows might have to traverse multiple ASes from the YouTube servers till reaching the access ISP. YouTube would normally assign user requests to the closest servers (in terms of latency), which could even be located inside the access ISP -- Google also follows the ``inside the ISP'' approach of Akamai, through the Google Global Cache framework[8] -- or at the edge, as having direct peering links between the operator and Google is normal. Still, due to it's load balancing policies, YouTube might assign users to other servers farther located, resulting in multi-AS paths from servers to customers. As a consequence, problems related to inter-AS routing, congestion at intermediate ASes, and multi-AS paths performance degradation are potential root causes for YouTube QoE degradation.
- **the CDN and the servers**: the final part of the end-to-end service diagnosis corresponds to the servers hosting and providing the YouTube videos. Software or hardware problems of the hosting servers, overloading situations of wrongly dimensioned servers, internal problems of the hosting datacenter, etc. are possible root causes to additionally diagnose.

Once we have enumerated the list of elements to diagnose, we can define a set of rules which shall be iteratively verified to detect the occurrence of events revealing the aforementioned problems.

[8] `https://peering.google.com/about/ggc.html`

Recall that the Reasoner works on top of diagnosis events tracked and recorded by different analysis algorithms, instead of directly operating on top of the raw measurements collected by the monitoring probes. Such events are either continuously generated by the running analysis algorithms (e.g., anomaly detection, path-change detection, path-congestion detection, QoE-based issues reporting at end-devices, etc.), or generated on demand, through the analysis of specific measurements (e.g., check if some specific YouTube server is reachable from gro-distributed probes at this time). The diagnosis rules consists in verifying the occurrence (or not) of these events among the registered ones by the different components of the mPlane.

The triggering event is generated by the anomaly detection analysis module, and consists of the detection of a significant drop in the QoE of users watching YouTube videos, using the aforementioned β parameter as monitored KPI. As described in D4.1, the anomaly detection module works by analyzing the complete empirical distribution of the monitored KPIs. Therefore, one important step before triggering the diagnosis process is to check the statistical consistency of the detected anomaly. For example, important deviations in the empirical distribution of the β KPI can be caused by a sudden and important drop/increase in the number of YouTube flows, or by an abrupt modification in the number of users watching YouTube. For doing so, the anomaly detection algorithm firstly checks for the presence of events related to major statistical variations in the number of YouTube flows and the number of users watching YouTube. The algorithm additionally defines an hysteresis-based approach for triggering the diagnosis, in which a number of consecutive anomaly alarms have to be flagged before launching the drilling down process. The following non-exhaustive list enumerates some of the domain-knowledge based rules for diagnosing this QoE-drop event through mPlane:

- **terminals and home networks:**

1. Device-related? $\rightarrow$ for all the involved user devices corresponding to the affected flows, check for the occurrence of end-device issues.
2. Device-OS related? $\rightarrow$ for all the involved user devices corresponding to the affected flows, check the heavy hitters of OS type, and the entropy of the OS class.
3. Set-top box related? $\rightarrow$ for all the involved boxes corresponding to the affected flows, check the heavy hitters of box-type, and the entropy of the OS class.

- **access network:**

1. Access-overloading? $\rightarrow$ check the occurrence of access-overloading events during the last available days, for the corresponding access networks or slots (e.g., users in the same mobile cell, or in the same aggregation network, or attached to the same DSLAM, etc.). Compare to similar events for other users accessing the same YouTube servers through a different access network. Overloading events tend to be periodic and not constant.
2. Access-configuration related? $\rightarrow$ check the occurrence of re-configuration events related to the corresponding access networks or slots.
3. Equipment failure related? $\rightarrow$ check the occurrence of outage events reported by the KPIs monitored by the ISP at the corresponding access networks.

- **core network:**

1. Intra-AS routing issues? → check for routing re-configuration events tracked by the ISP monitoring system occurring at the times of the detected throughput anomalies. Note that the ISP might have its own mPlane instance running, and therefore this and similar ISP-related event queries can be done through inter-supervisors communication.
2. Congestion-related issues? → check for co-occurrence of link congestion events.
3. Equipment failure related? → check the occurrence of outage events reported by the KPIs monitored by the ISP on its internal equipment, including routing/switching/forwarding equipments.
4. SDN/NFV-related? → check for (re)configuration events, Hypervisor failure-reporting events, etc. occurring at the times of the detected throughput anomalies.

- **Internet:**

1. inter-AS path-changes related? → check for end-to-end path change events in the corresponding temporal span of the detected anomaly.
2. path congestion related? check for flagged events related to abrupt increases in packet retransmissions per YouTube server, or in (avg RTT - min RTT) -- approximation to end-to-end queuing delay -- for all the flows provisioned by the corresponding YouTube servers.
3. intermediate AS issues related? → check for performance degradation events in the intermediate ASes, particularly including latency and congestion in the different end-to-end ASes path segments.

- **CDN servers:**

1. YouTube server reachability related? → verify if geo-distributed reachability measurements to the identified servers result in non-reachability problems.
2. YouTube server hardware/software related? → check for server hardware outages and/or software-related events at each single identified server IP during the tie span of the detected anomaly.
3. YouTube server overloading? → check for overloading events at each single identified server IP during the tie span of the detected anomaly.

The reader should note that the necessary measurements to verify this non-exhaustive list of diagnosis rules may or may not be available, depending of the extension of the monitoring layer.

3.8.3 Diagnosis/Iterative Analysis Graph

The previously described diagnosis rules are structured as a diagnosis graph, which is used for guiding the diagnosis and drill-down of the YouTube QoE-anomaly. As explained before, within mPlane we consider decision graphs in the form of decision trees. Fig. 34 depicts an exemplifying decision graph, integrating some of the previous diagnosis rules. As explained in 4.5, the branches

of a decision graph can be either conditionally or systematically followed. In our case, the analysis is conditional, starting from the end terminals till reaching the CDN servers.

The decision graph is structured in five different blocks, as follows:

1. QoE-based Anomaly Detection.
2. End-device Diagnosis.
3. ISP Diagnosis.
4. Internet paths Diagnosis.
5. CDN servers Diagnosis.

Note that these five blocks do not fully cover the aforementioned set of domain-knowledge based rules. Still, the description serves as an example on how to build a diagnosis graph within the mPlane framework. The **QoE-based Anomaly Detection** block consists of the anomaly detection analysis module, coupled with the QoE-baed monitoring for understanding whether the detected changes are causing QoE-based degradations or not. The **End-device Diagnosis** block focuses on the specific analysis of the type of end device associated to the anomalous YouTube flows. The **ISP Diagnosis** block consists of the diagnosis of the access ISP, which could be potentially materialized as described in 3.5. The **Internet paths Diagnosis** block focuses on the diagnosis of the end-to-end inter-AS paths, including both routing and path congestion analysis. Finally, the **CDN servers Diagnosis** block allows to identify server-related performance issues from end-to-end measurements, assuming that access to in-CDN measurements is not available for the realization of this use case.

3.8.4 Workflow of the Iterative Analysis

The iterative analysis performed by the Reasoner through this specific use-case-based decision graph assumes that a series of different continuous monitoring streams are being processed by the mPlane, which generate a series of logged diagnosis events. Also recall that the anomaly detection algorithm applied in this use case relies on the analysis of the complete empirical distribution of the monitored KPIs, and not on the analysis of a single percentile time-series. The following list enumerates the different streams which are processed to generate the diagnosis events that are analyzed in the iterative process:

- For all the filtered YouTube flows (with Tstat classification capabilities) observed at the vantage point, and exported to the DBStream repository:

1. time-series are continuously updated and analyzed for abrupt modifications for the following features: # flows, # bytes, # users, flow throughput, β QoE KPI, empirical entropy of QoE classes (bad, average, excellent), fraction of flows in the lowest QoE class, min RTT, average RTT, server elaboration time, fraction of retransmitted bytes per flow, empirical entropy
2. empirical distribution of average flow throughput per server IP is computed, and analyzed by the anomaly detection algorithm.

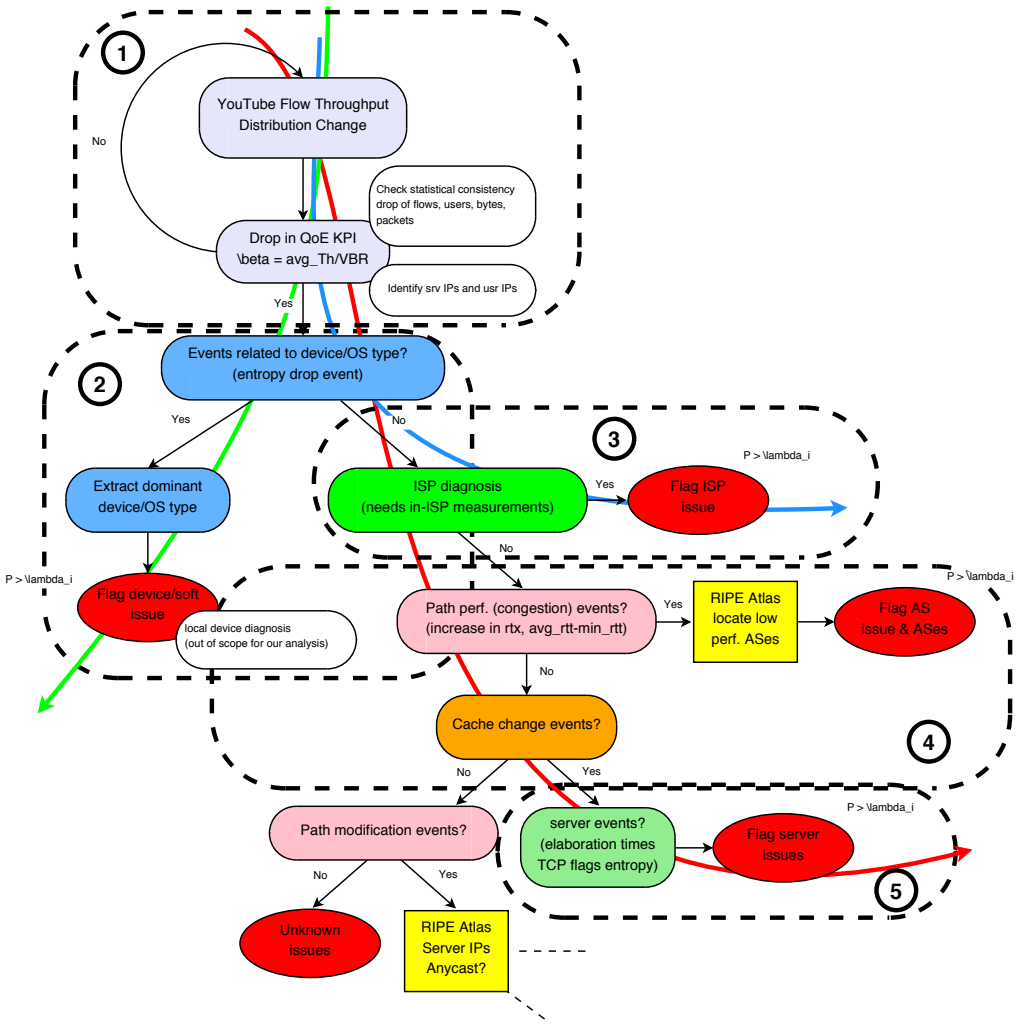


Figure 34: Diagnosis graph associated to the detection and troubleshooting support of large-scale QoE-based anomalies in YouTube.

3. empirical distributions of number of flows and bytes served per server IP or group of servers sharing the same network prefix (e.g., /24) are computed, and analyzed by the anomaly detection algorithm.

- For all the identified YouTube server IPs, the following time-series and events are tracked:

1. time-series of inter-AS paths (paths are tracked as ASes vectors), from the ASes where the server IPs are registered to till the vantage point.
2. for the aforementioned server IPs, path-change events are tracked (i.e., changes in the inter-AS path vectors).
3. for the aforementioned server IPs, the usage of IP-anycast is also registered.

Note that other events have to be similarly tracked at the different components of the end-to-end service, for example, at the access ISP, both at the access and at the core (related to the internal

routing tables, the routers, the aggregation network, the mobile stations, etc.), at the end-devices which are downloading the monitored YouTube flows, etc..

Assuming that the aforementioned measurements are available, the workflow of the anomaly detection use case goes as follows:

1. all traffic flows are analyzed by Tstat at the vantage point, and those belonging to YouTube are exported into DBStream.

2. the anomaly detection algorithm runs continuously on the YouTube flows within DBStream, considering as KPIs the per-flow average download throughput (to detect performance issues) and the number of flows served per /24 CDN subnetwork (to detect Google cache selection changes).

3. when an anomaly is detected as a major shift in the distribution of flows throughput towards lower throughput values, the diagnosis analysis is triggered (part (1) in Fig. 34).

4. the first step is to verify if the detected anomaly is statistically consistent, i.e., that it is not caused because of a big drop in the number of samples considered in the empirical distribution computation.

5. then the analysis verifies if this detected anomaly is actually impacting the QoE of the users, by analyzing the β QoE-based KPI. The analysis is not done on the distribution of β, but on the average and median values for all the downloaded flows. This is to better spot major QoE anomalies in YouTube.

6. the first diagnosis event to verify is a main drop on the time-series related to the empirical entropy of the operative system type of the devices downloading the captured flows. A drop in the empirical entropy would flag a major concentration on the distribution of the OS type of the devices, indicating a possible relation to the OS type (part (2) in Fig. 34).

7. the second diagnosis event to verify corresponds actually to a set of events related to the access ISP (part (3) in Fig. 34). We do not detail the specific events to evaluate at the ISP level, and take the steps presented in 3.5 as reference.

8. the third event to verify is the occurrence of performance degradation in the corresponding end-to-end paths carrying the impaired YouTube flows (part (4) in Fig. 34). Events tracked on the time series related to packet re-transmissions, queuing delay, etc. are checked in order to identify path congestion.

9. if path congestion is identified, then the Reasoner instructs active measurements from geo-distributed probes (e.g., using RIPE Atlas) to identify the specific AS or sub-path causing the performance degradation.

10. if no path performance degradation is observed, the analysis checks for events related to load balancing and cache selection modifications in the Google CDN serving the YouTube flows.

11. if no cache selection modification events are present in the logged events at the specific times of the detected YouTube QoE-based anomalies, the drilling-down checks for the occurrence of inter-AS routing changes which might be linked to the detected anomalies.

12. if cache selection modifications are present, then the analysis focuses on understanding if the new selected servers are the origin of the problems. For doing so, different application-level KPIs are verified on top of the monitored traffic, such as server elaboration times, TCP flags, etc..

This workflow is by no means complete, and more domain-based rules could be added to better drill down the targeted anomalies. Still, as we show next, such an iterative drill-down process allows to partially automate the detection and diagnosis of these so relevant anomalies in a complex service like YouTube. In particular, the following example describes the analysis process of a major YouTube QoE anomaly, probably caused by Google's cache selection policies, which chose servers that were not able to handle the load during the peak-traffic hours.

3.8.5 First Evaluation Results on the YouTube QoE Analysis

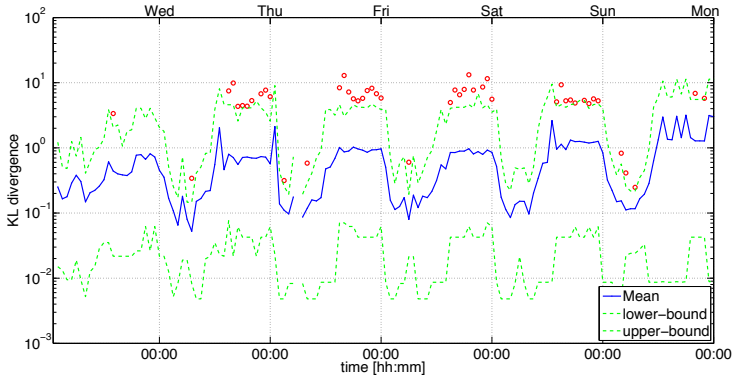
Let us consider a simplified version of the complete analysis to report some first evaluation results showing the functioning of the detection and diagnosis processes. The example consists of the detection and diagnosis of a Google's CDN server selection policy negatively impacting the watching experience of YouTube users during several days at peak-load times. Conversations with the ISP confirmed that the effect was indeed negatively perceived by the customers, which triggered a complete Root Cause Analysis (RCA) procedure to identify the origins of the problem. As the issue was caused by an unexpected caches selection done by Google (at least according to mPlane's diagnosis analysis), the ISP internal RCA did not identify any problems inside its boundaries.

Traffic flows are collected with Tstat at a link of a European fixed-line ISP aggregating 20,000 residential customers who access the Internet through ADSL connections. The complete data spans more than 10M YouTube video flows, served from more than 3,600 Google servers. Using Tstat filtering and classification modules, we only keep those flows carrying YouTube videos. The captured flows are periodically exported into the DBStream repository, where the analysis takes place. As reported by the ISP operations team, the anomaly occurs on Wednesday the 8th of May. We therefore focus the analysis on the week spanning the anomaly, from Monday the 6th till Sunday the 12th. In the following analysis, we generally use 50% percentile values instead of averages, to filter out outlying values.

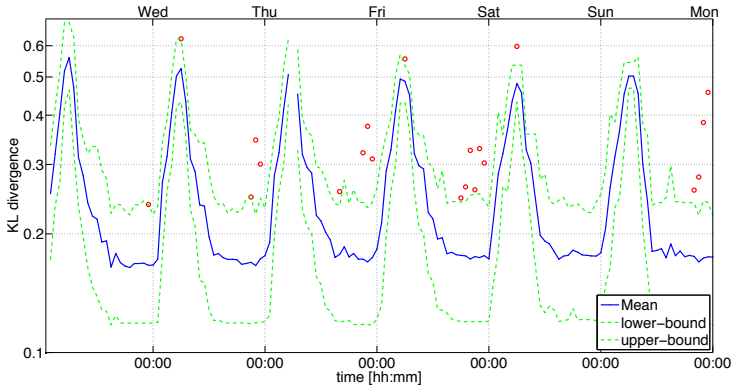
3.8.5.1 Detecting the QoE-based Anomaly

Fig. 35 depicts the output of the Anomaly Detection algorithm. Fig. 35(a) considers the per Google CDN /24 subnet served volume as the monitored feature. The red markers indicate when an anomaly is detected. From Wednesday the 8th of May onward the algorithm systematically rises alarms from 15:00 to 00:00, which correspond to a change in the cache selection policy, as we shall see next. In addition, fig. 35(b) reports the same information for the average video flows download rate. In this case, the analysis module detects some anomalies only between peak hours (21:00-23:00) from the 8th onward, suggesting that the throughput degradation are linked to high utilization of resources. Let us begin by understanding if the detected drop in the YouTube flow throughput has an impact on the QoE of the end users.

Fig. 36 plots the time series of three different performance indicators related to the YouTube download performance and to the end-user QoE. Fig. 36(a) depicts the median across all YouTube flows of the download flow throughput during the complete week. There is a normal reduction of the



(a) Anomalies in traffic volume served by CDN /24 subnets.



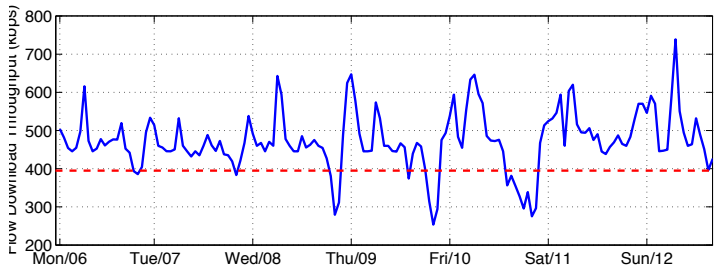
(b) Anomalies in the video flows average download rate across the YouTube users.

Figure 35: Detection of anomalies in YouTube traffic. Alarms and acceptance region for the distribution of (a) volume and (b) video flows average download rate. The red markers correspond to the flagged anomalies.

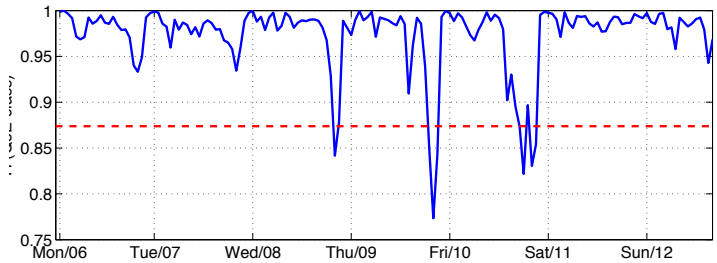
throughput on Monday and Tuesday at peak-load time, between 20hs and 23hs. However, from Wednesday on, this drop is significantly higher, and drops way below the bad QoE threshold $T_{h_1} = 400$ kbps, flagging a potential QoE impact to the users. Fig. 36(b) plots the entropy of the QoE classes built from thresholds $T_{h_1} = 400$ kbps and $T_{h_2} = 800$ kbps, consisting of bad QoE for flows with average download throughput below T_{h_1}, fair QoE for flows with average download throughput between T_{h_1} and T_{h_2}, and good QoE for flows with average download throughput above T_{h_2}. Recall that these thresholds correspond to the QoE mappings presented in Fig. 32, which only cover 360p videos. The drop in the throughput combined with the marked drop in the time series of the QoE classes entropy actually reveals that a major share of the YouTube videos are falling into the bad QoE class. Finally, Fig. 36(c) actually confirms that these drops are heavily affecting the users experience, as the time series of the KPI β falls well into the video stallings region, depicted in Fig. 33.

3.8.5.2 Anomaly Diagnosis

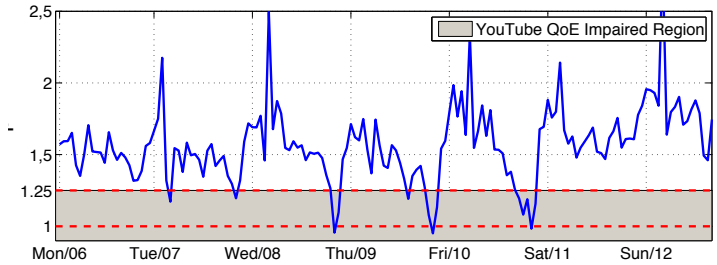
The root causes of the detected anomalies can be multiple: the Google CDN server selection strategies might be choosing wrong servers, the YouTube servers might be overloaded, path changes with much higher RTT from servers to the customers might have occurred, paths might be congested, or there might be problems at the access network. Diagnosing problems at the access network is



(a) 50%-p of the flow download throughput per hour for all YouTube flows.



(b) Entropy of QoE classes per hour for all YouTube flows.



