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:	1.1
Revision Date:	21 Apr 2015
Deliverable Type:	RTD
Due Date of Delivery:	21 Apr 2015
Actual Date of Delivery:	21 Apr 2015
Nature of the Deliverable:	(R)eport
Dissemination Level:	Public

Abstract:

This deliverable describes the logic 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 additionally details how the proposed logical design is generic enough to tackle different classes of measurement analysis processes, and in particular those classes covered by the mPlane use cases: troubleshooting-support based analysis, and generic measurements analysis. For doing so, it presents different per use case instantiations of the mPlane design, showing how it is possible to map the proposed logical design to the different classes of mPlane use cases. Finally, different learning techniques for extending and/or generating the knowledge structure of the Reasoner are over-viewed.

SEVENTH FRAMEWORK PROGRAMME

Disclaimer

The information, documentation and figures available in this deliverable are written by the mPlane Consortium partners under EC co-financing (project FP7-ICT-318627) and does not necessarily reflect the view of the European Commission.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.

Contents

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.

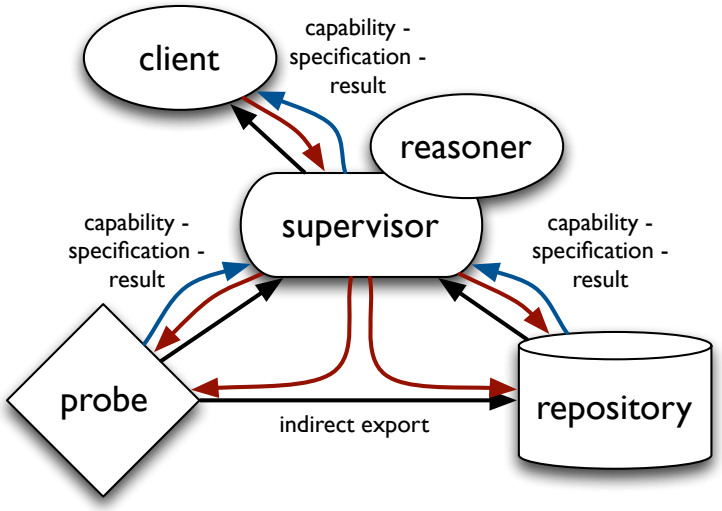


Figure 1: The mPlane architecture. The Reasoner coordinates the measurements and the analysis performed by probes and repositories, actuating through the Supervisor.

In this deliverable we present the **Reasoner**, which represents the "intelligence" of the mPlane. The Reasoner allows structured, iterative, and partially automated analysis of the measurements and intermediate analysis results. The term "partially" refers to the fact that even if desirable, the complete mPlane is not intended to fully automate the monitoring applications it supports, but rather to better guide and support the measurement and analysis process. The Reasoner 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. The communication between the Reasoner and the mPlane components is performed through the interfaces provided by the mPlane Reference Implementation and the corresponding components. This decoupling between coordination and measurement iteration allows to fully split the roles of the mPlane Reasoner and the mPlane Supervisor. Indeed, while the Supervisor offers access to each mPlane component capabilities via a common interface (using WP3 and WP2 components' interfaces), the Reasoner becomes a client of the Supervisor, like any other client that interacts with the Supervisor. As such, it allows the user to implement automatic processes, e.g., to

come up with a custom *instantiation* of the Reasoner that suits her needs. The Reasoner is therefore in essence a replacement for a human user for certain tasks, thus this separation of responsibilities makes a good deal of intuitive sense. Fig. 1 recalls the architecture of the mPlane, where the roles and interactions among mPlane components is depicted, specially regarding the Reasoner and the Supervisor.

Regarding the design of the mPlane Reasoner, the original design we had in mind at the time of writing the proposal was to have a "single" Reasoner that would orchestrate the iterative analysis performed by the system, for all the potential use cases. This would entail a holistic approach, to be generic enough to adapt to any scenario. However, when then working on the project, and after the precise definition of the selected use cases, we realized that this holistic approach would be impracticable. This is due to the heterogeneity of measurement scenarios, the general lack of common measuremnets' definitions, and to the general complexity of the task. Indeed, even if many different expert systems for automating network traffic analysis have been proposed in the past, both in the research community as well as in commercial solutions, their practical success is very marginal. As an example of this, neither the mPlane ISPs and SMEs nor other ISPs collaborating with the mPlane project members rely on such systems to handle their daily operations.