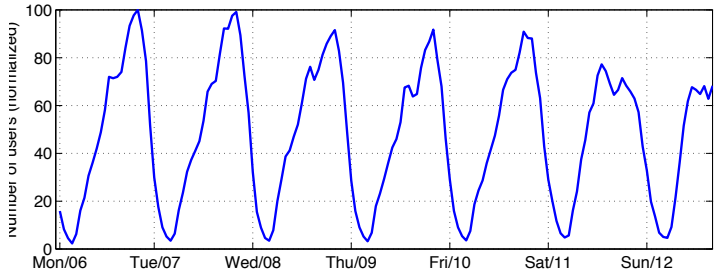
(c) 50%-p of β per hour for all YouTube flows.

Figure 36: Detecting the QoE-based anomaly. There is a clear drop in the download flow throughput from Wednesday till Friday at peak-load hours, between 20hs and 23hs. The combined drop in the entropy of the QoE classes and in the KPI β reveal a significant QoE degradation.

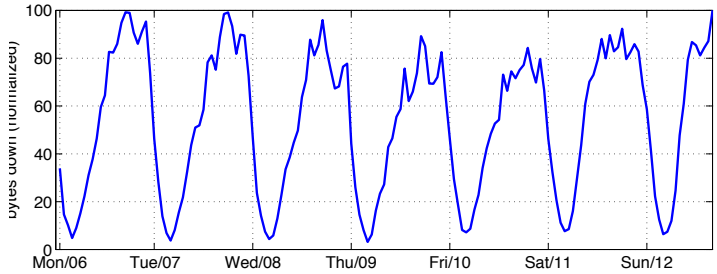
straightforward for the ISP, as this network belongs to itself. However, diagnosing the problem outside its boundaries is a much more complex task. As we said before, the ISP internal RCA did not identify any problems inside its boundaries, so we focus on the YouTube servers and on the download paths.

Fig. 37 depicts the time series of the per hour users and bytes down normalized counts during the analyzed week. While there is a drop in the number of bytes down from Wednesday afternoon on, there are no significant variations on the number of users during the working week (i.e., Monday till Friday), so we can be sure that the throughput and QoE strong variations observed in Fig. 36 are not tied to statistical variations of the sample size. Using the results in Fig.33(c), we can say that the drop in the bytes down suggests that the bad QoE affected the users engagement with the video playing, resulting in users dropping the watched videos when multiple stallings occur (i.e., when $\beta < 1.25$).

We study now the YouTube servers selection strategy and the servers providing the videos. Fig. 38(a) depicts the number of server IPs providing YouTube flows per hour. As depicted in Fig. 38(b), where the entropy of the AS number of the monitored server IPs is presented, there is a sharp shift



(a) Users count (normalized).



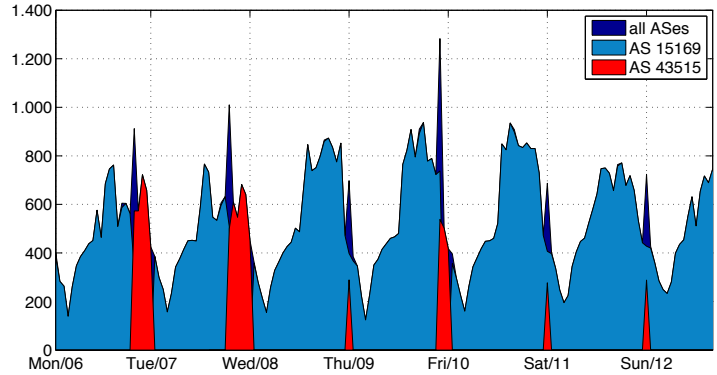
(b) Bytes down count (normalized).

Figure 37: Users and bytes down during the week of the anomaly. There are no significant changes during the specific times of the flagged anomaly.

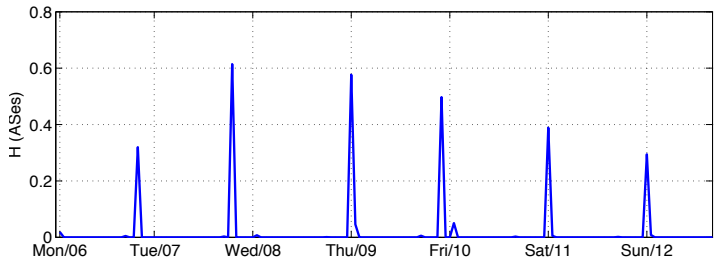
of servers from AS 15169 to AS 43515 around peak-loud hours. In addition, there is an important reduction on the number of servers selected from AS 43515 on the days of the anomaly. This suggests that a different server selection policy is set up exactly on the same days when the anomalies occur.

To further investigate this CDN server selection policy change, Fig. 39(a) shows the TSP of the video volume served by the different IPs in the dataset per hour, aggregated in /24 subnetworks, for 11 consecutive days. Recall that in the TSP, each point $\{i, j\}$ represents the degree of similarity between the distributions at hours t_i and t_j. The blue palette represents low similarity values, while reddish colors correspond to high similarity values. The TSP is symmetric around the 45° diagonal, thus the plot can be read either by column or by row. For a generic value of the ordinate at t_j, the points on the left (right) of the diagonal represent the degree of similarity between the past (future) distributions w.r.t. the reference distribution at t_j. Note the regular ``tile-wise'' texture within a period of 24 hours, due to a clear daily periodicity behavior in the selected servers. Specifically, there are two subnet sets periodically re-used in the first and second half of the day. The TSP clearly reveals that a different subnet set is used during the second half of the day from the 8th of May on, revealing a different cache selection policy. This change is also visible in the CDFs of the per subnet volume depicted in Fig. 39(b). Indeed, we can see that the same set of subnets is used between 00:00 and 15:00 before and after the anomaly, whereas the set used between 15:00 and 00:00 changes after the 8th, when the anomaly occurs.

Given this change in the server selection policy, we try to find out if the problem arises from the newly selected servers, or if the problem is located in the path connecting these servers to the users. Fig. 40 studies the latency from users to servers during the complete week. Fig. 40(a) depicts the median of the min RTT per hour as measured on top of all the YouTube flows. The marked increase in the RTT evidences that the servers selected during the anomaly are much farther than those



(a) IPs per hour hosting YouTube.



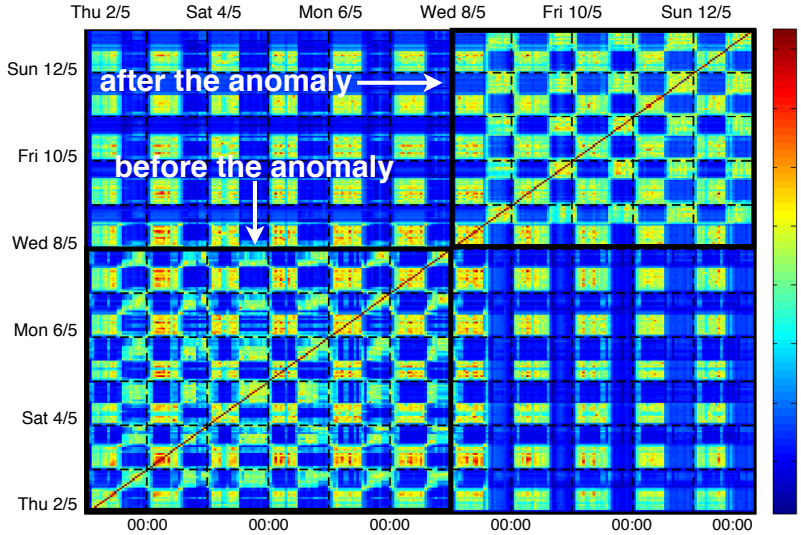
(b) Entropy of the AS numbers time-series.

Figure 38: IPs hosting YouTube during the week of the anomaly.

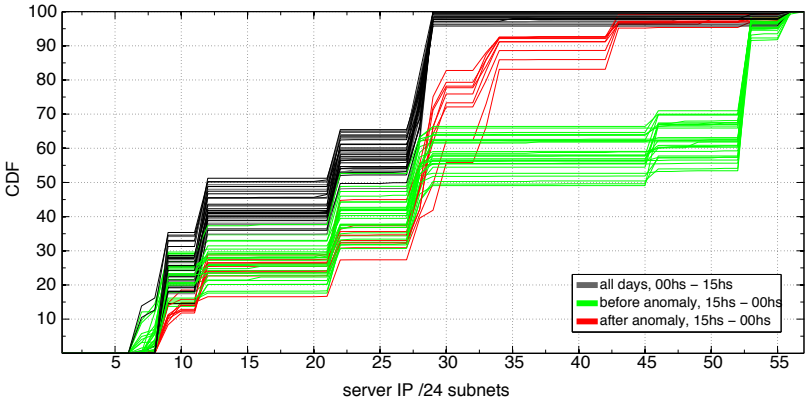
used before the anomaly. This increase impacts directly on the HTTP elaboration time (i.e., time between HTTP request and reply), as depicted in Fig. 40(b). To understand if these latency increases are additionally caused by path congestion, Fig. 40(c) plots the time series of the difference between the min RTT and the average RTT values; in a nutshell, in case of strong path congestion, the average RTT shall increase (queuing delay), whereas the min RTT normally keeps constant, as it is directly mapped to the geo-propagation delay. The differences before and during the anomalies do not present significant changes, suggesting that the paths between servers and clients are not suffering from congestion. This is also confirmed by the analysis of the packet retransmissions, which do not present significant variations.

The last part of the diagnosis focuses on the YouTube servers. Fig. 41 depicts the average (a) min RTT and (b) download flow throughput per server IP in a heatmap like plot. Each row in the plots corresponds to a single server IP. The previously flagged min RTT increase is clearly visible for the new set of IPs which become active from 15:00 to 00:00 from Wednesday on. For those server IPs, Fig. 41(b) shows the important throughput drop during peak-load hours. Note however that large min RTT values do not necessary result in lower throughputs, as many of the servers used before and during the anomaly are far located but provide high throughputs. Fig. 42 further studies this drop, comparing the relation between min RTT and average download flow throughput before and during the anomaly. The increase of the min RTT is not the root cause of the anomaly. However, there is a clear cluster of low throughput flows coming from far servers during the peak-load hours.

The conclusion we draw from the diagnosis analysis is that the origin of the anomaly is the cache selection policy applied by Google from Wednesday on, and more specifically, that the additionally selected servers between 15:00 and 00:00 were not correctly dimensioned to handle the traffic load



(a) TSP of volume per CDN /24 subnet.



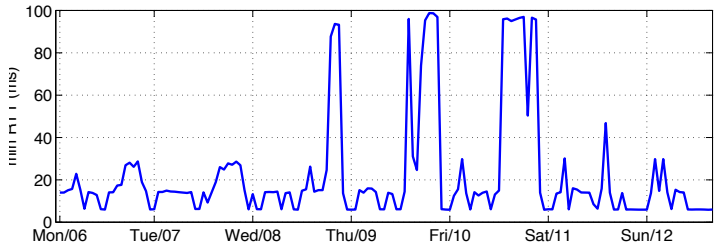
(b) Volume per CDN /24 subnet.

Figure 39: Traffic volume distributions per CDN /24 subnets. There is a clear shift on the selected caches serving YouTube before and after the reported anomaly on Wednesday the 8th of May, specifically in the afternoon, between 15:00 and 00:00.

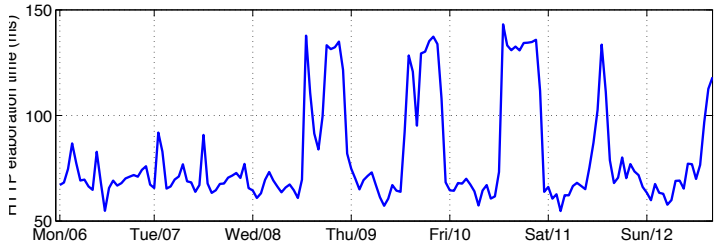
during peaks-hours, between 20:00 and 23:00. This shows that the dynamics of Google's server selection policies might result in poor end-user experience, on the one hand by choosing servers which might not be able to handle the load at specific times, or even by selecting servers without considering the underlying end-to-end path performance.

3.8.6 Complementary Evaluation Results on the Akamai CDN Analysis

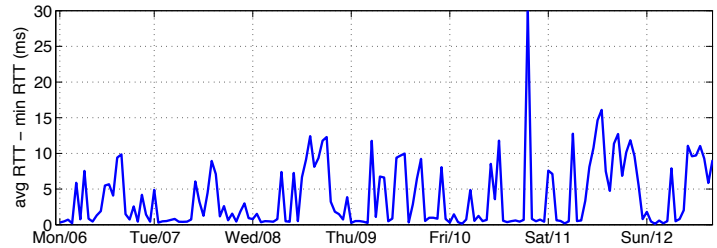
As complementary application results of the anomaly detection and diagnosis use case through mPlane, we focus now on the detection of major changes in the servers selected by major CDNs to deliver their services. CDNs are a vital part of the current Internet infrastructure. By deploying servers in multiple data centers across the Internet, content can be served to end-users with high availability and performance. However, CDNs pose challenges for ISPs, since changes in server allocation policies can cause sudden changes to the traffic carried by ISPs, impacting traffic engineering



(a) 50%-p of min RTT per hour for all YouTube flows.



(b) 50%-p of HTTP elaboration time per hour for all YouTube flows.



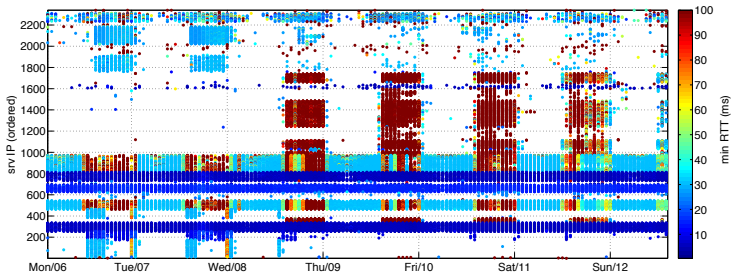
(c) 50%-p of avg RTT - min RTT per hour for all YouTube flows.

Figure 40: The servers selected during the anomaly are much farther than those used before. While there is a marked increase in the server elaboration time, the difference between avg. and min. RTT remains bounded during the anomaly, so we discard the hypothesis of path congestion.

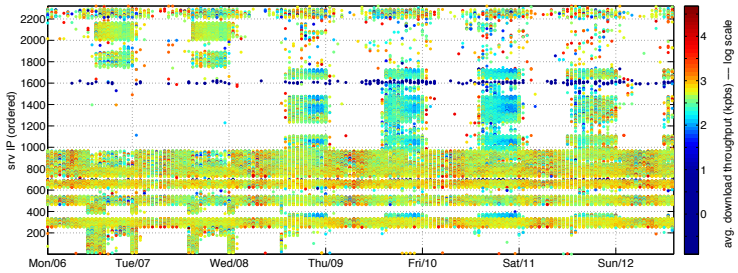
and possibly impairing end-user quality of experience. As such, ISPs need advanced tools to track and diagnose shifts in the traffic served by CDNs. Among CDN companies, Akamai is the leading CDN provider.

We thus instrumented mPlane to track the traffic served by the Akamai CDN servers as seen from a large ISP [46]. In this case, mPlane is instrumented using multiple Tstat passive probes, DBStream as repository, and a single supervisor hosted by the ISP. Tstat probes provide per-flow statistics from three Points-of-Presence (PoP) aggregating 45.000 end-users connected to the Internet, which are exported to the DBStream repository for further processing. Flows are pre-filtered at DBStream by correlating the server IPs with the external MaxMind databases. Specifically, we focus the analysis on a single /25 subnet hosting Akamai caches, which serve the majority of the flows. Servers in this network are reached by a direct peering agreement between the ISP and Akamai. Nodes are very close, typically less than 5ms far away from customers in the monitored PoPs. We refer to this subset as ``preferred'' in the following analysis.

The iterative analysis performed by the Reasoner follows the tree-like structure depicted in Fig. 43. We do not provide a full description of the involved processes and analysis modules due to space limitations, but rather follow the sequence of analysis steps involved in the diagnosis of cache selection shifts observed at the Akamai preferred cache. We note that the main goal of this case



(a) Average min RTT per server IP.



(b) Average download flow throughput per server IP.

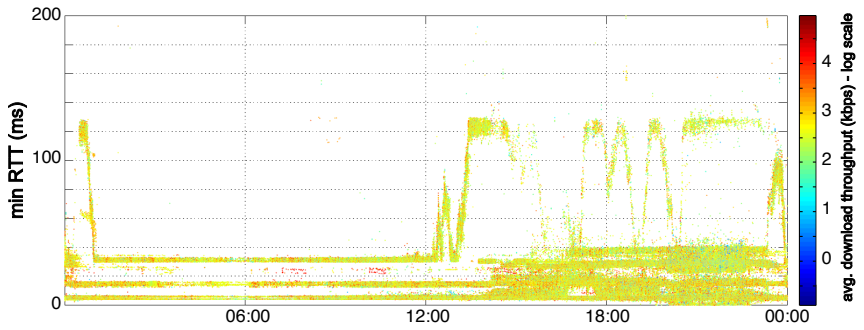
Figure 41: There is a new set of server IPs providing YouTube videos from Wednesday on from farther locations. As visible in (b), the average download flow throughput for each of these new server IPs is much lower than the one obtained from other servers.

study is to exemplify the iterative, guided measurements analysis process.

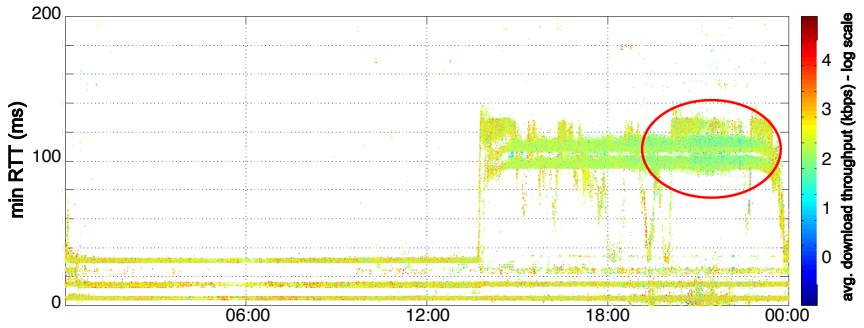
Overall daily patterns and change detection: Fig. 44 (top) details the evolution of the number of flows served by the Akamai CDN on two consecutive days as seen from one vantage point. The preferred cache serves about 30% of traffic at peak time. Surprisingly, traffic served by the preferred cache presents occasional drops. These are effects of the CDN server selection policies shifting traffic back and forth among CDN nodes. Fig. 44 (bottom) reports the evolution of the difference of the number of flows served by the preferred cache in two consecutive time windows of 5 minutes. The iterative analysis process is triggered by the detected abrupt changes in the number of served flows, marked as step (1) in Fig. 43.

Single servers load: The step (2) of the analysis corresponds to checking if the sudden traffic shifts are due to some server failure in the preferred subnet. Fig. 45(a) reports a heatmap of the load for each IP address in the /25 subnet over the 2 days. For each IP address, DBStream computes the fraction of served flows in 5 min. time windows. A color scale is used to represent each cell. The smaller the value, the lighter the color. Only 40 servers are active and constantly used, and few servers handle up to 62% of requests (darker red lines). All servers show lighter colors in correspondence of the traffic shift, thus the server failure hypothesis is ruled out.

Per-service analysis: CDN nodes host very different content, e.g., the same CDN server can host both Facebook and iTunes/AppleStore objects. Tstat exposes this information by snooping the Full Qualified Domain Name (FQDN) of the requested content. At step (3), the Reasoner checks if the observed traffic shifts are due to the CDN moving some specific content, reflecting some service-related issues. DBStream filters flows per service, and computes the fraction of requests served by the preferred and other caches. The obtained values are represented by the heatmap shown in Fig. 45(b). The most popular services are reported, sorted by the probability of being served by the



(a) min RTT and avg. download throughput per flow - before the anomaly.



(b) min RTT and avg. download throughput per flow - during the anomaly

Figure 42: The increase of the min RTT is not the root cause of the anomaly, as there are no major issues previous to the anomaly. However, there is a clear cluster of servers offering low throughput during the peak-load hours on an anomalous day.

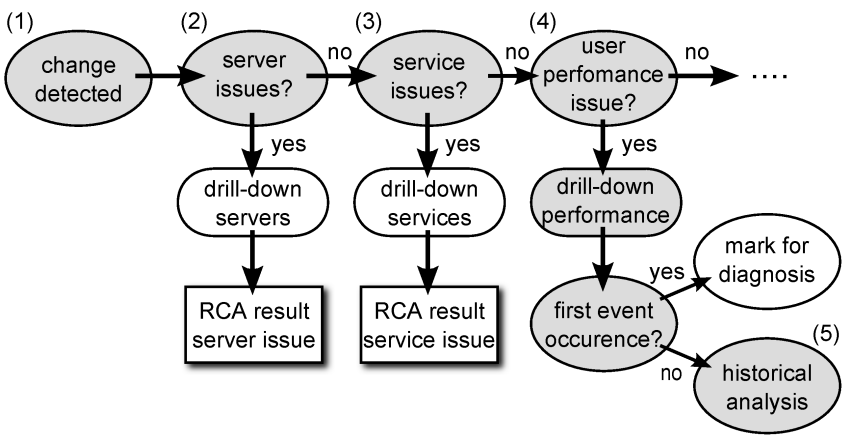


Figure 43: A rule-based reasoning approach for CDN cache selection analysis.

preferred cache. The results clearly show two groups: the bottom 300 services are normally served by some server at the preferred cache (red dots). The other 200 services are served exclusively by other Akamai CDN servers (green dots). At the same time as the traffic shifts occur, practically all services are migrated to other caches, indicated by the green vertical bars in the plot. Results indicate that the traffic shifts are not related to some particular service, but are rather the effect of changes in the server allocation policies impacting all services.

Impact on performance: Step (4) corresponds to the verification of the end-user performance.

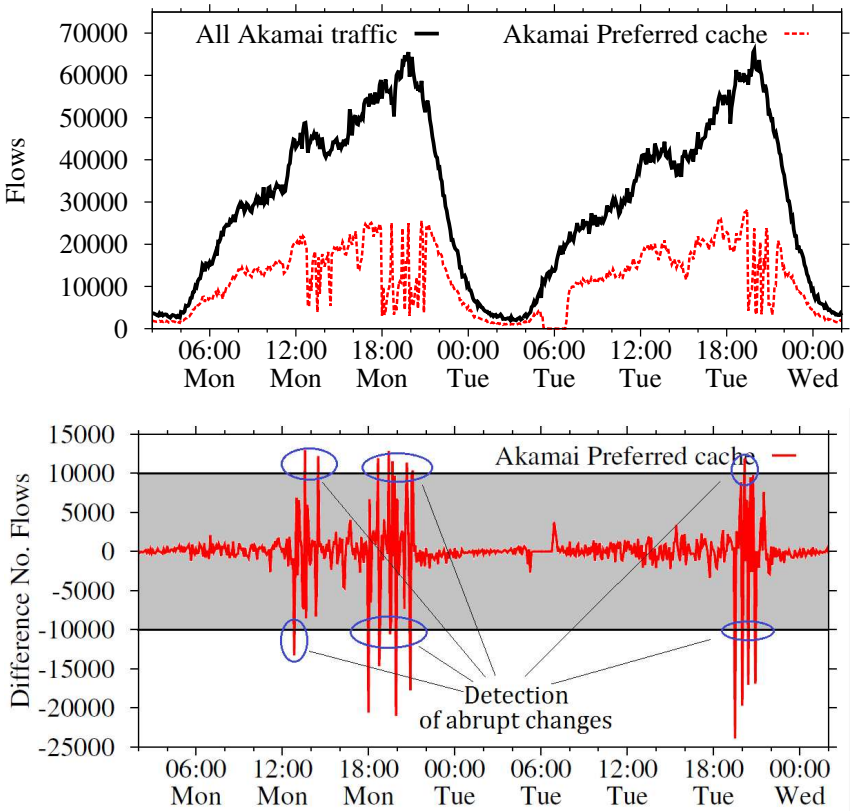
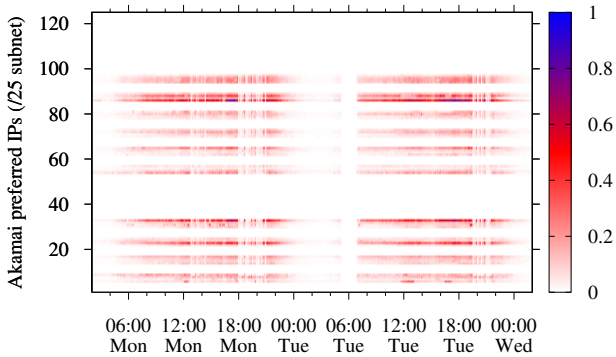


Figure 44: Evolution of number of flows served by Akamai CDN (top) and difference of number of flows served in consecutive 5 minutes time windows (bottom).

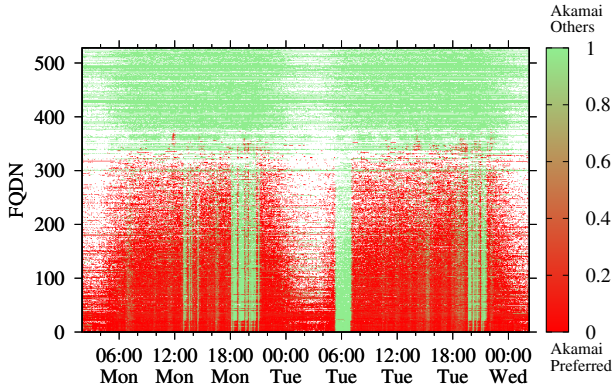
The analysis is performed both in terms of downlink throughput and delay. The analysis of the downlink throughput does not reveal any interesting evidence, thus we move on to the analysis of the elaboration time[9]. Fig. 45(c) reports the evolution of the 5th, 25th, 50th, and 75th percentiles of the elaboration time for the considered time period (y-axis is in log scale). Results show that during the traffic shifts on Monday, some impairment of the elaboration time is visible. In particular, the 50th percentile grows from about 10 ms to about 20 ms before and during the shifts happening at 18:00. Even if the same traffic shifts occur also on Tuesday, the 50th percentile of the elaboration time does not increase. The analysis of a whole week of traffic before the event (i.e., step (5), historical analysis) reveals that the same 50th percentile increase happens on all days before Tuesday, but does not occur on Tuesday and the following days. Still, also on Tuesday and on the days after, the same traffic shifts from the preferred cache occur.

The historical analysis does not provide a final root cause for the flagged traffic shifts. Yet, it allows to reveal the occurrence of a maintenance event on Tuesday, visible at Fig. 44 from 5am to 7am as a CDN outage. The most interesting observation from such flagged maintenance event is that the end-performance issues in terms of delay are solved after it, as confirmed in Fig. 45(d), which compares the per-day average RTT between users and serves, before and after this maintenance event.

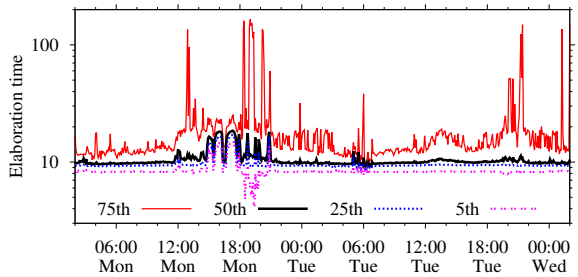
[9]The time between the client first packet with payload, and the server first packet with payload. In case of HTTP, it corresponds to the time between the `HTTP-request` and the `HTTP-response`.



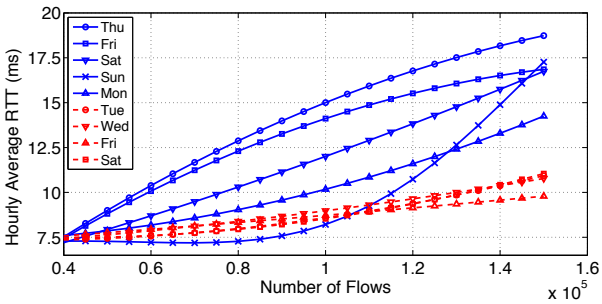
(a) Evolution of traffic load for each IP in the preferred cache.



(b) Evolution of the volume of requests per service name.



(c) Evolution of server elaboration time percentiles for the preferred cache.



(d) Daily average RTT vs flow counts for 9 days.

Figure 45: Iterative analysis of cache selection policies in Akamai.

3.9 Verification and Certification of Service Level Agreements

The goal of this use case is to certify Service Level Agreements and release a certificate. An SLA is a negotiated agreement between two parties, where one is the customer and the other is the service

provider. To certify SLA we use active measurement (RTT, TCP and UDP tests). To release an SLA certification is required an intelligent system that will analyze the data.

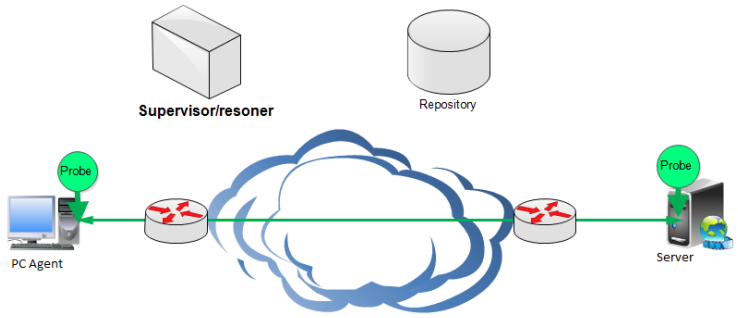


Figure 46: SLA certification probe (Agent and Server)

3.9.1 The Role of the Reasoner

The Reasoner receives the alarms from the repository for various problems detected on the measured data. After which the Reasoner may make additional requests to the probe for further tests, or to repeat a single or multiple tests.The Reasoner, in the simplest case, when the repository generates an alarm for UDP error, the Reasoner requests all the data that are related to that error. After having access on all the data related to the error, the Reasoner makes the calculation for the correct UDP speed and it makes a request to the probe to reduce the speed on the calculated one for confirmation. The main roles of the Reasoner are:

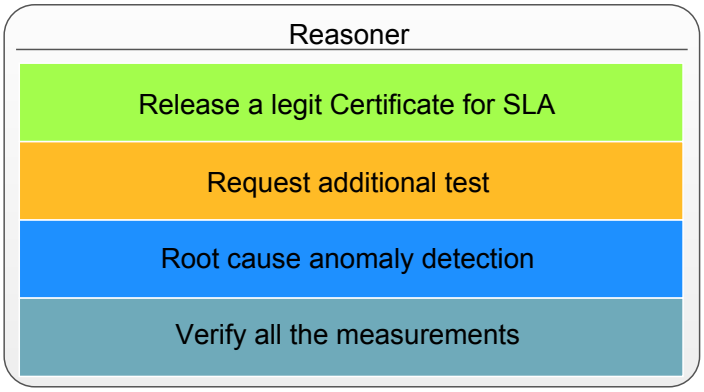


Figure 47: Main roles of the Reasoner

- Verify that all the measurements are correct. The Reasoner verifies if the bandwidth, layer 2 and 4, was stable or if any measurement had big degradation during the tests. It correlates all the means value of all tests, if some data do not correlate the Reasoner may discard them.
- In case the SLA fails, it has to analyze the data for root cause anomaly detection. When the SLA fails the Reasoner can try to find out the root cause of the problem. This is done with the

help of tstat passive probe. After the root cause analysis has completed the Reasoner sends a report to the user about the possible cause of th SLA fail.

- Request additional measurements, when other data are needed for the SLA certification. Usually when the Reasoner finds errors on the data it requests additional tests from the active probe, which may include: ping, traceroute, repeat UDP test with different settings that are calculated by the Reasoner, etc..
- Release the SLA certificate, if everything checks, the Reasoner releases a valid legit certificate. The finality of the SLA measurement is to have a legit certificate for the minimum level of service offered or received. After the Reasoner has analyses all the data of all the test it releases a certificate, that contains the mean value of the active measurement, a detailed certificate may be released by the Reasoner on the user request.

3.9.2 Analysis WorkFlow

When the test are completed the results are send to the repository. The repository makes simples check on the received data. It check if the bandwidth, at layer 2 and 4, was stable or not and if there was an error reported on UDP. If the repository finds an error, it flags the measurement and sends an alarm to the Reasoner.

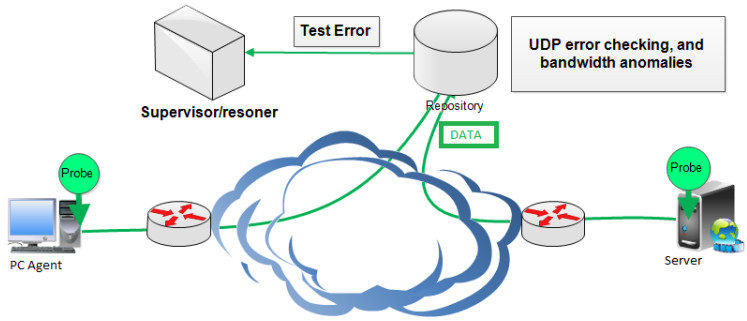


Figure 48: Generation of an alarm by the repository

When the Reasoner receives the alarm, it gets the related data to that alarm from the repository. On the analysis procedure first the Reasoner sees on what type of alarm it has received. The Reasoner can understand three types of alarms:

- UDP error more than 0.1% detected.
- Bandwidth instability detected.
- Unknown error.

Depending on the type of the received alarm the Reasoner has different procedures. If the Reasoner receives an Unknown alarm, it may discard the data, since there is a problem with the data that he cant understand. While the two main procedures are when the Reasoner receives and UDP error alarm. Upon receiving one of the alarms the Reasoner may also act on the probe by giving to it additional instructions, Fig. 49.

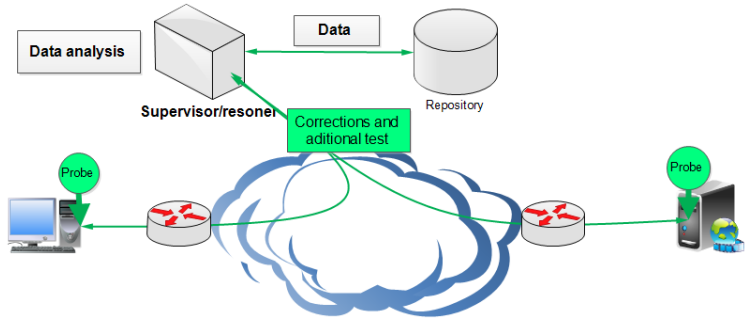


Figure 49: Reasoner requires additional measurements for the SLA process

3.9.2.1 Speed correction workflow

Step 1 - active measurement: To certify SLA active measurements are done, from one server to a client. As a default configuration the probe does 10 consecutive pings, one TCP test and one UDP test.

Step 2 - active alarm generation: After the probe completes the tests it send the result to the repository, which will make a simple analysis to check if there is any UDP error or anomaly. If it finds an error it will generate an alarm and send it to the Reasoner.

Step 3 - active speed correction: When the Reasoner receives the alarm it requests the data from the repository and begins the analysis. Upon identifying the UDP error the Reasoner calculates the correct speed as:

$$Correct\_Speed = Tested\_Speed * (100\% - error\_reported/100) \tag{3.1}$$

The Reasoner communicates to the probe, Fig. 49, that it needs to do an additional test, to confirm the correct UDP speed. This speed will be accepted as the line capacity of the client for the SLA certification, of course confirmation with all the measurement.

3.9.2.2 Anomaly detection workflow

In general, anomaly detection is a continuous process. The supervisor instructs the probes to periodically perform measurements, either passively or actively, and, thanks to correlation algorithms available at the repository, it compares measured features to baseline parameters. Periodically generated results are then forwarded by the supervisor to the analyst who can then take further actions.

The numbers in Fig. 50 refer to the steps which compose the workflow for the detection of anomalies. Each operation is mapped to the proper measurement component.

Step 1 - active probing: Active probes continuously perform tests to i) to monitor specific network parameters (e.g., they can perform `traceroute` towards given nodes), or ii) to resemble the behavior of actual users (e.g., they can emit requests for web-pages containing YouTube videos). Measurement results are transferred then to the repository.

Step 2 - augmenting information through passive monitoring: Passive probes run in parallel to active tests (for instance, on the same machine running the active measurements) and comple-

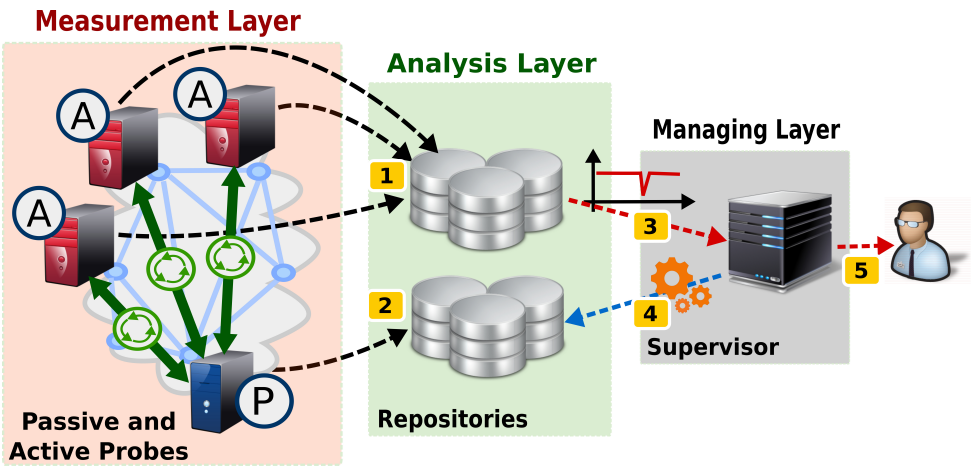


Figure 50: The distributed measurement platform inspired to mPlane for ISP network troubleshooting, and its workflow.

ment them with a continuous collection of traffic captures. The resulting measurement data is also periodically transferred to the proper repository.

Step 3 - detection of anomalies: Once measurement data is available at the repository, a set of selected features is sampled and aggregated into timeseries, and analysis modules embedding anomaly detection routines are run on them. As soon as a sudden change in the timeseries is detected, an alert is raised to the supervisor.

Step 4 - correlating multi-source measurement data: When alarms about unexpected degradations are detected, the supervisor runs correlation analysis (e.g., Factor Analysis) to investigate which features, or classes of features, show a similar abrupt change. Correlated features are then compared against a catalog of known anomaly patterns, and if a match is found, an alert is raised to the supervisor.

3.9.3 First Evaluation Results - Speed correction and anomaly detection

- Speed correction

When the probe does the UDP test it launches the transfer at max speed, on Fig. 51 the probe started the UDP measure at 500 Mbps, this resulted on an error of nearly 80%. When the Reasoner receives this alarm it makes the calculation for the correct speed with equation 3.1.

On Fig. 52 we have reported the variation of the UDP error depending on the transmitting speed on a 100 Mbps connection. As it can be seen the error decreases to zero after a speed 96 Mbps. In the example of Fig. 51, the error reported to the Reasoner was around 80%, while applying the formula of equation 3.1, the Reasoner calculated that the correct line capacity, on which the UDP should be launched was 95.5 Mbps.

Upon doing the calculation, as we sad before the Reasoner will request to the probe to repeat the UDP test at 95.5 Mbps. After the probe repeats the UDP test we have the result of Fig. 53. In this case the error reported after the Reasoner made the corrections was under 0.1%, that is acceptable for SLA purposes.

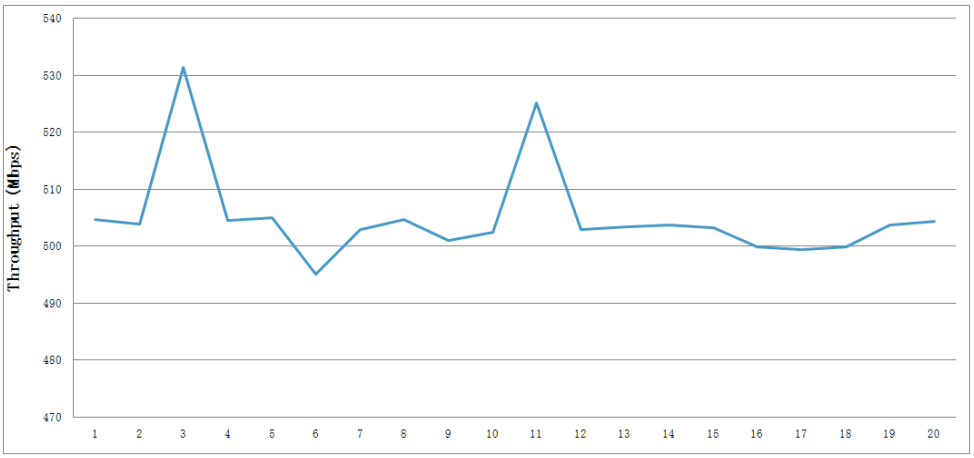


Figure 51: UDP test at maximal available speed.

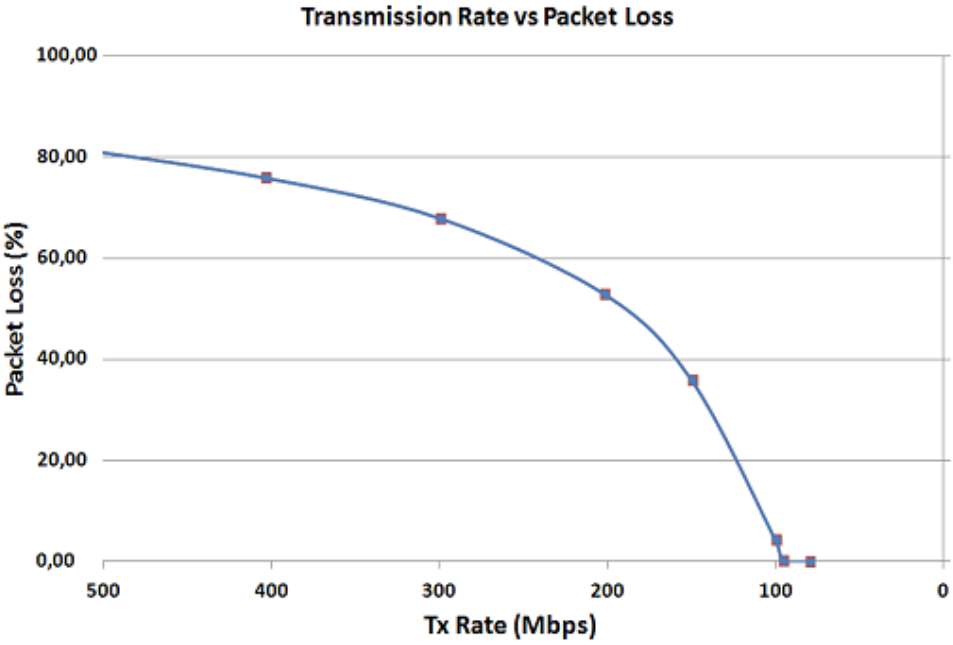


Figure 52: UDP error on a 100 Mbps connection.

- Results for anomaly detection[10]

This section its we report firs results on anomaly detection. This section is done in collaboration with Polito and Fastweb. The data reported here are from a period of three months on Fastweb networks, whom for the active measurement have used IQM probe, developed by Fastweb and for the passive measurements was used tstat. We also show that by correlating the results obtained by the active tests with those gathered from the passive probe, we can pinpoint the cause(s) behind the glitches.

By analyzing our dataset, we could identify three main classes of anomalies. We report them in the following.

[10] Exploiting Hybrid Measurements for Network Troubleshooting - Networks 2014 (Submitted)

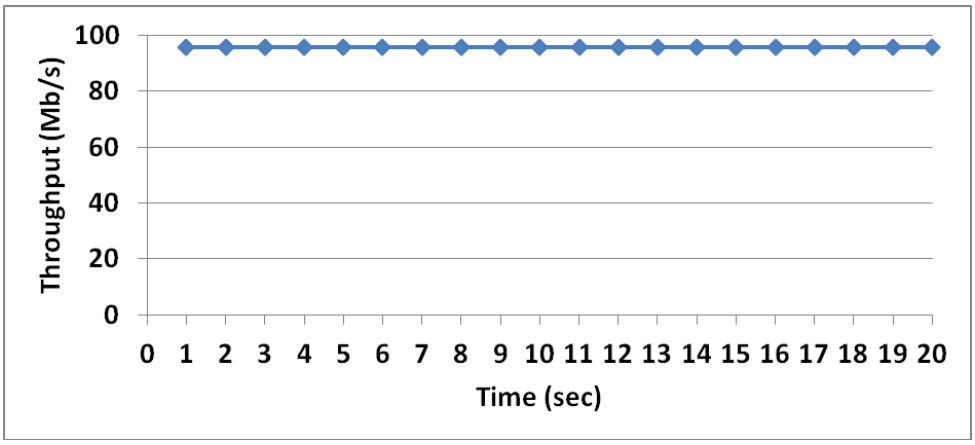


Figure 53: Correct speed calculated by the Reasoner and retested by the probe.