The solution we took and that we present in this deliverable is then to have a possible generic framework that could allow the design of specific Reasoners, e.g., a different **Reasoner instantiation** for each use case, based on the same generic principles. This would allow us to customize each instantiation of the Reasoner to the specific use case, and in general would bring flexible design principles to adapt to different scenarios and measurement applications. For the aforementioned reasons, this deliverable does not focus on detailing interfaces or highly abstract representations, but rather on the logical entities which compose the Reasoner, which allow to instantiate a Reasoner tailored to the specific needs of a use case.

Starting by the **design and specification of the different logical components of the Reasoner in section 2**, this deliverable presents in **section 3 a per use-case class description on how a Reasoner can be instantiated** in the practice from such a logical design. These use-case classes are defined on the basis of the specific role of the Reasoner in the measurement analysis process. In particular, mPlane use cases can be split in troubleshooting-support based analysis and generic measurements analysis. Finally, **different learning approaches for extending and/or generating new analysis rules within the Knowledge Structure are presented in appendix A**, and some considerations for building diagnosis graphs are discussed.

A final note on the level of abstraction considered in the description of the different logical components of the Reasoner, as presented in this document. As we said before, we decided to go for a more practical approach rather than a more generic and holistic one, trading flexibility by specificity. While this is a general trade-off applicable to all domains, it becomes highly relevant when thinking about a system such as the Reasoner, highly related to the concepts of intelligence. Generally speaking, when designing intelligent systems, one can follow a more *procedural* approach in which knowledge (in the more general term) is directly integrated into the specific task, or a more *declarative* approach, in which knowledge is represented completely independently of the specific tasks to address [9]. The procedural approach provides more concrete and specific definitions, considering in advance all the components and analysis procedures. However, the obtained results lack versatility and adaptability, and are more difficult to modify. The declarative approach is more

flexible but requires a very hard logical base to operate properly, and even if it results in a great level of abstraction, its main blocking point is on the difficulty to properly define and constantly maintain a good and updated logic describing the operational context.

Based on the previously discussed observations and to favor a more practical (and thus more applicable and easier-to-reuse approach), we have decided to follow a more procedural approach for the design of the Reasoner. Still, as we see next, the proposed logical components are generic enough to allow the instantiation of a Reasoner tackling the specific needs of any measurement and analysis process.

2 Design and Specification of the Reasoner

In this section we present the design and specification of the logical components conforming the Reasoner, which represents the intelligence 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. When considering the use cases selected within the mPlane project, we can identify two main groups or classes: (i) those use cases based on troubleshooting support, and (ii) those use cases based on generic measurements analysis. 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 [35]. 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.

Fig. 2 depicts an overview on the complete Reasoner system. The Reasoner is divided in three major logical 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 causes were automatically identified. Next we provide a general description of these logical components, which are further detailed in the following sections.

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***.