3.9.3.1 Low SNR on ADSL lines

In top plot of Fig. 54 we report the evolution over time of the throughput measured by an active probe accessing the ISP network through a U-1Mbps/D-16Mbps[11] ADSL interface for a period of two days. Observe that the download throughput curve appears to be noisy during the first day, while after midnight, the ADSL line was re-calibrated to U-1Mbps/D-8Mbps. Since then, speed-test measures are much more stable over time. By correlating such output with the statistics provided by Tstat, we could notice during the first day, a fairly large rate of retransmitted segments in the flows[12] (center plot), and a constant coefficient of variation of the RTT (bottom plot). The absence of evident day-night patterns let us exclude that this situation might be due to network congestion, since this typically emerges only during peak periods (see the next case).

The most probable cause of this anomaly is the presence of a low signal-to-noise ratio (SNR) at the physical link, which can lead to large bit error rate (BER). Losses due to noise then causes TCP congestion control to (randomly) slow down the download. The confirmation of this hypothesis is given by the second half of the plots in Fig. 54, when the ADSL modem automatically tunes the ADSL downlink capacity to improve the SNR, i.e., negotiating 8Mb/s instead of 16Mb/s, thus considerably reducing the packet loss rate, and making RTT measurement more stable.

3.9.3.2 Congestion in the Network

As before, top plot of Fig. 55 reports the evolution over time of the throughput measured by an active probe (U-1Mbps/D-12Mbps ADSL interface in this case). During the 48h observation window, a clear degradation of the available throughout is detected. We notice that no degradation is observable during the night, i.e., when the network is typically lightly loaded. Conversely, during peak time available capacity decreases. This suggests that there might be some congestion in the network. By inspecting the statistics provided by Tstat at the server side, we could confirm this intuition. Indeed, notice how the coefficient of variation of the RTT (bottom plot) and the rate of retransmitted packets (center plot) measured on TCP connections carrying FTP file transfers con-

[11] We remark that the throughput reported in the plots is below the nominal bandwidth value since it is measured at application level, therefore, netting transport, network, and data-link layers overheads.

[12] Tstat detects retransmissions in TCP flows by tracking the sequence number of transmitted packets.

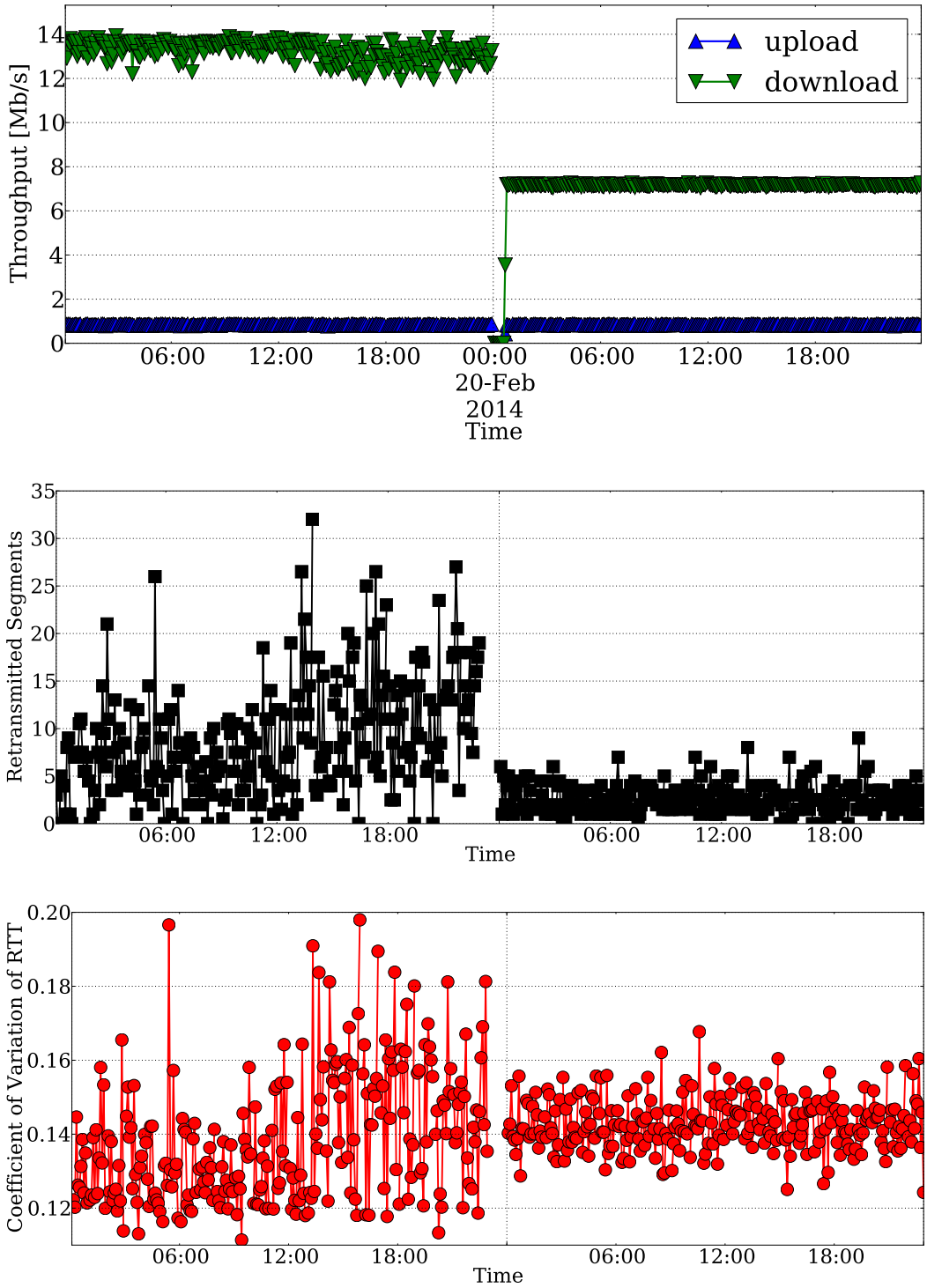


Figure 54: Evolution of time of the throughput measured by an active probe (top), the number retransmitted segments (center), and the coefficient of variation of the RTT (bottom). U-1Mbps/D-16Mbps ADSL probe.

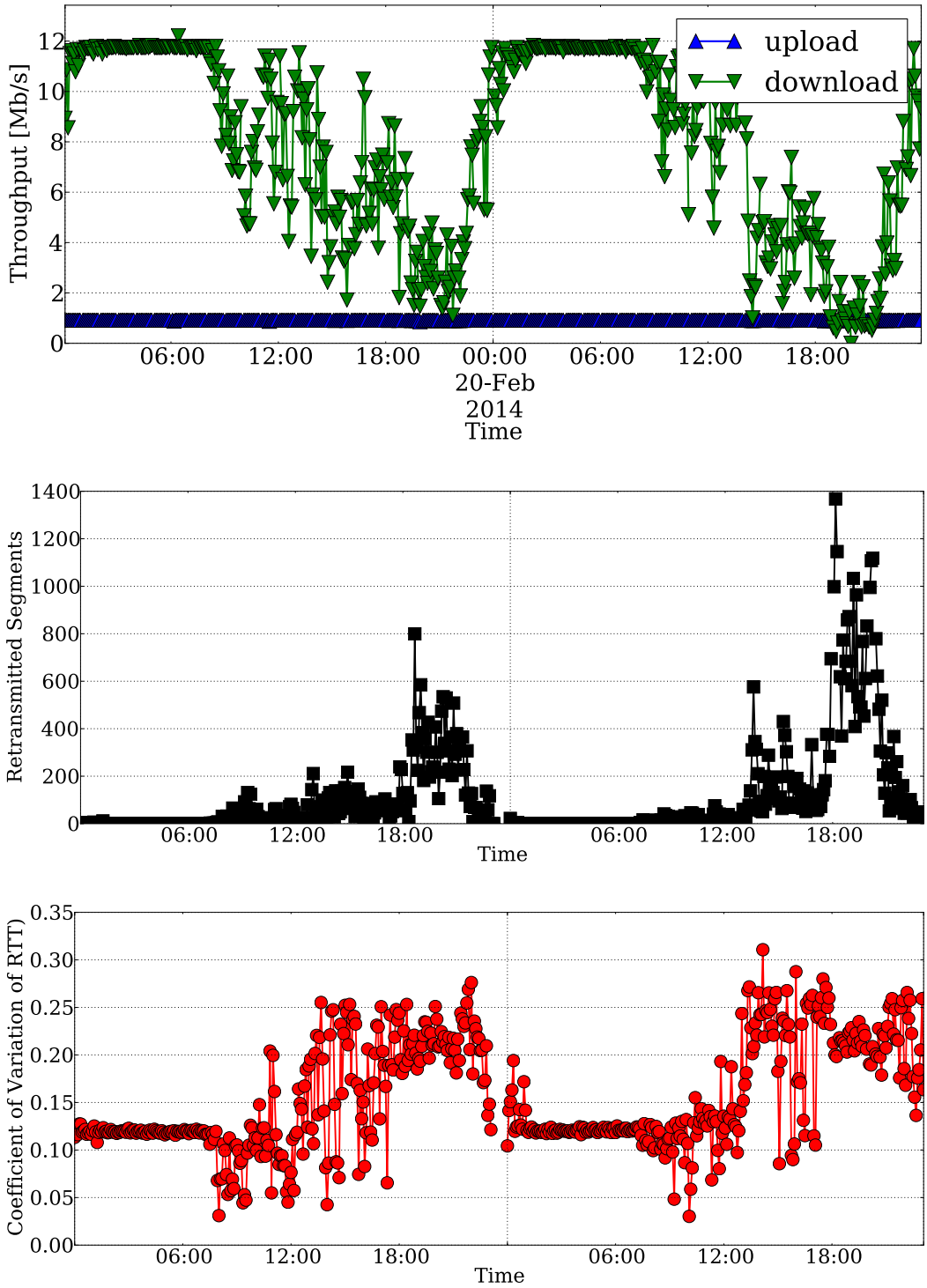


Figure 55: Evolution over time of the throughput measured by an actibe probe (top), the number of retransmitted segments (center), the coefficient of variation of the RTT (bottom). U-1Mbps/D-12Mbps ADSL probe.

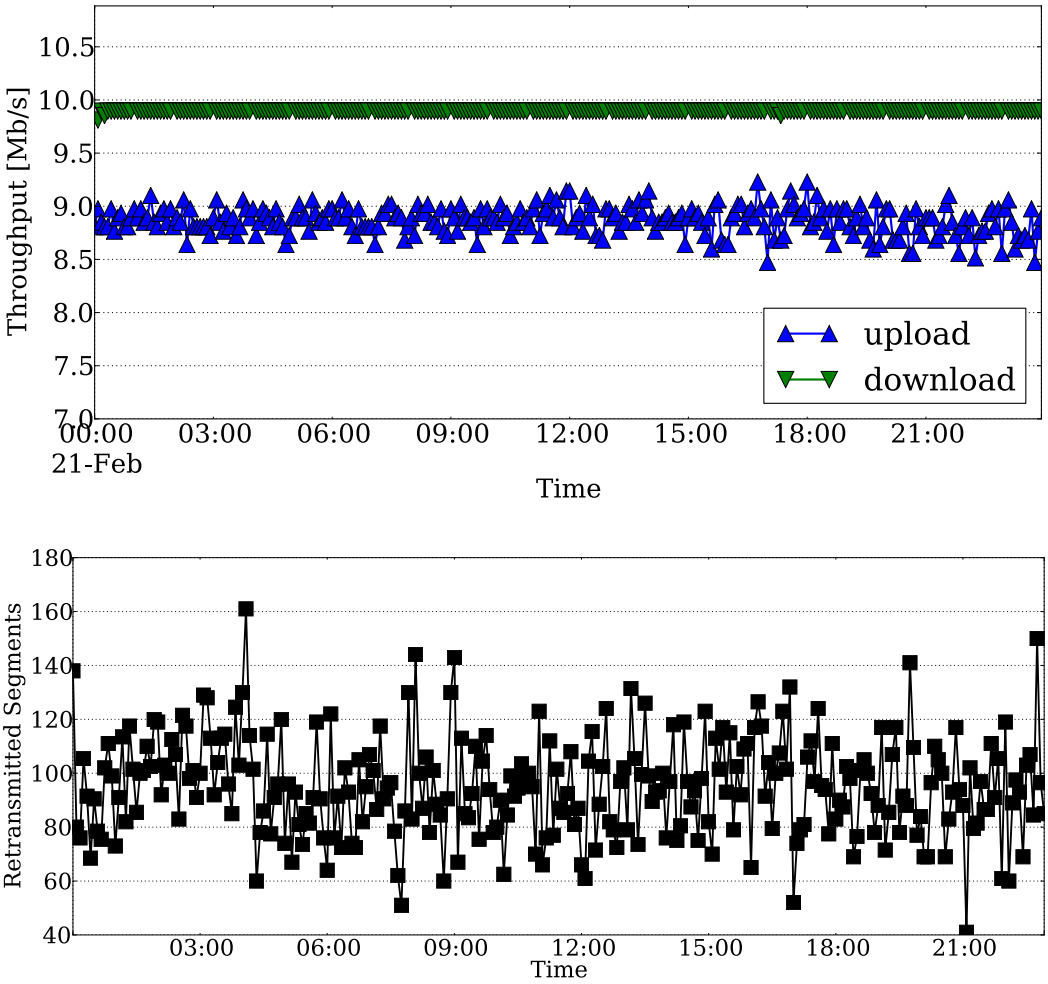


Figure 56: Evolution over time of the throughput measured by an actibe probe (top), and the retransmitted packet rate as measured by Tstat (bottom). U-10Mbps/D-10Mbps FTTH probe.

siderably increase during the network utilization period.

By manually checking, we could find out that such probe accesses the Internet through a bottlenecked Virtual Leased Line, and its available bandwidth is out of the control of the operator.

3.9.3.3 Congestion at the Uplink

In top plot of Fig. 56 we report the evolution over time of the throughput measured by an active probe accessing the ISP network through a U-10Mbps/D-10Mbps FTTH interface. Another kind of unexpected result is illustrated. Indeed, the upload throughput is below the expected value, and its profile is very noisy. While this may be common for an ADSL interface (see the first case), it is rather unexpected for the case of a fiber access link, which is not affected by low SNR issues. By inspecting Tstat statistics, we observe that the rate of retransmitted packets (bottom plot) is relatively high and constant in time, meaning that there is some time-independent loss on the path towards the server. We compared these results with others obtained from other probes accessing the network

through FTTH, but we could not notice any similar loss pattern at the uplink. This hints for problems at the FTTH home gateway, where possibly some self-congestion is induced when packets send by the FTP client (connected via a 100Mb/s fast Ethernet) are queued at the FTTH uplink buffer (running 10Mb/s). We instrumented then the IQM probe to flood the server via MTU-sized ICMP echo requests. This generates a burst of packets that enters the buffer. By observing the first packet that appears to be lost, we estimate the buffer size. Thanks to this further experiment we verify that such hag presented some issues in the management of its output buffer, thus reducing its actually employed size and causing losses during the upload speed-test.

3.9.3.4 Other Types of Anomalies

Finally, during our analysis we could also identify a link failure event happening at the core of the network. Such failure involved all probes in a specific portion of the network. The reports obtained from active probes together with the exceptional nature of the event were enough to reconstruct the cause behind such distributed degradation. However, even if aggregating active and passive measurements was not necessary in this case, we could detect it thanks to the distributed design of the mPlane architecture. Indeed, apart from enabling the correlation of measurement data generated from different kinds of sources, mPlane design allows to *spatially* correlate measurements coming from different points of the network.

4 Learning Approaches for the Reasoner

This section presents a set of different learning techniques to explore data and discover interesting and hidden relations among features related to the analyzed use cases. The final goal of such learning approaches is to enhance the analysis capabilities of the Reasoner, by extending the set of relevant events and analysis rules initially defined by expert domain knowledge. Besides the standard battery of supervised and unsupervised machine learning and data mining approaches usually applied in the domain of data analysis and knowledge extraction, we are working in some specific promising approaches for the case of real network measurements, which by nature are noisy, unlabeled, and do not always follow specific probabilistic distributions.

In particular, we present an approach for extracting knowledge out of unlabeled data based on sub-space clustering techniques, an approach for exploratory data analysis based on association rule mining techniques, and an approach for modeling spatio-temporal relationships among real-world measurements, where common statistical assumptions do not apply. In addition, we present some basic yet complementary discussion on the idea of correlating measurements from different sources, as well as discussing some interesting considerations for building diagnosis graphs within the Reasoner framework.

4.1 Clustering for Unsupervised Learning

When it comes to learning techniques for extracting useful information and detect the occurrence of specific patterns in large amounts of measurements, two different approaches are applied: supervised learning for classification and prediction, and unsupervised learning for pattern extraction and data mining. A supervised approach consists in building a model to recognize or predict the class (i.e., classification) or the value (i.e., regression or prediction) of a pattern, i.e., a set of features describing certain element under analysis. In the mPlane context, a pattern could be for example a traffic flow, a specific network event, etc.. Supervised-based models for recognition and prediction are generally highly effective to analyze known patterns, provided that a good learning set of patterns correctly labeled is available for training (what we call the ground truth). Supervised learning can also be applied to identify which are the best features describing a certain known pattern, i.e., it can be applied for feature selection. The main limitation of supervised-based learning approaches is that of requiring a well defined ground truth set of measurements to provide proper results. Depending on the nature of the data, labeling a dataset is generally not an easy task, which can be done either manually or under very controlled situations. Labeling datasets is not only time consuming and expensive, but also very prone to errors in the practice. In addition, in the case of traffic measurements in the wild, i.e., in the case of Internet based measurements, it is extremely difficult to obtain proper ground truth to construct classification and/or prediction models.

Our thesis is that supervised-based approaches are not sufficient to tackle the data analysis and knowledge generation problem when considering mPlane-like measurements, and that a holistic solution should also include knowledge-independent or ground-truth-independent analysis techniques. To this aim we propose an unsupervised learning based approach that is capable of detecting the occurrence of novel and significant changes in the features describing specific patterns without relying on labeled datasets and/or training. Based on the observation that novel network events are, by definition, events that deviate markedly from the majority of them, the proposed unsupervised approach relies on robust clustering algorithms to identify new clusters and outliers.

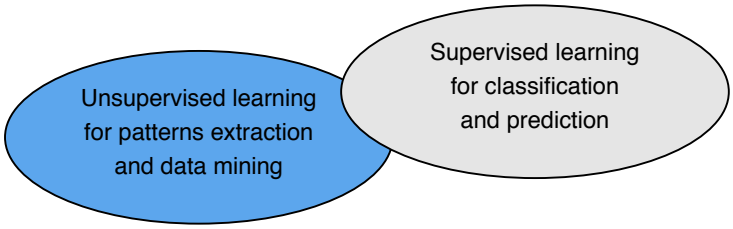


Figure 57: Different learning approaches for extracting useful information and detect the occurrence of specific patterns in large amounts of measurements.

In particular, we devised a robust multi-clustering algorithm based on a combination of Sub-Space Clustering (SSC) [34], Density-based Clustering [11], and Evidence Accumulation Clustering (EAC) [14] techniques. The method uses this multi-clustering algorithm to blindly extract the patterns which are novel and more relevant. The evidence of relevant structures provided by the clustering algorithm is used to produce filtering rules that characterize the identified patterns and simplify its analysis. The characterization of novel patterns can be in general a very hard and time-consuming task, particularly when dealing with unknown patterns. Even experts can be quickly overwhelmed if simple and easy-to-interpret information is not provided to prioritize the time spent in the analysis. To alleviate this issue, the most relevant filtering rules are combined into a new traffic signature that characterizes the identified patterns in simple terms. This signature can ultimately be integrated into the reasoner's knowledge base to identify its presence in the future, extending as such the complete diagnosis process.

4.1.1 Unsupervised Patterns Analysis

The unsupervised analysis takes as input a set of unlabeled patterns to analyze (e.g., IP flows, network events, time-slotted traffic measurements, etc.). Without loss of generality, let $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ be the set of n unlabeled patterns to analyze. Each pattern $\mathbf{y}_i \in \mathbf{Y}$ is described by a set of m features. Let $\mathbf{x}_i \in R^m$ be the vector of traffic features describing pattern $\mathbf{y}_i$, and $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in R^{n \times m}$ the complete matrix of features, referred to as the *feature space*.

The selection of features is a key issue to any learning-based algorithm, and it becomes critical in the case of unsupervised analysis, because there is no additional information to select the most relevant set. The features used can be both selected based on domain knowledge, and/or on a take-all basis, selecting as many features as possible, and then deciding which are the ones offering the best structure description. As we said before, such a feature selection can be directly done in the case of labeled data, but is more challenging to perform in the unsupervised case. Still, techniques based on Sub-Space Clustering [34] can be applied in this case to select the best sub-spaces of the feature space which provide the best structural information, based on the definition of a structural-information metric.

The complete approach is based on clustering techniques applied to $\mathbf{X}$. The objective of clustering is to partition a set of unlabeled patterns into homogeneous groups of similar characteristics or *clusters*, based on some measure of similarity. Samples that do not belong to any of these clusters are classified as *outliers*. Our goal is to identify in $\mathbf{Y}$ the different patterns that may reveal novel and relevant information, considering as baseline the normal behavior of the patterns. For doing so, the reader should note that novelty may consist of either outliers (i.e., single isolated patterns) or compact small-size clusters, depending on the way patterns are described (e.g., different ways of

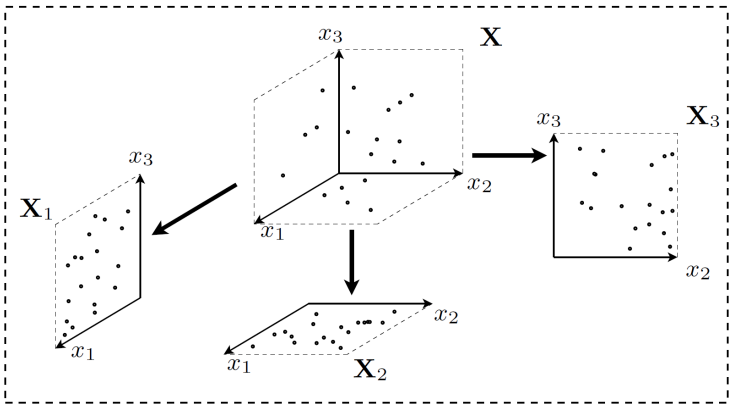


Figure 58: Sub-Space Clustering: 2-dimensional sub-spaces $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$ are obtain from a 3-dimensional feature space $\mathbf{X}$ by simple projection. Units in the graph are irrelevant.

aggregating traffic measurements). Unfortunately, even if hundreds of clustering algorithms exist [21], it is very difficult to find a single one that can handle all types of cluster shapes and sizes. Different clustering algorithms produce different partitions of data, and even the same clustering algorithm provides different results when using different initializations and/or different algorithm parameters. This is in fact one of the major drawbacks in current cluster analysis techniques: the lack of robustness.

To avoid such a limitation, we have developed a divide & conquer clustering approach, using the notions of *clustering ensemble* and *multiple clusterings combination*. These ideas are well-known within the machine-learning community, but the application of these techniques for network measurements analysis is novel and appealing: why not taking advantage of the information provided by multiple partitions of $\mathbf{X}$ to improve clustering robustness and identification of structure? A clustering ensemble $\mathbf{P}$ consists of a set of multiple partitions P_i produced for the same data. Each partition provides an independent evidence of data structure, which can be combined to construct a new measure of similarity that better reflects natural groupings and outliers. There are different ways to produce a clustering ensemble. For example, multiple partitions can be generated by using different clustering algorithms, or by applying the same clustering algorithm with different setting parameters or initializations. We particularly use Sub-Space Clustering (SSC) [34] to produce multiple data partitions, doing density-based clustering in N different sub-spaces $\mathbf{X}_i$ of the original space.

4.1.2 Clustering Ensemble and Sub-Space Clustering

Instead of directly partitioning the complete feature space $\mathbf{X}$ using a traditional inter-pattern similarity measure (i.e., the Euclidean distance), we do parallel clustering in N different sub-spaces $\mathbf{X}_i \subset \mathbf{X}$ of smaller dimensions, obtaining N different partitions P_i of the patterns in $\mathbf{Y}$. Each sub-space $\mathbf{X}_i$ is constructed using only $k < m$ traffic features; this permits to analyze the structure of $\mathbf{X}$ from $N(m, k)$ different perspectives, using a finer-grained resolution. Each $\mathbf{X}_i \subset \mathbf{X}$ is obtained by projection of $\mathbf{X}$ into k features out of the m attributes, resulting in N k-dimensional sub-spaces. To deeply explore the complete feature space, the number of sub-spaces N that are analyzed corresponds to the number of k combinations obtained from m.

Fig. 58 explains this approach; in the example, a 3-dimensional feature space $\mathbf{X}$ is projected into

$N = 3$ 2-dimensional sub-spaces $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$, which are then independently partitioned via density-based clustering. Each partition P_i is obtained by applying DBSCAN [11] to sub-space $\mathbf{X}_i$. DBSCAN is a powerful clustering algorithm that discovers clusters of arbitrary shapes and sizes [21], relying on a density-based notion of clusters: clusters are high-density regions of the space, separated by low-density areas. This algorithm perfectly fits our unsupervised analysis, because it is not necessary to specify a-priori difficult to set parameters such as the number of clusters to identify.

The results obtained by clustering sub-space $\mathbf{X}_i$ are twofold: a set of $p(i)$ clusters $\{C_1^i, C_2^i, .., C_{p(i)}^i\}$ and a set of $q(i)$ outliers $\{o_1^i, o_2^i, .., o_{q(i)}^i\}$. To set the number of dimensions k of each sub-space, we take a very useful property of monotonicity in clustering sets, known as the downward closure property: if a collection of elements is a cluster in a k-dimensional space, then it is also part of a cluster in any $(k-1)$ projections of this space. This directly implies that, if there exists any interesting evidence of density in $\mathbf{X}$, it will certainly be present in its lowest-dimensional sub-spaces. Using small values for k provides several advantages: firstly, doing clustering in low-dimensional spaces is more efficient and faster than clustering in bigger dimensions. Secondly, density-based clustering algorithms such as DBSCAN provide better results in low-dimensional spaces [21], because high-dimensional spaces are usually sparse, making it difficult to distinguish between high and low density regions. Finally, clustering multiple low-dimensional sub-spaces provides a finer-grained analysis, which improves the analysis characteristics. Our approach uses therefore $k = 2$ for Sub-Space Clustering, which gives $N = m(m-1)/2$ partitions.

Having produced the N partitions, the question now is how to use the information provided by the multiple clusters and outliers identified by density-based clustering. A possible answer is provided in [14], where authors introduced the idea of Evidence Accumulation Clustering (EAC). EAC uses the clustering results of multiple partitions to produce a new inter-patterns similarity measure that better reflects their natural groupings. In this direction, the information provided by the partitions P_i is combined to produce a new similarity measure between the patterns in $\mathbf{Y}$, which has the paramount advantage of clearly highlighting both those outliers and small-size clusters that were simultaneously identified in different sub-spaces. This new similarity measure is finally used to easily extract the relevant novel patterns from the rest. Briefly speaking, if we can find a group of patterns that are remarkably different from the rest in different sub-spaces, then we have found an novel piece of information worth to flag and further analyze.

4.1.3 Automatic Characterization of Detected Patterns

The following task after the detection of a group of relevant patterns is to automatically produce a set of K filtering rules $f_k(\mathbf{Y})$, $k = 1, .., K$ to characterize them. In the one hand, such filtering rules provide useful insights on the nature of the patterns, easing the analysis task of the network operator. On the other hand, different rules can be combined to construct a signature of the novel data, which can be used to easily detect its occurrence in the future. To produce filtering rules $f_k(\mathbf{Y})$, the algorithm selects those sub-spaces $\mathbf{X}_i$ where the separation between the identified patterns and the rest is the biggest. We define two different classes of filtering rule: *absolute* rules $f_A(\mathbf{Y})$ and *relative* rules $f_R(\mathbf{Y})$. Absolute rules are only used in the characterization of small-size clusters, and correspond to the presence of dominant features in the patterns of the relevant cluster. An absolute rule for feature j has the form $f_A(\mathbf{Y}) = \{\mathbf{y}_i \in \mathbf{Y} : x_i(j) == \lambda\}$. On the other hand, relative filtering rules depend on the relative separation between novel and previously seen patterns. Basically, if the patters are well separated from the rest in a certain partition P_i, then the features of the

Algorithm 1 Evidence Accumulation for Ranking Outliers (EA4RO)

Initialization:
  Set dissimilarity vector D to a null $n \times 1$ vector
  Set smallest cluster-size $n_{\text{min}} = \alpha \,.\, n$
for $i = 1 : N$ **do**
  Set density neighborhood δ_i for DBSCAN
  $P_i = \text{DBSCAN}\ (\mathbf{X}_i, \delta_i, n_{\text{min}})$
  Update $D(j), \forall$ outlier $o_j^i \in P_i$:
    $w_i \leftarrow \dfrac{n}{(n - n_{\text{max}_i}) + \epsilon}$
    $D(j) \leftarrow D(j) + \text{d}_\text{M}(o_j^i, C_{\text{max}}^i)\, w_i$
end for
Rank patterns: D_{rank} = sort(D)
Set detection threshold: T_h = find-slope-break(D_{rank})

corresponding sub-space $\mathbf{X}_i$ are good candidates to define a relative filtering rule. A relative rule defined for feature j has the form $f_R(\mathbf{Y}) = \{\mathbf{y}_i \in \mathbf{Y} : x_i(j) < \lambda \text{ or } x_i(j) > \lambda\}$. We shall also define a *covering relation* between filtering rules: we say that rule f_1 *covers* rule $f_2 \leftrightarrow f_2(\mathbf{Y}) \subset f_1(\mathbf{Y})$. If two or more rules overlap (i.e., they are associated to the same feature), the algorithm keeps the one that covers the rest.

In order to construct a compact signature describing the detected patterns, we have to devise a procedure to select the most discriminant filtering rules. Absolute rules are important, because they define inherent characteristics of the patterns. Regarding relatives rules, their relevance is directly tied to the degree of separation between patterns. In the case of outliers, we select the K features for which the normalized distance to the normal-operation patterns (statistically represented by the biggest cluster in each sub-space) is among the top-K biggest distances. In the case of small-size clusters, we rank the degree of separation to the rest of the clusters using the well-known Fisher Score (FS) [20], and select the top-K ranked rules. The FS basically measures the separation between clusters, relative to the total variance within each cluster. To finally construct the signature, the absolute rules and the top-K relative rules are combined into a single inclusive predicate, using the covering relation in case of overlapping rules.

4.1.4 Ranking Outliers using Evidence Accumulation

The previously described approach can also be adapted to exclusively focus on the identification of outliers. For doing so, we implement a particular algorithm for Evidence Accumulation, called Evidence Accumulation for Ranking Outliers (EA4RO): instead of producing a similarity measure between the n different patterns described in $\mathbf{X}$, EA4RO constructs a dissimilarity vector $D \in R^n$ in which it accumulates the distance between the different outliers o_j^i found in each sub-space $i = 1, .., N$ and the centroid of the corresponding sub-space-biggest-cluster C_{max}^i. The idea is to clearly highlight those patterns that are far from the normal-operation patterns at each of the different sub-spaces, statistically represented by C_{max}^i.

Algorithm 1 presents a pseudo-code for EA4RO. The different parameters used by EA4RO are automatically set by the algorithm itself. The first two parameters are used by the density-based clustering algorithm: n_{min} specifies the minimum number of patterns that can be classified as a cluster,

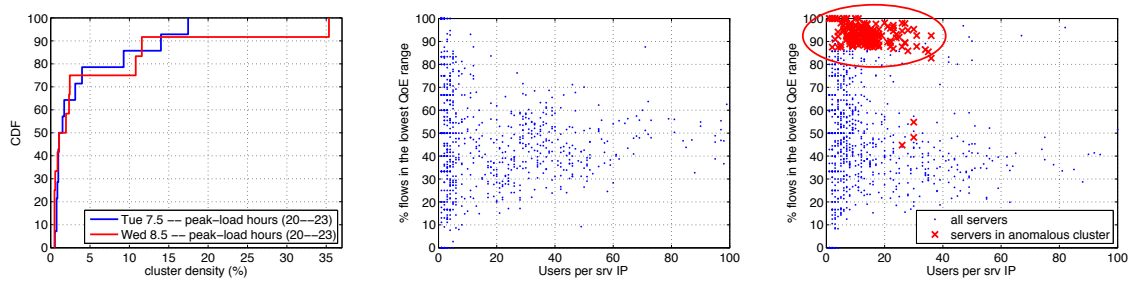
while δ_i indicates the maximum neighborhood distance of a pattern to identify dense regions. n_{min} is set at the initialization of the algorithm, simply as a fraction α of the total number of patterns n to analyze. δ_i is set as a fraction of the average distance between patterns in sub-space $\mathbf{X}_i$, which is estimated from 10% of the patterns, randomly selected. This permits to fast-up computations. The weighting factor w_i is used as an outlier-boosting parameter, as it gives more relevance to those outliers that are ``less probable'': w_i takes bigger values when the size n_{max_i} of cluster C^i_{max} is closer to the total number of patterns n. Finally, instead of using a simple Euclidean distance as a measure of dissimilarity, we compute the Mahalanobis distance d_M between outliers and the centroid of the biggest cluster. The Mahalanobis distance takes into account the correlation between patterns, dividing the standard Euclidean distance by the variance of the patterns. This permits to boost the degree of dissimilarity of an outlier when the variance of the samples is smaller.

In the last part of EA4RO, patterns are ranked according to the dissimilarity obtained in D, and the detection threshold T_h is set. The computation of T_h is simply achieved by finding the value for which the slope of the sorted dissimilarity values in D_{rank} presents a major change. The detection is finally done as a binary thresholding operation on D: if $D(i) > T_h$, then pattern $\mathbf{y}_i$ is selected as relevant.

4.1.5 An Example of Unsupervised Analysis

We present now a brief example describing the unsupervised analysis technique, in the specific case of detecting and diagnosing QoE-based anomalies in the YouTube service. The use case corresponds to the detection and diagnosis of an event of performance degradation in the QoE of a large number of users watching YouTube videos. The idea is to detect the occurrence of such events by tracking the evolution of the structure of the traffic, constructed through the presented multi-clustering algorithm. In this example, a pattern corresponds to all the traffic provided by a YouTube server during a specific period of time. Each pattern, i.e., each YouTube server, is characterized by a set of 9 features, including the number of flows, number of bytes, number of users, the median of the download throughput per flow, the entropy of the QoE classes (the definition of the QoE classes comes from the previously presented use case on Anomaly Detection), fraction of flows in the lowest QoE class, median of the min RTT, median of the average RTT, and median of the elaboration time, all of them computed in a temporal basis, i.e., per hour.

Fig. 59(a) depicts the distribution of the density of the clusters obtained with the presented approach (the density is measured in terms of fraction of TouTube server IPs contained in the cluster) identified during the peak-load hours, on a day previous to the occurrence of one if such QoE-based anomalies and during the day of the anomaly. There is a clear shift in the cluster density during the hours of the anomaly, revealing the appearance of a new cluster, containing about 35% of the YouTube servers. As presented in Figs. 59(b) and 59(c), the newly observed cluster corresponds to a set of server IPs providing a large share of YouTbe flows with low QoE, impacting a potentially large number of users. The interesting observation is that this set of server IPs can be identified by the multi-clustering approach, making it possible to detect the studied low QoE event in a completely unsupervised manner.



(a) Distribution of clusters density. (b) % bad QoE flows vs. # users, before anomaly. (c) % bad QoE flows vs. # users, before anomaly.

Figure 59: Unsupervised detection of the anomaly through clustering. There is a clear shift in the cluster density during the hours of the anomaly.

4.2 Association Rule Mining for network data analysis

The automatic analysis of huge network traffic data is a challenging and promising task. Association rule mining is an exploratory data analysis method able to discover interesting and hidden correlations among data, and it has been successfully applied to network traffic data in the context of the mPlane project. The challenge is twofold: (i) this data mining process is characterized by computationally intensive tasks, thus requiring efficient distributed approaches to increase its scalability, and (ii) its results must add value to the domain expert knowledge.

This section describes a cloud-based approach, named SEARUM, to efficiently mine association rules on a distributed computing model. SEARUM has been applied to mPlane network traffic data and consists of a series of distributed MapReduce jobs run in the cloud. Each job performs a different step in the association rule mining process.

Extracted rules can be exploited by the Reasoner to enrich the domain knowledge and to model the traffic behavior. The Reasoner could also benefit from extracted rules to better support network administration staff in drilling down root causes of anomalies and understanding traffic patterns.

4.2.1 Introduction

Association rule mining is a two-step process: (i) Frequent itemset extraction and (ii) association rule generation from frequent itemsets [1]. 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 [49, 48, 31, 16] and with a distributed architecture [35, 10, 29, 50]. However, when a large set of frequent itemsets is extracted, the generation of association rules from this set becomes a critical task.

The analysis of the large amount of Internet traffic data is an important task since a huge amount of interesting knowledge can be automatically mined to effectively support both service providers and Internet applications. To profile network communications, we analyzed traffic metrics and statistical measurements computed on traffic flows. The results showed the effectiveness and efficiency of the SEARUM architecture in mining interesting patterns on a distributed computing model.

4.2.2 Problem statement

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 $\mathrm{c}(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$ [33].

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.

Furthermore, to rank the most interesting rules, we used the lift index [33], which measures the (symmetric) correlation between antecedent and consequent of the extracted rules. The lift of an association rule $X \rightarrow Y$ is defined as [33]

$$\mathrm{lift(X,Y)} = \frac{\mathrm{c}(X \rightarrow Y)}{\mathrm{s(Y)}} = \frac{\mathrm{s}(X \rightarrow Y)}{\mathrm{s(X)s(Y)}} \tag{4.1}$$

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. In this work the mined rules are ranked according to their lift value to focus on the subset of most (positively or negatively) correlated rules.

4.2.3 Architecture

SEARUM 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 association rule mining. Currently, each job is performed by one or more MapReduce tasks run on a Hadoop cluster.

The SEARUM architecture contains the following jobs, described in details in the subsequent sections:

- Network measurement acquisition
- Data pre-processing
- Item frequency computation

- Itemset mining
- Rule extraction
- Rule aggregation and sorting

4.2.3.1 Network measurement acquisition

The first step to analyse network traffic is collecting network measurements. To this aim, we epxloited a passive probe running Tstat [12, 32], a passive monitoring tool developed in the context of the mPlane project. Tstat rebuilds each TCP connection by matching incoming and outgoing segments. Thus, a flow-level analysis can be performed [32]. A TCP flow is identified by snooping the signaling flags (SYN, FIN, RST). The status of the TCP sender is rebuilt by matching sequence numbers on data segments with the corresponding acknowledgement (ACK) numbers.

To evaluate the SEARUM cloud-based service in a real-world application, we focus on a subset of measurements describing the traffic flow among the many provided by Tstat. The most meaningful features, selected with the support of domain experts, 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[1]
- 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[2]
- 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, $P\{reord\}$, $P\{dup\}$, $NumPkt$, $DataPkt$, $DataBytes$, $WinMax$, $WinMin$, $WinScale$, $Port$, $Class$. To obtain reliable estimates on reordering and duplicate probabilities, only TCP flows which last more than $P = 10$ packets are considered. This choice allow focusing the analysis on long-lived flows, where the network path has a more relevant impact, thus providing more valuable information.

[1] The initial TTL value is set by the source, typical values being 32, 64, 128 and 255.

[2] $P\{reord\}$ and $P\{dup\}$ are computed by observing the TCP sequence and acknowledgement numbers carried by segments of a given flow.

4.2.3.2 Data pre-processing

This step performs the following two activities:

- Value discretization
- Transactional format conversion

Associaton rule mining requires a transactional dataset of categorical values. The discretization step converts continuously valued measurements into categorical bins. Then, data are converted from the tabular to the transactional format. An example is reported in Table 10.

Automatic discretization approaches can exploit state-of-the-art techniques (e.g., clustering, statistical-based algorithms, etc.) to select appropriate bins depending on data distribution. These approaches yielded poorly significant bins on network data considered in this study. More specifically, the most frequent values were split into too many bins with respect to the real applicative interest. Hence, discretized bins are fixed-size and determined by domain experts based on the significance in the networking context. The fixed-size bins have been determined as follows:

- RTT: a bin each 5 ms for values from 0 ms to 200 ms, an additional bin for values higher than 200 ms.
- Hop: a bin for each value from 1 to 20, an additional bin for values exceeding 20.
- $P\{reord\}$: a bin each 0.1 from 0 to 1.
- $P\{dup\}$: a bin each 0.1 from 0 to 1.
- $NumPkt$, $DataPkt$, and $DataBytes$: logarithmic bins, base 10, e.g., 5432 falls in the 3-4 bin since the value is between 10^3 and 10^4.
- $WinMax$ and $WinMin$: a bin for each multiple N of 4 Kb, where N is a power of 2, e.g., the bin 8-16 means that the TCP window is between 8 and 16 times 4 Kb.
- $WinScale$, $Port$, and $Class$: a bin for each value (no discretization).

Both the value discretization and the transactional format conversion are performed by a single map only job. Each record is processed by the map function and, if the number of packets is above the threshold (10 packets), the corresponding discretized transaction is emitted as a result of the mapping. This task entails an inherently parallel elaboration, considering that can be applied independently to each record.

	RTT	$NumPkt$	$P\{reord\}$
original	7	5432	0.88
discretized	5-10	3-4	0.9
transactional	RTT=5-10	$NumPkt$=3-4	$P\{reord\}$=0.9

Table 10: Pre-processing example

4.2.3.3 Item frequency computation

A second job is exploited to compute the item frequency from the transactions emitted by the pre-processing phase. An example is reported in Tables 11 and 12. Table 11 has three sample transactions that represent a possible output of the pre-processing phase. A map function is exploited to process each transaction: the map emits 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. A reduce 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 MapReduce job. As a running example, we will consider the sample result of this job reported in Table 12, as obtained by the sample transactions in Table 11.

transaction 1	RTT=5-10	$NumPkt$=3-4	Hop=10
transaction 2	RTT=5-10		Hop=11
transaction 3	RTT=5-10	$NumPkt$=3-4	
transaction 3	RTT=5-10	$NumPkt$=3-4	Hop=11

Table 11: Sample transactions

item	**sup count**	**sup**
RTT=5-10	4	100%
$NumPkt$=3-4	3	75%
Hop=10	1	25%
Hop=11	2	50%

Table 12: Sample items

4.2.3.4 Itemset mining

A third job performs the itemset mining by exploiting the parallel FP-growth algorithm. This step consists of multiple MapReduce tasks. From the sample items of Table 12, a result of this job is reported in Table 13, where only itemsets with support higher than 50% have been extracted.

ID	**itemset**			**sup count**	**sup**
1	RTT=5-10			4	100%
2	RTT=5-10	$NumPkt$=3-4		3	75%
3	RTT=5-10		Hop=11	2	50%
4		$NumPkt$=3-4		3	75%
5			Hop=11	2	50%

Table 13: Sample itemsets

4.2.3.5 Rule extraction

The rule extraction step consists of a MapReduce job, as detailed in the following. For each itemset of length k (k-itemset), the map function emits:

- a (key,$value$) pair with
  - key: the k-itemset itself
  - $value$: the k-itemset support count
- for each $(k-1)$-itemset, a (key,$value$) pair with
  - key the $(k-1)$-itemset
  - $value$ the pair (k-itemset, support count of the k-itemset).

Then, the reduce function performs the actual rule extraction. Since each $(k-1)$-itemset emitted as key contains its k-itemset and the k-itemset support count as value, the missing item in the $(k-1)$-itemset with respect to the k-itemset is the rule consequent (head), whereas the $(k-1)$-itemset is the antecedent (rule body). The support count values of the k-itemset, the $(k-1)$-itemset and the consequent item are used to compute the support, confidence, and lift of the rule, as defined in Section 4.2.2. Table 14 reports the rules extracted from the itemsets of the running example (see Table 13).

rule	sup count	sup	conf	lift
RTT=5-10 $\rightarrow$ $NumPkt$=3-4	3	75%	75%	0.75
$NumPkt$=3-4 $\rightarrow$ RTT=5-10	3	75%	100%	1.33
RTT=5-10 $\rightarrow$ Hop=11	2	50%	50%	0.50
Hop=11 $\rightarrow$ RTT=5-10	2	50%	100%	2.00

Table 14: Sample rules

4.2.3.6 Rule aggregation and sorting

A final step is executed by means of a MapReduce job to sort and aggregate the rules according to the consequent and the quality measure. As discussed in Section 4.2.2, we selected the lift as rule quality measure. Sorting and aggregating on the consequent helps in analyzing the extracted rules for finding significant correlations. A sample output based on our running example is reported in Table 15.

antecedent		consequent
Hop=11, $NumPkt$=3-4	$\rightarrow$	RTT=5-10
RTT=5-10	$\rightarrow$	$NumPkt$=3-4
RTT=5-10	$\rightarrow$	Hop=11

Table 15: Sample rules, sorted and aggregated

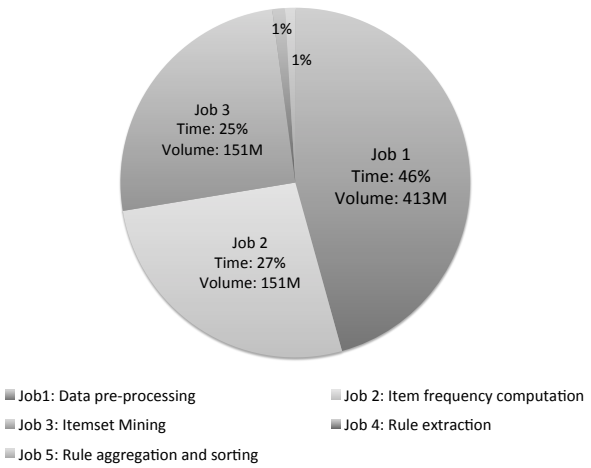


Figure 60: Dataset D2: Execution time distribution among jobs for MinSup=30% and MinConf=50%

4.2.4 Experimental evaluation

A set of preliminary experiments have been performed analyzing SEARUM behavior on real mPlane datasets. We assessed (i) the percentage of the overall mining time devoted to performing each job (Section 4.2.4.1), (ii) the performance of the association rule mining (Section 4.2.4.2), (iii) the network knowledge characterization (Section 4.2.4.3) and (iv) the number of extracted association rules by varying the support and confidence thresholds (Section 4.2.4.4),

SEARUM has been applied to two real datasets. We will refer to each dataset as D1 or D2 as shown in Table 16, where the number of TCP flows and the size of each dataset are also reported.