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

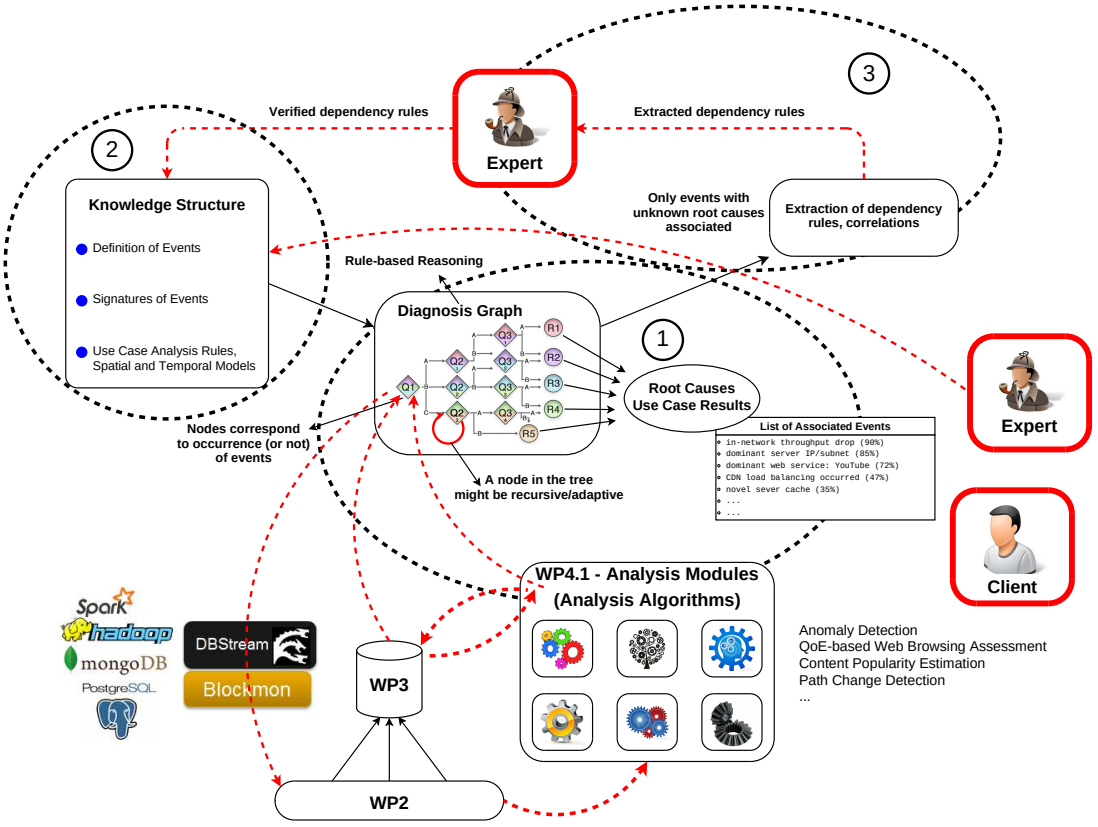


Figure 2: Automatic anomaly diagnosis and events exploration through the Reasoner.

analysis, such as anomaly detection) or in an on-demand fashion (e.g., specific reactive query, such as server reachability measurements). In general terms, 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), and are directly related to the specific analysis goal (e.g., Anomaly Detection). Symptom events are normally employed as triggers for some specific action. A very simple and intuitive example is provided by the Anomaly Detection scenarios. For example, if the goal of the analysis is to detect QoE-relevant degradation in YouTube, a symptom event could be defined by the occurrence of a drop in the average downlink flow throughput of YouTube flows below a certain threshold (let us assume for the moment that such a predefined threshold exists).

Diagnosis events refer to the evidence of a potential root cause taking place (e.g., Google CDN load-balancing). Diagnosis events provide contextual details of the problems flagged by symptom events, and are used by the Reasoner in a more evolved analysis fashion, as by combining the occurrence of several diagnosis events it might resort to the root causes of the detected symptoms. If we go back again to the example of YouTube, when it comes to diagnosing QoE-relevant degradation, potential diagnosis events can be defined by the occurrence of congestion at the access links, the occurrence of server-to-customers path modifications, the occurrence of anomalously high rates of DNS non-resolved or delayed requests, etc. While non of these diagnosis events is a direct symptom of QoE degradation in YouTube, it could directly point to the underlying root causes.

The iterative analysis engine of the Reasoner verifies 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 logical 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 logical 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 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/analysis modules needed to generate/detect the occurrence of the corresponding event. Events are generated by the relevant analysis modules, applied to a set of monitored KPIs (e.g., abrupt change detection, heavy hitter identification, statistical traffic/KPIs analysis, etc.). As the reader should note, **the fields defining an event could be formally defined as standard *entities*** with a corresponding entity *type* and *name*, and using whatever type of representation language. However, as we claimed in the introduction, we decided not to follow this approach and let definitions more open, such that every specific instantiation of these concepts could be tailored for the specific needs of the analysis process.