Dataset	Number of TCP flows	Size [Gbyte]
D1	11,325,006	5.28
D2	413,012,989	192.56

Table 16: Network traffic datasets

MapReduce jobs of the SEARUM workflow (see Section 4.2.3) have been developed in Java using the Hadoop Java API. Part of the code has been developed as an extension of the Apache Mahout project, [36], which provides a limited implementation of the parallel itemset mining algorithm FP-Growth to mine the top-k closed itemsets [30].

Experiments have been performed on a cluster of 5 nodes running Cloudera's Distribution of Apache Hadoop (CDH4). Each cluster node is a 2.67 GHz six-core Intel(R) Xeon(R) X5650 machine with 32 Gbyte of main memory running Ubuntu 12.04 server with the 3.5.0-23-generic kernel. All reported execution times are real times, including both system and user time, obtained from the Cloudera Hadoop web administration control panel.

4.2.4.1 Execution time distribution among jobs

Since SEARUM consists of a sequential workflow, we analyzed how much time is spent at each step. Fig. 60 shows the percentage of the total execution time for each job of the SEARUM architecture.

The pre-processing job represents the most expensive step with almost 50% of the time. This behavior is due to the higher data volume to be processed at this stage with respect to the subsequent steps: pre-processing filters flows with less than 10 packets, thus reducing the data volume from 413 millions of records to 151 millions of transactions. Note that the pre-processing job is executed only once for each discretization bin set, while it provides results that can be used many times by subsequent jobs, e.g., to mine itemsets and rules with different $MinSup$ and $MinConf$ constraints.

When the $MinSup$ value decreases, the computational complexity of the itemset mining job may significantly increase, since a very large number of itemsets may be generated. As a reference, Fig. 60 reports times for MinSup=30% and MinConf=50% on dataset D2.

4.2.4.2 Evaluation of association rule mining

To evaluate the effectiveness of SEARUM in association rule mining, we measured the achieved speedup for different numbers of nodes. We considered 3 configurations: 1 node, 3 nodes, and 5 nodes. Dataset D2 has been used since it is the largest. Fig. 61 reports the speedup when $MinSup = 30\%$ and $MinConf = 50\%$ are applied.

The first histogram in the Fig. (i.e., 1 node) corresponds to an execution of SEARUM on a single node. The speedup evaluation for increasing numbers of nodes is compared to the single-node performance.

The achieved speedup is the result of job distribution. The contribution of the former is especially relevant when considering large datasets and/or low minimum support thresholds ($MinSup$), which make the mining activity more computationally intensive. As reported in Fig. 61 the SEARUM performance progressively improves when distributing the mining task on an increasing number of nodes. For instance, a 5-node cluster achieves a speedup of 4.5, showing that SEARUM has promising attitude to scale in larger clouds.

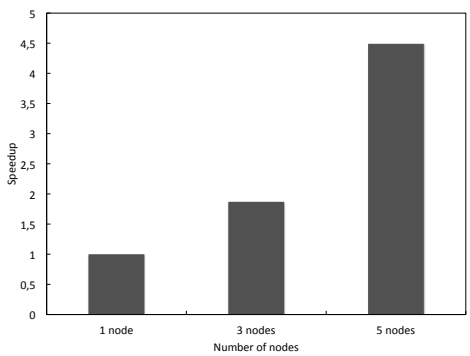
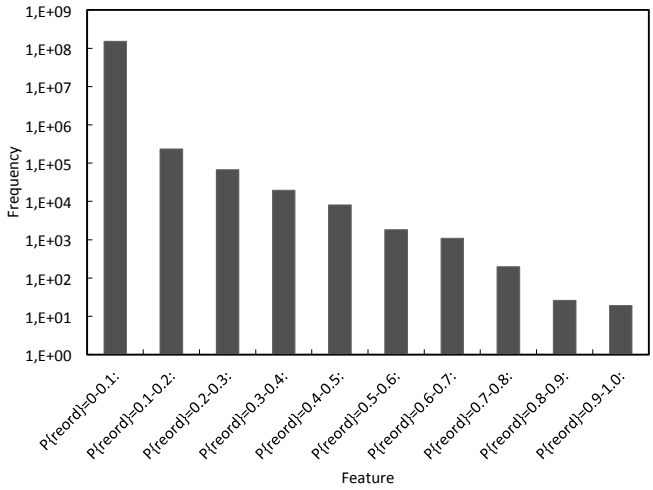
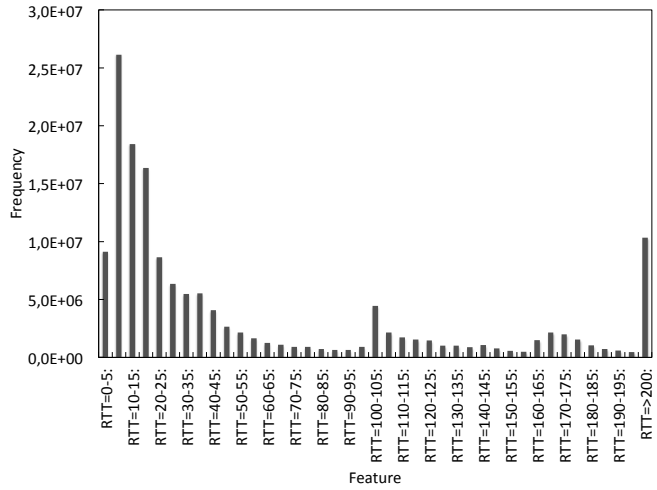


Figure 61: SEARUM speedup on D2 dataset

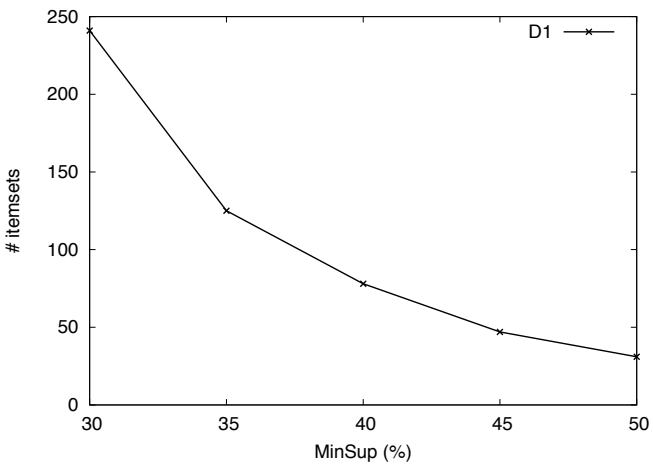


(a) Item distribution for the $P\{reord\}$ feature

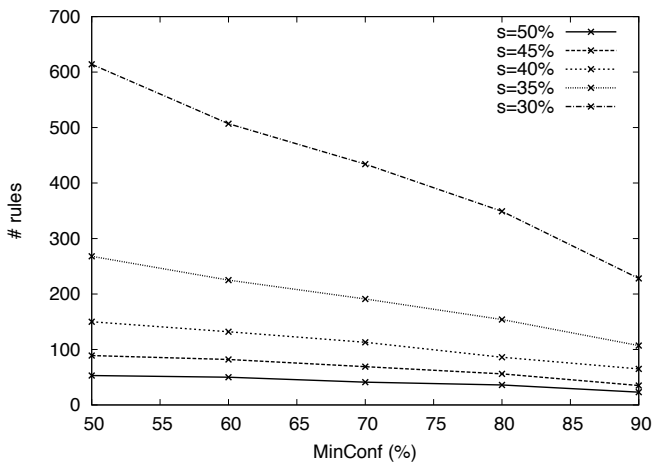


(b) Item distribution for the RTT feature

Figure 62: Dataset D2



(a) Number of extracted itemsets for different $MinSup$ values



(b) Number of extracted rules for different values of $MinConf$ and $MinSup$

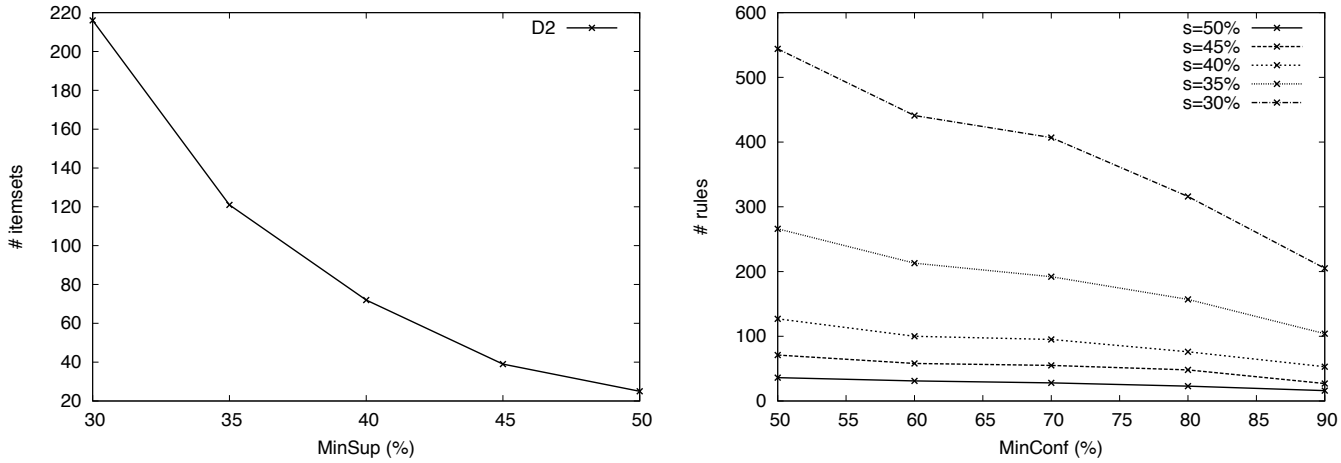
Figure 63: Dataset D1: Effect of $MinSup$ and $MinConf$ thresholds

4.2.4.3 Network knowledge characterization

We evaluated the effectiveness of the proposed approach on real network traffic traces. In particular, we analyzed: (i) the usefulness of the extracted association rules in supporting the knowledge discovery process, and (ii) the item frequency distribution.

As example, the following two rules R1 and R2 are generated from dataset D1 and D2, respectively. Both rule have high confidence values and lift greater than 1 (rule support, confidence, and lift are reported in brackets after each rule).

$R_1 : \{Port = 80, P\{reord\} = 0 - 0.1, DataPkt = 1 - 2, DataBytes = 4 - 5\} \rightarrow Class = HTTP$ (0.313, 0.999, 1.765)



(a) Number of extracted itemsets for different values of $MinSup$ (b) Number of extracted rules for different values of $MinConf$ and $MinSup$

Figure 64: Dataset D2: Effect of $MinSup$ and $MinConf$ thresholds

$R_2 : \{P\{dup\} = 0 - 0.1, NumPkt \leq 1, DataPkt \leq 1, Class = SSL\} \rightarrow Port = 443$ (0.013, 0.993, 4.944)

Based on rule R_1, the HTTP protocol is mainly used to transmit a set of TCP flows sent by the server through the TPC port 80. For these flows, the number of packets is in the range 10÷100 and a large number of bytes is transmitted (from 10,000 to 100,000). These flows can be generated when very large files are downloaded (e.g., YouTube videos).

Rule R_2 reports that the TCP $Port$ 443 (HTTPS) is mainly used to transmit flows with SSL/TLS coded protocol and less than 10 packets. These flows can be generated when logging into websites through a secure connection (e.g., Facebook, Twitter).

We also analyzed the item frequency distribution to characterize the network activity. Fig. 62 considers the Round-Trip-Time (RTT) and the flow reordering probability ($P\{reord\}$), which are discussed as representative features.

The item distribution for the $P\{reord\}$ feature is characterized by a very frequent item which models most TCP flows: they have a very low $P\{reord\}$, i.e., from 0 to 0.1. This data distribution analyzed over time and for different (sub)networks may be exploited to identify periods of time or (sub)networks that become less reliable or whose packets change path more frequently than usual.

The item distribution for the RTT, instead, shows four peaks:

- the first peak around 5-20 ms may represent local network traffic
- the second peak around 100 ms may represent external traffic inside the same ISP or in the same geographical zone (e.g., country, continent)
- the third peak around 170 ms may represent traffic towards long-distance destinations (e.g., other continents)
- finally, the last peak over 200 ms may represent network problems or unresponsive services

4.2.4.4 Effect of the support and confidence thresholds

Minimum support ($MinSup$) and confidence ($MinConf$) thresholds significantly affect the number of extracted itemsets and association rules.

When decreasing the $MinSup$ value, the number of frequent itemsets grows non linearly [1] and the complexity of the frequent itemset extraction task significantly increases. High $MinConf$ values represent a tighter constraint on rule selection. Consequently, when increasing $MinConf$ less rules are mined, but these rules tend to represent stronger correlations among data. High $MinConf$ values should be often combined with low $MinSup$ values to lead the extraction of peculiar (i.e., not very frequent) but highly correlated rules.

Figs. 63(a) and 64(a) plot, for the two reference datasets, the number of extracted itemsets when varying $MinSup$. Fig. 63(b) and 64(b) report the number of association rules for different $MinConf$ values.

4.3 Statistical Learning for Knowledge Discovery

One of the main question concerning the analysis of data collected by the mPlane to discover new knowledge is which technique should be considered in order to model the (hidden) spatio-temporal relationship $y = F(\bar{x})$ between the variables of the system (the input $\bar{x} = \{x_1, \cdots, x_n\}$) and the output y, and knowing this model, predict system response. The latter consists in producing a good predicting function $F(x)$ such that $\hat{y} = F(x)$ minimizes the loss function $L(y, \hat{y})$, e.g., the root mean squared error $\sqrt{E[y - \hat{y}]^2}$. The whole point of statistical learning (and generalization) is concerned with the problem of finding function F on the basis of empirical data while minimizing the empirical risk.

We first assume that such information can be progressively derived (inferred) from the statistical nature of the events experienced by the network (process referred to as learning from experience), as sufficient input data, i.e., observations, is obtained online from a set of observers or monitoring points. Many classes of statistical learning techniques have been developed for the purpose of realizing such learning objective (Linear Discriminant, Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), Hidden Markov Model (HMM), etc.) and each class comprises multiple variants. The techniques are often combined to allow first finding structures and features in input data without guidance about inputs and outputs, e.g., without probability distributions (unsupervised learning) and then estimate/predict the output for a novel (never-seen-before) input after learning on the training data set (supervised learning). Determining which technique would suit the learning objective at hand and underlying learning tasks (e.g., grouping, summarization/dimensionality reduction, association, estimation, classification, regression) first requires the examination of the input data properties on which the learning technique performs.

Most of the commonly envisaged statistical learning techniques assume propositional data being identically and independently distributed (i.i.d. assumption). This assumption implies that an element in the sequence is independent of the random variables that came before it, and random samples of homogeneous data objects result from a single relation. These common assumptions have to be contrasted with the intrinsic properties of real-world data sets, in particular, those characterizing records collected by distributed monitoring points. These data are not identically distributed (heterogeneous) and not independent (multi-relational structures). For instance, disruption of connectivity between related entities can result from a single AS relationship failure which induces different re-routing events (as multiple paths are often affected) leading to different recovery probability and non-identical recovery times. The attachment point of a server may be congested because of the occurrence of correlated events involving similar entities which try to access the same data objects with a probability proportional to their popularity. The same observation can be drawn when the learning task consists in modeling and predicting the joint bandwidth - delay product of a forwarding path. One sees behind these examples that out-of-the-shelf statistical learning techniques cannot effectively account for the intrinsic properties of the data characterizing these environments.

4.3.1 Markov Logic Network (MLN)

Filling this gap is the main purpose of Statistical Relational Learning (SRL) [36] which combines i) relational logic learning to model complex relational structures and inter-dependency properties in data with ii) probabilistic graphical models (such as Bayesian networks or Markov networks) to model the uncertainty on the data. The resulting process can perform robust and accurate learning

tasks out of multi-relational and inter-dependent data. In the context of communication networks, SRL is of particular interest when considering the objective of learning (hidden) dependencies between multi-relational, heterogeneous, and semi-structured data. The congestion (or concurrent access) prediction problem from the similarity of spatio-temporal relationships between entities belonging to the same community pattern provides a representative example of such learning problem. The motivations stem from the fact that the models learned from both intrinsic (propositional) and relational information perform better than those learned from intrinsic information alone. These models offer also better (predictive) accuracy, robustness, and understanding of the relational structures when processing heterogeneous and/or (inter-)dependent data sets. However, this learning technique induces a harder learning task and higher complexity.

Statistical Relational Learning (SRL) combines probabilistic graphical models (probabilistic learning and inference) to model and reason about uncertainty with representation language to describe relational properties of the data and complex dependencies between them (logical learning and inference). Graphical models provide a principled approach to deal with uncertainty and relational data by means of probability theory. These models represent dependency structure between random variables by joint distributions. Two types of graphical models are commonly considered: Markov(ian) networks and Bayes(ian) networks. On the one hand, Markov networks, described by undirected graphs, where edges do not carry arrows (no acyclic constraint) and have no directional significance, are useful for expressing symmetric relationships (soft constraints) between random variables. On the other hand, Bayesian networks, represented by Directed Acyclic Graphs (DAG), where edges have a particular directionality indicated by the arrows (acyclic constraint), are useful for expressing causal relationships between random variables. In addition to the distinction between undirected and directed graphical models, the differentiation between main representation syntaxes, i.e., first-order logic vs. frame-based representation provides a complete categorization of the different SRL models. One distinguishes, as part of the directed models, between rule-based models Bayesian Logic Programs (BLP) [26] and frame-based models Probabilistic Relational Models (PRM) [15] and, as part of undirected models, between frame-based models Relational Markov Networks (RMN) [9] and rule-based models Markov Logic Networks (MLN) [39]. Selection of the MLN model stems from the following reasons: it suits control processes whose execution is causality-independent, it is more flexible when the data are made available sequentially (as it is the case when monitoring communication networks with continuous stream of input data), and it enables exploiting data sparseness (condition met when only partial data is available upon occurrence of failures).

4.3.2 incremental Markov Logic Network (iMLN)

MLN (as other statistical relational learning models) have been designed independently on the input data arrival process, i.e., the processing algorithm performs on complete data set (in batch mode). However, when performing online learning in communication networks, input data arrives following different temporal patterns (in sequential mode) and the model is to be updated as data arrives. For this purpose, we extend the MLN model, which represents a probability distribution over possible worlds, to cover incremental updates from arrival of input data.

An MLN is formally defined as a set of pairs of formulas F_i in first order logic and their corresponding weights w_i denoted $\{(F_i, w_i)\}$. In first-order logic, formulas are recursively built from atomic formulas (nodes of the Markov network). Each formula F_i has an associated weight w_i: the higher the weight value, the greater the difference in probability between a world that satisfies the for-

mula and one that does not, other things being equal. It is important to emphasize that an MLN becomes a Markov network only with respect to a specific grounding and interpretation. Indeed, atomic formulas do not have a truth value unless they are grounded and given an interpretation. Thus, one requires that each node represents a ground atom, i.e., an atomic formula all of whose argument terms contain no variables. A possible world along with its interpretation assigns a truth value to each possible ground atom: when a world violates one formula, it is less probable although not impossible. The fewer formulas a world violates, the more probable it is.

Together with a set of constants in the domain of discourse, an MLN defines a (ground) Markov network with i) one binary node for each possible grounding of each atomic formula or atom appearing in the MLN (the value of the node is 1 if the ground atom is true and 0, otherwise) and ii) one feature f_i for each grounding of each first-order logic formula F_i in the MLN with the corresponding weight w_i(the value of this feature is 1, if the grounding of the formula is true, 0 otherwise). Each state of the ground Markov network, represented as a log-linear model, presents a possible world x (i.e., assignment of truth values to all possible nodes or ground atoms). The probability distribution over possible worlds x specified by the ground Markov network probability is given by:

$$P(X = x) = \frac{1}{Z} exp(\sum_{i=1}^{n} w_i \, n_i(x)) \tag{4.2}$$

In equation 4.2, n denotes the number of formulas in the MLN, the denominator Z the partition function used to make the summation of all possible groundings adding up to 1, w_i the weight of the formula F_i, and $n_i(x)$ the number of true (satisfied) groundings for the formula F_i in x. We also operate under the closed world assumption, i.e., if a ground atom is absent in the data, it is assumed to be false.

4.3.2.1 Weight Learning

Besides MLN structure learning (not covered in this section), the main learning task consists in learning MLN weights. Assuming we have at our disposal a given set of formulas $\{F_1, F_2, \cdots, F_n\}$, the learning task consists in finding the respective weights $\{w_1, w_2, \cdots, w_n\}$. These weights can be learned generatively by maximizing the likelihood of one or more possible worlds that form training samples. To avoid requiring inference at each step, one can instead obtain the weights w_i from the pseudo-likelihood (PL) approximation of the joint probability distribution of a world x based on its Markov blanket. The Markov blanket (MB) of a node X is the minimal set of variables that must be observed to make this node independent of all other nodes in a model. In an undirected model such as a Markov network, the Markov blanket includes the node's neighbors in the graph. Note that the use of the pseudo-likelihood approximation does not require inference at each step and avoids the use of the partition function Z. It is indeed impractical to perform exact inference on large Markov models because of the computations on the partition function Z. If x is a possible world and x_k is the kth ground truth value, the PL approximation of x given weights w is provided by:

$$PL_w(X = x) = \prod_{k=1}^{n} P_w(X_k = x_k | MB(X_k)) \tag{4.3}$$

4.3.2.2 Inference Tasks

A basic inference task consists in finding the most probable state of the world given the evidence x, i.e., the world in which the sum of the weight of all satisfied groundings is maximized. For this purpose, given the evidence x, it suffices to compute the following [37]:

$$arg \max_y P(y|x) = arg \max_y \sum_i (w_i \, n_i(x, y)), \tag{4.4}$$

where $n_i(x, y)$ denotes the number of true groundings of formula F_i involving atoms y. Computation of equation (3) relies on a weighted SAT solver as corresponding from equation (1) to the weighted MaxSAT problem. In order to find a truth assignment that maximizes the sum of the weights of satisfied formulas, one can use (to avoid local optima while searching) the MaxWalkSAT solver [25], a weighted variant of the WalkSAT stochastic local-search (SLS) satisfiability solver. In order to predict the occurrence of most likely patterns given the observation of certain events (predictive inference problem), it suffices to compute given evidence x, the equation 4.4 by means of the MaxWalkSAT algorithm.

Another key inference task consists in computing the probability $P(F_i)$ that a given formula F_i holds, given an MLN and possibly a set of one or more formulas as evidence. As the probability of a formula is the sum of the probabilities of the worlds x where it holds ($P(F_i) = \sum_x P(X = x)$), the conditional probability is given by:

$$P(F_i|F_j) = \frac{P(F_i \wedge F_j)}{P(F_j)} = \frac{\sum_{x \in \Xi_i \cap \Xi_j} P(X = x)}{\sum_{x \in \Xi_j} P(X = x)}, \tag{4.5}$$

where Ξ_i (Ξ_j) is the set of worlds where F_i (F_j) holds and $P(X = x)$ is given by Eq.4.2. To avoid exponential time in the number of possible ground atoms, equation 4.5 can be approximated using probabilistic inference methods like Markov Chain Monte Carlo (MCMC). This algorithm samples a sequence of states according to their probabilities, counts the fraction of sampled states where the formula holds, and rejects any state that violates one of them (i.e., it rejects all moves to states where F_j does not hold, and counts the number of samples in which F_i holds). However, as MCMC breaks down when deterministic or near-deterministic dependencies are present, it is combined with satistifiability testing (by extending the WalkSAT solver) in the MC-SAT inference algorithm [18]. Using this procedure, it is possible to find for instance the probability that the formula F_i holds, knowing that the formulas (evidence) F_j and F_k do.

4.3.3 Application Example

Assuming user communities can be detected and identified based e.g. on similarity between users, this information could be used to provide input to the concurrent access detection problem. As MLN enables to compactly represent the dependencies between data and relations (compared to the approaches that process them independently) together with collective inference; the objective is to find a more accurate predictive model about possible occurrences of concurrent access of the same data objects by users (a.k.a. friends) belonging to the same community.

In the simplest instance of this problem, if two elements x_1 and x_2 are friends $Friends(x_1, x_2)$ they share the same interests implying that if x_1 access data object y ($Get(x_1, y)$), its friend would most likely execute the same action ($Get(x_2, y)$).

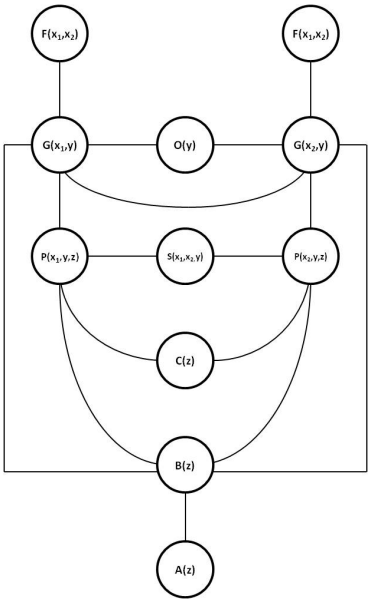


Figure 65: Ground Markov Network

If the path used by x_1 to fetch/access this content y comprises a component z that intersects the path used by x_2 to retrieve the same content, one may suspect existence of at least one component z more loaded than the others.

Assuming we have at our disposal the following set of formulas F_1, F_2, F_3, F_4, F_5:

Weight(w)	**Formula**
w_1	$F_1 : \forall z\ Busy(z) \Rightarrow Alarm(z)$
w_2	$F_2 : \forall x,\ \exists y, z\ Get(x, y) \wedge Path(x, y, z) \Rightarrow Busy(z)$
w_3	$F_3 : \forall x_1, x_2,\ \exists y, z,\ \ Path(x_1, y, z) \wedge Path(x_2, y, z) \Rightarrow Shared(x_1, x_2, y)$
w_4	$F_4 : \forall x, \exists y, z,\ \ Get(x, y) \wedge Comp(z) \Rightarrow Path(x, y, z)$
w_5	$F_5 : \forall x_1, x_2,\ \exists y\ Friends(x_1, x_2) \wedge Object(y) \Rightarrow (Get(x_1, y) \Leftrightarrow Get(x_2, y))$

By learning the weights, the purpose is to derive the probability the formula is satisfied from the training sample(s).

The ground Markov network corresponding to this MLN is depicted in Fig. 65, where A stands for Alarm, B for Busy (Component), C for Component, F for Friends, G for Get, and P for Path. In practice, user communities can be quite large and number of data objects even larger. Extending the model to larger number of elements $x_1, \cdots, x_n$ and knowing path not limited to single flow, enables to further elaborate the model and consider that at least k elements have to access this object before reaching a certain load.

The decomposition of the ground Markov network into sub-networks is represented in Fig. 66. The dashed entities represent the elements detected by one of the monitoring agents and the dotted entities those associated to the other. In the same figure, the gray vertices represent the shared atoms between sub-networks. The important characteristic underlying this (ground) Markov network is the following: assuming the ground Markov network is learned incrementally from the information detected by individual monitors (each monitor being associated to one sub-network), a set of relationships has to be established where the size of this set reflects the number of ground atoms

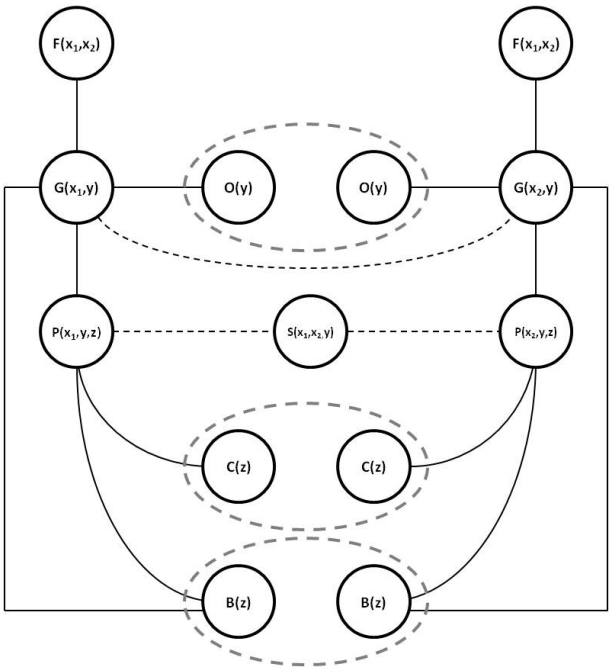


Figure 66: Ground Markov Network Decomposition

interconnecting these sub-networks.

Further, we would like to predict the probability of occurrence of certain busy component patterns from the detection of some relationship between community entities. To solve this predictive inference problem, it suffices to compute, using a set of formulas as evidence, the equation 4.5 by means of the MC-SAT algorithm. Using this procedure, the MLN model is able to determine for instance the probability that the formula $\forall x,\ \exists y, z\ Get(x, y) \wedge Path(x, y, z) \Rightarrow Busy(z)$ holds given that the formula $\forall x, \exists y, z,\ Get(x, y) \wedge Comp(z) \Rightarrow Path(x, y, z)$ and $\forall x_1, x_2,\ \exists y\ Friends(x_1, x_2) \wedge Object(y) \Rightarrow (Get(x_1, y) \Leftrightarrow Get(x_2, y))$ do.

Now, assuming that $\exists y$ such that $Friends(x_1, x_2) \wedge Object(y) \Rightarrow (Get(x_1, y) \Leftrightarrow Get(x_2, y))$, $Friends(x_2, x_3) \wedge Object(y) \Rightarrow (Get(x_2, y) \Leftrightarrow Get(x_3, y))$, and $Friends(x_3, x_1) \wedge Object(y) \Rightarrow (Get(x_3, y) \Leftrightarrow Get(x_1, y))$ hold, we are interested in determining the probability of the formula $Get(x_1, y) \wedge Path(x_1, y, z) \Rightarrow Busy(z)$ holds given that the formula $Get(x_2, y) \wedge Comp(z) \Rightarrow Path(x_2, y, z)$ and $Get(x_2, y) \wedge Comp(z) \Rightarrow Path(x_2, y, z)$ hold.

Simulation results obtained by running the MLN model for a predictive inference task confirm the inherent problem of decomposing a learning method originally designed to perform on data presenting (hidden) correlations but without accounting for their spatial distribution and sequential arrival. This observation leads to consider the MLN model decomposition in which the corresponding shared atoms can themselves correspond to sub-networks so as to improve the relationship creation process between atoms. The latter is of major importance to support scaling with respect to the community size scale, number of friend relationships, number of data objects, etc. and in turn to the number of information sources/monitoring points. An important challenge thus consists in determining the best achievable tradeoff between learning performance (accuracy, sensitivity, specificity, etc.), relationship creation cost, and computational complexity of the SRL methods (in particular, when applied to predictive tasks).

4.4 Approaches for Data Correlation

One of the key features of the Reasoner is the ability to correlate data coming from multiple mPlane components, e.g., probes and repositories, to effectively drill down to the root cause of a problem or to optimize the network usage with respect to the different use cases that the project plans to address.

Several use cases take advantage of supervised approaches to achieve their goal and require to combine offline data, such as the models gathered during a training phase, with online data, such as real-time information about the connections as collected by the mPlane probes, or the topological information of the network. The importance of collecting offline data to perform training serves two purposes. First, it is useful for bootstrapping the analysis module and being able to feed the reasoner with input right from the beginning. Second, it allows to collect historical data, which may be exploited to update the models themselves over time, in an on-line learning fashion.

For example, in the use case ``Supporting DaaS troubleshooting'', models of (classes of) applications of thin-client connections represent the historical (offline) data kept in the repository; live measurements of thin-client connections taken by probes, such as packet sizes and inter-arrival times, represent instead the online data that the analysis module combines with the offline models to match the thin-client connection to a particular application. By correlating topological and delay information, e.g., collected by mPlane probes running tracebox, with the above information, the Reasoner can identify the node responsible for the bottleneck (i.e., the poor users' QoE) along the path, and act accordingly to overcome the issue: for instance, by migrating the remote server across datacenters. Decision-tree tools can drive such decisions, and some of them are described in the workflows of the specific use cases in Chapter 3.

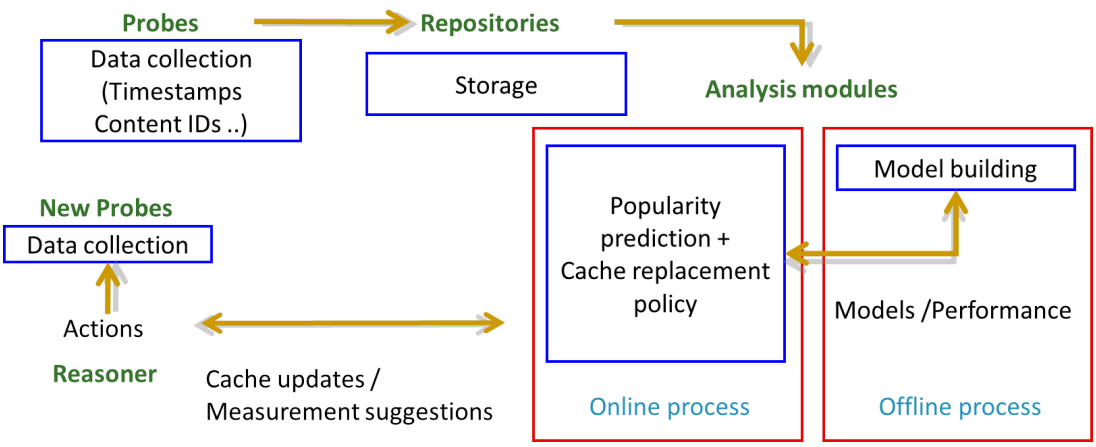


Figure 67: Correlating offline and online data in the use case "Estimating content and service popularity for network optimization".

Data correlation is useful for gaining a better picture of the status of the network where a given use case is deployed, improving its troubleshooting, and optimizing network resources' usage. Moreover, it allows the coexistence of several specialized probes, designed around a specific task, and combine the outcome of all of them.

In the use case ``Estimating content and service popularity for network optimization'', models of content popularity evolution represent the historical (offline) data kept in the repository; the evolution of views for contents being seen by vantage points represents instead the online data that the

analysis module combines with the offline models for determining to which model the popularity evolution of a content belongs to, based on which the Reasoner suggests caching strategies.

Fig. 67 outlines the use of offline and online data in the context of this use case. In this scenario, a smart use of other source of data could actually improve the overall caching strategy. In fact, given that the Reasoner takes as input also the topological position of the caches for which it computes the list of upcoming popular objects, those list of objects could be clustered and their placement further optimized: for instance, knowing that some caches are organized according to a hierarchical scheme, we can include objects which are popular in multiple areas served by different caches into an upper-layer cache. Hierarchical clustering approaches could be exploited for this purposes.

4.5 Considerations for Building Diagnosis Graphs

In this section, we develop considerations that are especially important when building diagnosis graphs that are, at least in part, based on active measurements.

We give both general insights, as well as experimental findings to corroborate our reasoning, that we describe using the formalism exemplified in Fig. 68, where diagnosis actions are arranged as a tree. Depth in the tree represents a loose temporal sequence, so that items at the same depth are actions happening in parallel. (e.g., actions B and E, or any in $\{C, D, G, H, I\}$).

Branches can be either conditionally or systematically followed: for instance, irrespectively of results of action A, the system launches actions E and B (the former is faster than the latter, so their execution is not necessarily happening or completed at the same time). Similarly, C and D are launched after completion of task B. Conversely, depending on the result of action F, either action G (when $F = -1$) or H (when $F = 0$) or I (when $F = 1$) is launched.

This simplistic formalism allows to both:

- Encode drill-down methods where chains of actions are followed depending on values of their predecessor actions, so that one single path of the tree is followed from the root to the leaves (i.e., in case all edges have conditions depending on results at the previous node)

- Encode sequences of measurements, with a variable degree of parallelism and loose scheduling properties, that ultimately allow to gather a set of measurement features (namely, in the example of Fig. 68 the feature vector will be constituted by all the A, B, C, D, E, F measurements and one of the {G,H,I}) over which data mining or big data approaches can be applied.

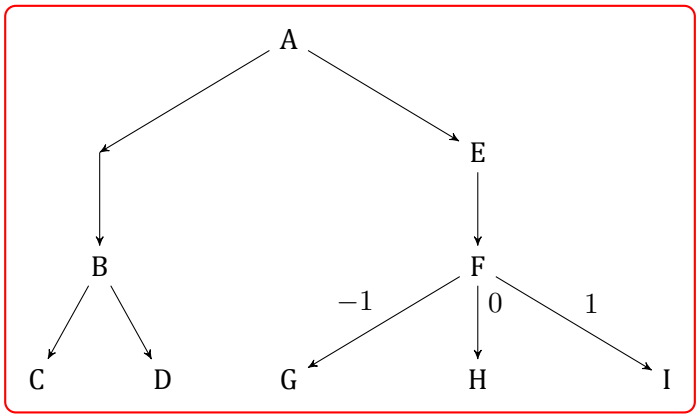


Figure 68: Example of diagnosis and measurement scheduling tree

With the above formalism, we now list challenges that can be faced by the mPlane supervisors. We also give guidelines on how to avoid them, or propose solutions whenever possible.

4.5.1 Scheduling tradeoff

We assume for simplicity that measurements have no conditional execution[3], and represent three scheduling strategies as shown in Fig. 69.

At the extreme left, all measurements are scheduled in sequence. The main drawback of this scheduling strategy is the long execution time: as the measurement conditions evolve over time, diagnosis decisions are possibly taken over measurements that represent a different behaviour of the network service to be measured, weakening the correlation between measurements.

At the extreme right, all measurements are scheduled in parallel, which guarantees the shortest execution time, but possibly raises problems due to possible interference of multiple measurements happening in parallel. Notice that while the figure represents a single tree, this tree will possibly be instantiated per user. Therefore, the number of simultaneous measurements due to the fanout of the per-user tree has to be scaled up by the scale of the measurement campaign.

The intermediate scenario represents a tradeoff between the long duration of sequential scheduling and the possible interference of parallel scheduling. Remapping a sequential or a parallel tree to an intermediate one is not easy in the general case; yet, scheduling implications have a possibly determinant effect on the net result of the outcome of the reasoner algorithm, and we discuss them further in the remainder of the section.

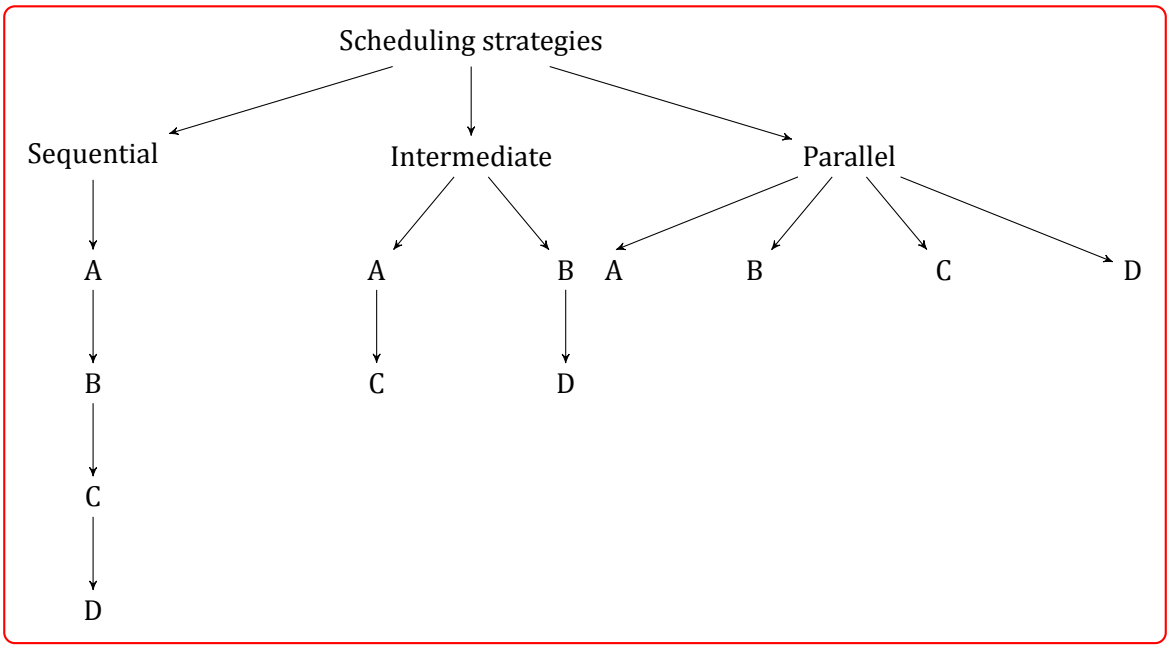


Figure 69: Scheduling tradeoff

[3] In case measurements have conditional execution, considerations developed in this section still hold, but this would unnecessarily lead to a more complex notation. Indeed, a generic measurement label (e.g., X) can be thought of encoding a more complex scenario depending on previous steps (e.g., $X = F = -1?G : F = 0?H : F = 1?I : Exception$ where conditions reported in Fig. 68 are encoded with the ternary operator $condition?ifclause : elseclause$).

4.5.2 Implications of temporal properties and guidelines

In machine learning terms, the vector of measurements (A, B, C, D) represents a scenario where all measurements are taken at possibly different times $(A(t_A), B(t_B), C(t_C), D(t_D))$. where t_A, t_B, t_C, t_D represent the measurement start time (that is $t = t_A = t_B = t_C = t_D$ only for parallel scheduling in Fig. 69).

Clearly, in the case of sequential scheduling, the network conditions can potentially change during the measurement period (or the phenomena causing performance degradations can possibly vanish, depending on the length of the chain). This also implies that correlation between any pair of measurements $X, Y \in A, B, C, D$ may weaken making the detection problem harder: if $X(t)$ and $Y(t)$ are correlated at time t, it does not mean that they are necessarily correlated at times t_X and t_Y.

This becomes especially problematic if measurement A, B, C, D are carried from different probes, implying that in mPlane terms, a new HTTPS connection has to be established, and a TLS handshake performed. To reduce unnecessary delay $\|t_X - t_Y\|$ between any pair of measurement s$X, Y \in A, B, C, D$, and of the whole measurement chain, it would be desirable to opportunistically establish connections in parallel at the root of the tree, and then sequentially schedule measurements over these established connections. Consequently, the time delay between a pair of consecutive measurements would be bound to the duration of the measurement itself (which is generally either known, deterministic, and tunable, or can be accurately statistically bound). Possibly, some proactively opened connections might not be used in practice later on, if trees encode conditional execution.

4.5.3 Interaction of homogeneous measurements

When measuring any given metric of interest, *precision* and *accuracy* are intrinsic to each tool, and can limit the usefulness of the measurement, or even possibly lead to misinformation in case of a large bias. More importantly, there may be *side effects* for the measurement tools, that are possibly well-known and thus avoidable or hidden and thus more insidious. An example of a well-known effect is represented by the Heisenberg uncertainty principle of quantum mechanics, which expresses limits on the simultaneous precision of complementary physical properties (in this specific case, position and momentum).

In the context of the Internet, let us focus on bandwidth measurements, that are notoriously difficult. Already considering a single measurement, there are not only several techniques based on a variety of principles (e.g., that can be coarsely split into Probe Gap [19, 45], vs. Probe Rate [22, 38]) models, but there are also studies that compare the precision and accuracy of different bandwidth measurement techniques [43, 28, 44, 42]. In general, techniques that are more intrusive are also more accurate, which is an intuitive tradeoff.