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%), {1.5 kflows, 235.5 MB, srvIP_list,...}>
```

As we said before, 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 once again the YouTube QoE Anomaly Detection use case, a symptom event could be a major drop in the YouTube flows throughput impacting the QoE of a large number of 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 kflows, 6.1 kusers, srvIP_list, download_th dist, avg_QoE,...}>

In this case, an event reflecting an anomaly in the YouTube QoE of `6,100 users` is generated by the Statistical Anomaly Detection module `SAD`, configured with the input parameters x_1, x_2, ..., x_k. The `SAD` analysis module 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:

$$\text{if} \left\{ \begin{array}{l} \textbf{YouTube\_QoE\_anomaly}: \\ \text{QoE anomaly detected in PoP\_1, ISP\_A} \\ \qquad\qquad \& \\ \textbf{Google\_load\_balancing}: \\ \text{New Google cache selected in AS 15169} \\ \qquad\qquad \& \\ \sim\textbf{Path\_perf}: \\ \text{No performance issues in inter-AS end-to-end paths} \\ \qquad\qquad \& \\ \textbf{Server\_anomaly}: \\ \text{Server issues detected in new servers between 20hs-23hs} \end{array} \right\} \rightarrow \begin{array}{c} \text{YouTube servers} \\ \text{overloaded at peak time.} \end{array}$$

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 fault diagnosis is to clearly understand the dependency relationships between 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.

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

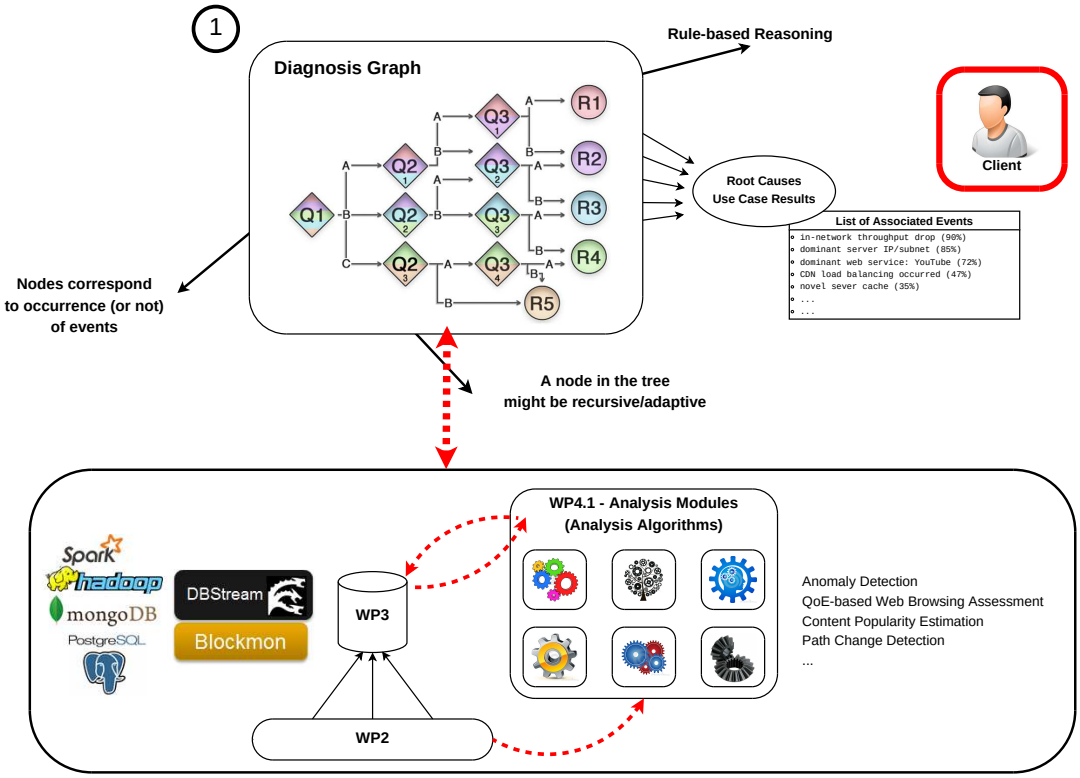


Figure 3: The diagnosis graph. A diagnosis graph combines a set of analysis rules in a graph-like structure, which explores the temporal and spatial relationships between symptom and diagnosis events to find the most probable root causes.

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 which permit to structure the diagnosis graphs. This list of events and rules available at the Knowledge Structure permits their re-usage 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 such, the proposed approach is generic to address the needs of multiple use cases, by defining the required events and rules, and re-using those already fitting the purpose of the analysis.