Yet, what the above tradeoff is hiding, is the fact that it applies to independent measurements, as these techniques implicitly assume that they are used in isolation. In practice, in the context of mPlane, this is unlikely (due to the scale of the population, the scale of the measurement campaign per each reasoner, and the existence of several independent reasoners) or anyway hard to enforce at any time. Consequently, while side effects may arise, it would be hard to precisely traceback the events leading to these side effects. This has already been observed in previous research on Peer-2-peer networks [8], as the peer-selection component needs to take an informed decision based

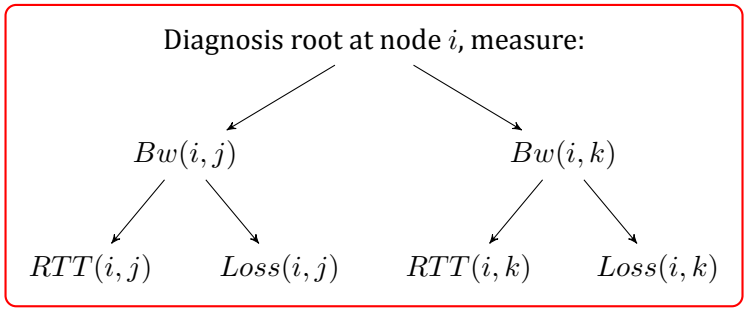


Figure 70: Interaction of homogeneous measurements: $Bw(i,j)$ and $Bw(i,k)$ may mutually interfere

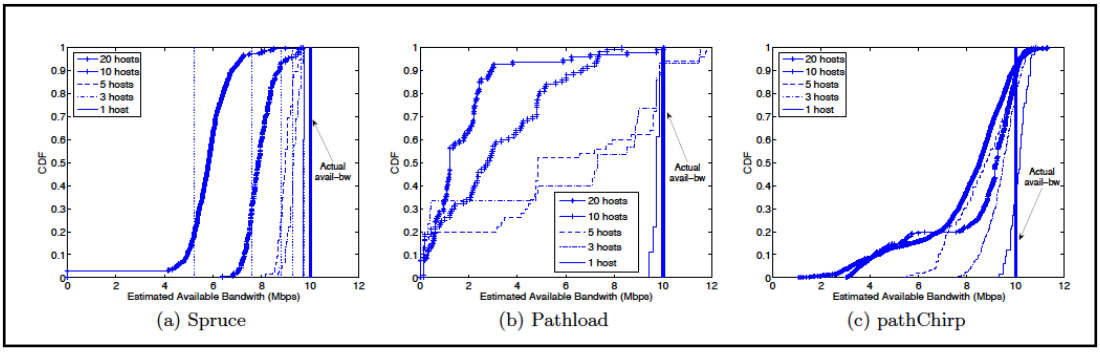


Figure 71: Interference between available bandwidth estimation tools (courtesy of [8])

on measurements coming from a relatively large peer population. Therefore, parallel probing is used in order to reduce the duration of the peer selection process as noted in 4.5.2. Simultaneous measurements of a bottleneck link from a number of hosts [8] show that the accuracy of current available bandwidth estimation tools, namely Spruce [45], Pathload [22], and PathChirp [38] drops significantly due to mutual interference. As shown in Fig. 71, when only one host is measuring the bottleneck link, all tools provide accurate estimations. However, as the number of probing hosts increases, tools tend to under-estimate the available bandwidth. Pathload and PathChirp are severely impacted by simultaneous measurements whereas Spruce results, in comparison, show more robustness. In addition to these valuable results, Croce et al. [8] also suggest the use of piggy-backed probes whenever possible to alleviate the effects of mutual interference.

At the same time, more general guidelines are hard to precisely state due to the fact that concurrent measurements do not necessarily imply interference. For instance, consider the example in Fig. 70 where host i is scheduling parallel bandwidth measurements to hosts j and k (i.e., $Bw(i,j)$ and $Bw(i,k)$), and subsequently scheduling parallel measurements of RTT and losses to the same hosts ($RTT(i,j), RTT(i,k), Loss(i,j), Loss(i,k)$). Clearly, measurements $Bw(i,j)$ and $Bw(i,k)$ will interact if i is using the same physical interface for both measurements (in case a multi-homed host i probes j,k over unrelated interfaces such as 3G and WiFi, this would not cause interference). Additionally, the bottleneck towards $Bw(i,j)$ and $Bw(i,k)$ should be located in the path segment common to both i,j and i,k pairs (which may not be the case when per-flow load balancing techniques are used, or when the bottleneck is not close to node i, etc.).

As such, while it is known that measurements may interfere, and that thus it would be good prac-

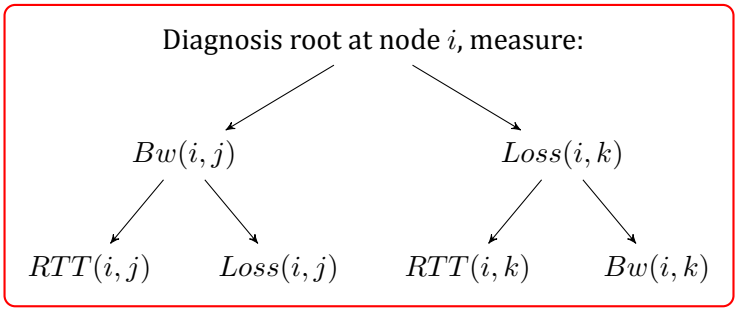


Figure 72: Interaction of heterogeneous measurement: $RTT(i,j)$ may be affected by $Bw(i,k)$

tice to reduce their potential interference by scheduling them in series, at the same time it would be perfectly legitimate to schedule them in parallel in the case of a non shared bottleneck. However, this is hard to assess in the general case as it not only requires that fine-grained topological information is taken into account to define the scheduling tree, but also it depends on dynamic components that will be hard to assess in real time. Clearly this is even more challenging since while the example considers $Bw(i,j)$ and $Bw(i,k)$, whose path shares a vertex (and an edge unless in the multi-homed case), however potentially any $Bw(i,j)$ and $Bw(m,n)$ measurement may interfere if the measurements share a bottleneck link. It follows that the problem of distributed measurement scheduling cannot be solved in the general instance. Additionally, we have exemplified the interference problem through the measurement of the bandwidth, but interference is a general problem (e.g., from SNR measurement of nearby hosts at the physical-layer, to application-layer response time on idle vs busy servers) and clearly ad hoc studies similar to [8] may be necessary for the metric of interest.

4.5.4 Interaction of heterogeneous measurements

Not only two measurements of the same metric may interact, but they may also interfere with simultaneous measurements of other metrics. To illustrate the situation, we again resort to an example of a simple tree, and focus on parallel measurements of bandwidth vs. delay (or loss) as shown in Fig. 72.

As we have seen in the previous section, bandwidth estimation tools can be coarsely split into two classes. The probe gap model infers available bandwidth from observing the inter-packet-gap (IPG) measured on a packet pair injected by the sender, which is thus a very low level of intrusiveness. It follows that, irrespectively of its lower accuracy to measure the available bandwidth, its influence on delay (or loss) measurement is expected to be very limited.

In the probe rate model, the sender iteratively injects trains of packets at different rates: the receiver then detects when the available bandwidth is exceeded by observing the increase of the One-Way-Delay (OWD) or Round Trip Time (RTT). In other words, the very same principle is to use self-induced congestion and measure the response to an intrusive active probe, varying the probe rate so as to find the right level: i.e., when the probe rate is lower than the available bandwidth no queuing happens, so that the available bandwidth is found for a probing rate yielding the smallest non-null amount of queuing delay.

Interference with delay (or loss) measurement is thus intrinsic in the methodology: as probes typi-

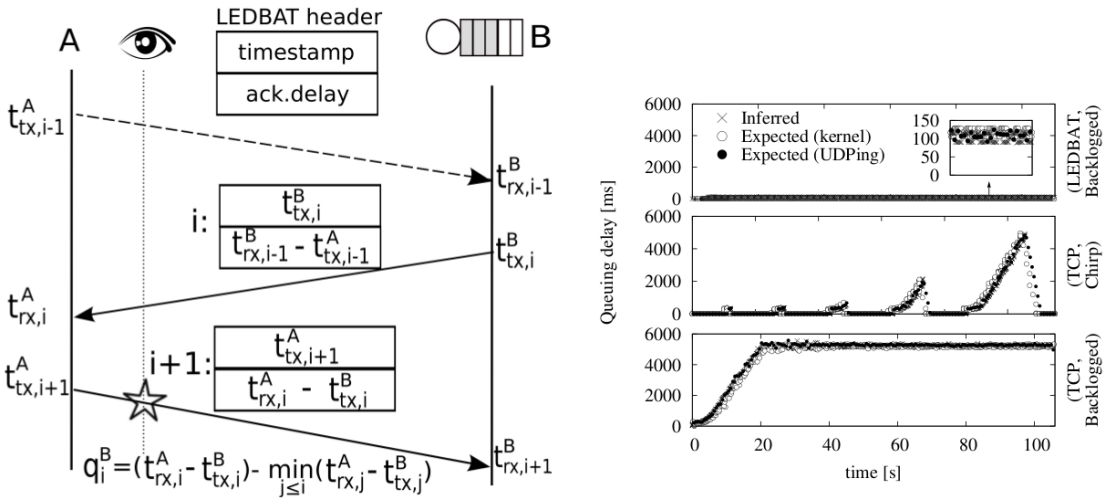


Figure 73: Interference of latency and load measurement: methodology to infer queueing delay (left) and validation with controlled background traffic (right).

cally iterate with dichotomic or binary searches to let the probe rate converge to the available bandwidth rate (where convergence is measured as a function of the inferred queueing delay), it follows that the mechanism by design alters queuing (and possibly induces losses when the probe rate is too aggressive). As a consequence, in Fig. 72, $RTT(i,k)$ may be affected by $Bw(i,k)$, depending on the technique employed to measure $Bw(i,k)$. Additionally, in case the bottleneck is located in a segment common to both the i,j and i,k paths, then bandwidth measurement $Bw(i,j)$ can possibly affect $Loss(i,k)$ when the probing rate is too high with respect to the available bandwidth, and provided that the buffer fills up during the measurement timescale (similar considerations hold for $Bw(i,k)$ vs $RTT(i,j)$ and $Loss(i,j)$).

This is illustrated in [6], which shows an account of the techniques developed in mPlane to measure the extent of queuing delay of remote Internet hosts, by exploiting chunk transfers from unmodified BitTorrent hosts[7, 5]. Specifically, we estimate queuing delay by collecting one-way delay (OWD) samples, establishing the minimum as a baseline delay, and then measuring the degree to which a sample differs from the baseline, as illustrated (validated) in the left (right) plot of Fig. 73. This is a classic approach used in congestion control to drive congestion window dynamics, starting from Jain's pioneering work in the late 80s [23], to the widely known TCP Vegas [2] in the late 90s, to ultimately the LEDBAT [41] protocol proposed in 2010 by BitTorrent as a replacement of TCP for data transfer. Specifically, our innovation in [6] was to demonstrate how a passive observer of LEDBAT or TCP traffic can use this approach to estimate the uplink delays experienced by a remote host, and to conduct large scale measurement studies showing that, when the maximum queuing delay is potentially very large (up to several seconds[27], for which the bufferbloat term was coined) this was due to the measurement methodology of latency under load, causing mutual interference between measurements, and that whenever measurements were done in a non-biased non-intrusive fashion, the typically observed queueing delay statistics were much lower[7, 5].

In this context, the knowledge of the measurement methodology is useful as it implicitly shows the principle behind self-induced congestion, and shows the intrinsic interference between bandwidth latency and loss. Considering the top plot, where no other traffic other than probe traffic is sent,

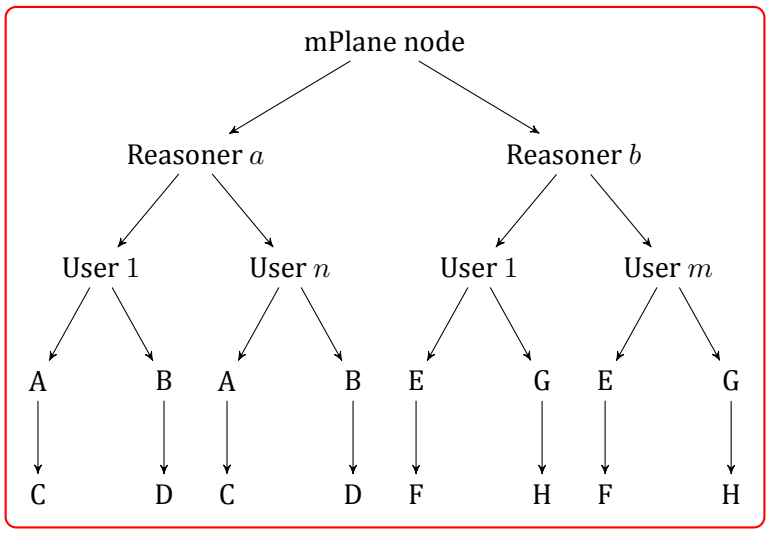


Figure 74: Multiple levels of dependencies: per-user, per-reasoner, per-node

results show that the available bandwidth exploited by LEDBAT causes a 100ms queueing delay: while this queueing delay is not harmful for the user compared to multi-second queueing delays (aka bufferbloat), however it affects the queueing delay measurement. In mPlane terms, LEDBAT can be thought as an available bandwidth measurement tool, and the 100ms queuing delay can be thought as the lowest empirical value that allows to reliably identify the bottleneck bandwidth, and the magnitude of influence on delay measurement. In the bottom plot of Fig. 73, it can be seen that under sustained TCP transfers, the queuing delay saturates to the maximum available (aka bufferbloat): since the buffer is full, some TCP packets will be dropped, so that we also infer a bandwidth-loss interference.

Exposing interference between measurements allows to arrange scheduling trees that, at least for a single user, minimizes the interference. Additionally, as in the previous section, it is possible to select tools that are less intrusive, to reduce the interference, though this may lead to loss of accuracy for some measurements. Assessing that the combination of tools, and the selected scheduling of measurements, will not cause interference while retaining sufficient accuracy to meet the reasoner objectives, requires rigorous calibration and validation.

4.5.5 Interaction of multiple diagnosis trees

Finally, a further level of dependencies occurs in the case where a given mPlane node is running multiple reasoners as shown in Fig. 74. Here, measurements instrumented by reasoners a and b are likely to mutually interfere leading to a biased or inaccurate diagnosis.

Three scenarios can potentially lead to mutual interference in this context. The first scenario consists in more than one reasoner requesting a given user registered to the domain of the mPlane node to run the same kind of network performance measurements, e.g. measure the delay of the access link. Multiple instances of the same measurement will be running at the user access link causing waste of network resources, interference with user traffic, and interference with each other. In the second scenario, the same user is requested to launch two different measurements (e.g., delay

and bandwidth of a given path) from two diagnosis graphs. Both scenarios are similar to the cases discussed at length in the previous sections where homogenous and heterogenous parallel measurements from one diagnosis tree and one user mutually interact. In the third scenario, multiple reasoners send requests to a subset of their users (not necessarily the same users) to measure, for instance, the latency of a video streaming service experiencing a performance deterioration. Although the measurements are launched from different users, they are targeting the same service. If the geographical scope of the measurements is limited to one area and a significant number of mPlane probes are sending measurements to the same target server(s), this situation could overload the server (thus biasing all the measurements). In addition, depending on the frequency of measurements, the servers can interpret this measurement traffic as a distributed denial of service attack. They can also blacklist the mPlane probes causing a situation where diagnosis of the service performance problems is not feasible anymore.

Coordination among the reasoners running on the same mPlane node can potentially reduce the effects of mutual interference. Sharing measurements or scheduling them are two possible ways of coordination. For instance, if reasoners a and b are actively monitoring the same performance metric of a given network component, then it is better to measure this component once. Sharing measurements across reasoners also implies giving mutual access to historic performance data of common users at the mPlane repositories. When the reasoners require measurements of different metrics from the same user such as delay and bandwidth, scheduling policies can solve the interference problem by allocating non overlapping time slots for each reasoner. However, implementing such scheduling policies is not so trivial as it should take into account the execution time and the priority level of each measurement. Another possible alternative to scheduling, which is currently adopted by RIPE atlas is to enforce a cap on the data rate consumption of a given reasoner per user and a limit on the number of probes targeting the same destination.

5 Conclusions

This deliverable presents the design and specification of the different components of the mPlane Reasoner. The Reasoner-relevant concepts are exemplified through the realization of the iterative analysis process for each of the mPlane use cases. By presenting the analysis workflow of the Reasoner for each of the use cases, along with some primarily results with real world measurements in different types of networks and setups, the reader can get a more tangible picture of how the overall mPlane operates in the practice.

For those use cases supporting troubleshooting applications, the deliverable also presents a set of domain knowledge based rules which provide a basic Knowledge Structure for mPlane, allowing the instantiation of new diagnosis graphs to tackle new use cases.

Finally, taking into account that relying exclusively on analysis rules defined on the basis of domain knowledge and operational experience can result in lower analysis performance, the deliverable presents different learning approaches for extending and/or generating new analysis rules within the Knowledge Structure.

The design and specification of the Reasoner is not yet completed and we are still working on improving the presented concepts and techniques, but the core of the system as presented in this deliverable already brings all the different components required for the complete mPlane to operate as envisioned.

References

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

[2] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *ACM SIGCOMM CCR*, 24(4):24--35, 1994.

[3] P. Casas, A. Sackl, S. Egger, and R. Schatz. YouTube & Facebook Quality of Experience in Mobile Broadband Networks. *IEEE Globecom Workshops*, 2012.

[4] P. Casas, M. Seufert, and R. Schatz. YOUQMON: A System for On-line Monitoring of YouTube QoE in Operational 3G Networks. *IFIP Performance*, 2013.

[5] C. Chirichella and D. Rossi. To the moon and back: are internet bufferbloat delays really that large. In *IEEE INFOCOM Workshop on Traffic Measurement and Analysis (TMA'13)*, Turin, Italy, April 14-19 2013. keyword=ledbat,bufferbloat,mplane.

[6] C. Chirichella, D. Rossi, C. Testa, T. Friedman, and A. Pescape. Passive bufferbloat measurement exploiting transport layer information. In *IEEE GLOBECOM*, December 2013. keyword=traffic,ledbat.

[7] C. Chirichella, D. Rossi, C. Testa, T. Friedman, and A. Pescape. Remotely gauging upstream bufferbloat delays. In *Passive and Active Measurement (PAM), Extended Abstract*, Hong Kong, China, March 18-19 2013. keyword=ledbat,bufferbloat,mplane.

[8] D. Croce, M. Mellia, and E. Leonardi. The quest for bandwidth estimation techniques for large-scale distributed systems. In *Proc. ACM SIGMETRICS*, 2010.

[9] A. D.Koller. Probabilistic frame-based systems. In *Proceedings of 15th National Conference on Artificial Intelligence (AAAI)*, pages 580--587, July 1998.

[10] 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.

[11] M. Ester, P. Kriegel, J. Sander, and X.Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *ACM SIGKDD*, 1996.

[12] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi. Experiences of internet traffic monitoring with tstat. *IEEE Network Magazine*, May 2011.

[13] A. Finamore, M. Mellia, M. Munafo, R. Torres, and S. R. Rao. YouTube Everywhere: Impact of Device and Infrastructure Synergies on User Experience. *ACM IMC*, 2011.

[14] A. Fred and A. K. Jain. Combining Multiple Clusterings Using Evidence Accumulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6), 2005.

[15] L. Getoor and D. Jensen. Proc. of AAAI-2000 Workshop on Learning Statistical Models from Relational Data. *AAAI Press*, 2000.

[16] 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.

[17] S. Hemminger. Network Emulation with NetEm. In *Linux Conf Au*, Apr 2005.

[18] P. H.Poon. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings 21st National Conference on Artificial Intelligence (AAAI)*. AAAI Press, 2006.

[19] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications (JSAC)*, 21:879--894, 2003.

[20] T. Jaakkola and D. Haussler. Exploiting Generative Models in Discriminative Classifiers. *Advances in Neural Inf. Processing Sys. II*, 1998.

[21] A. K. Jain. Data Clustering: 50 Years Beyond K-Means. *Pattern Recognition Letters*, 31(8), 2010.

[22] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *ACM SIGCOMM*, 2002.

[23] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM CCR*, 19(5):56--71, 1989.

[24] J. Jiang, V. Sekar, I. Stoica, and H. Zhang. Shedding Light on the Structure of Internet Video Quality Problems in the Wild. *ACM CoNEXT*, 2013.

[25] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. *The Satisfiability Problem: Theory and Applications*, 1997.

[26] L. R. K.Kersting. Adaptive bayesian logic programs. In *Proceedings of 11th Conference on Inductive Logic Programming (ILP-01)*, volume 2157 of *Lecture Note in Computer Science*. Springer, 2001.

[27] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *ACM Internet Measurement Conference (ACM IMC'10)*, 2010.

[28] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye. Bandwidth estimation in broadband access networks. In *Proc. Internet Measurement Conference*, 2004.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] L. Parsons, E. Haque, and H. Liu. Subspace Clustering for High Dimensional Data: a Review. *ACM SIGKDD Expl. Newsletter*, 6(1), 2004.

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

[36] T. A. M. Project. The Apache Mahout machine learning library. Available: http://mahout.apache.org/ Last access on March 2013. *Apache Mahout Project Website*, 2013.

[37] D. P.Taskar, P. Abbeel. Discriminative probabilistic models for relational data. In *Proceedings of 18th Conference on Uncertainty in Artificial Intelligence (UAI 2002)*, pages 485--1¤74, 2002.

[38] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Proc. of Passive and Active Measurement Workshop*, 2003.

[39] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2), 2006.

[40] `https://atlas.ripe.net`.

[41] S. Shalunov et al. Low Extra Delay Background Transport (LEDBAT). IETF RFC 6817, 2012.

[42] A. Shriram and J. Kaur. Empirical evaluation of techniques for measuring available bandwidth. In *IEEE INFOCOM*, 2007.

[43] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and kc claffy. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *Proc. of Passive and Active Measurement Workshop*, 2005.

[44] J. Sommers, P. Barford, and W. Willinger. Laboratory-based calibration of available bandwidth estimation tools. *Microprocessors and Microsystems*, 31(4):222--235, 2007.

[45] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. Internet Measurement Conference*, 2003.

[46] B. Trammell, P. Casas, D. Rossi, A. Bär, Z. Ben-Houidi, I. Leontiadis, T. Szemethy, and M. Mellia. mPlane: an Intelligent Measurement Plane for the Internet. *IEEE Communications Magazine*, 42(5), 2014.

[47] H. Yan, L. Breslau, Z. Ge, D. Massey, D. Pei, and J. Yates. G-RCA: A Generic Root Cause Analysis Platform for Service Quality Management in Large IP Networks. *ACM CoNEXT*, 2010.

[48] 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.

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

[50] 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.