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 information related to the generated rules, i.e., they select the most relevant features related to the specific events.

Learning approaches for the Reasoner are described in appendix A.

3 Examples of Instantiation of the Reasoner

This section presents the Reasoner by example, with the main target of exemplifying how the previously descried design can be mapped or instantiated into a Reasoner tailored for the considered use case. We chose two different use cases representing each of the aforementioned use-case classes to make the exercise more generic: for **troubleshooting support-based use cases**, we address the cases of Anomaly Detection and Root Cause Analysis as well as DaaS Troubleshooting. In the case of **generic measurements-based analysis use cases**, we select the cases of Content Popularity Estimation as well as Passive Content Curation.

The following sections briefly describe the role of the instantiated Reasoner, a set of iterative analysis and diagnosis rules, and a diagnosis/iterative analysis graph, the last two built from expert domain knowledge.

3.1 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 event is detected.

As an example of the application of the Reasoner in this use case, we present a scenario aiming at detecting and diagnosing 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 particular instantiation of the mPlane Reasoner for this use case.

3.1.1 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 in the specific use case. These algorithms include 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 [29]. Secondly,

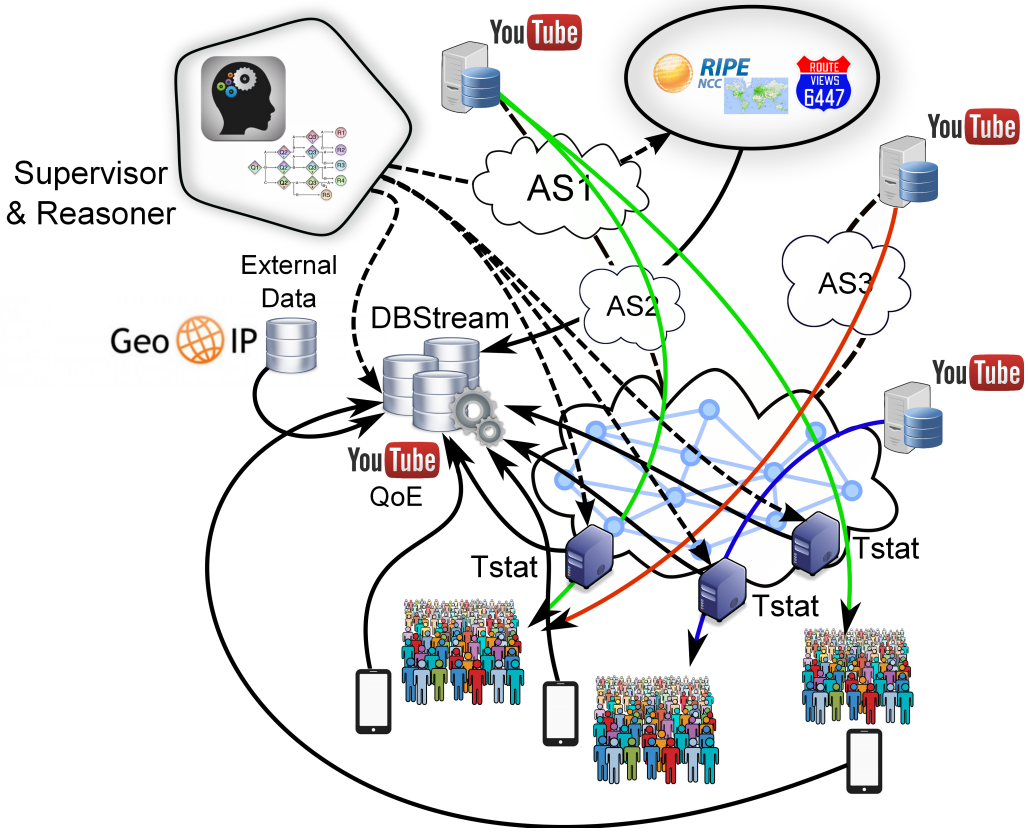


Figure 4: Detection and diagnosis of QoE-based large-scale anomalies in YouTube. The term large-scale reflects those anomalies which impact a large number of customers. QoE-based measurements at the end-devices are potentially used for diagnosing device-problems. However, as we target large-scale anomalies, end-device issues are generally discarded from the beginning.

the Reasoner is in charge 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. 4 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 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 charge of performing the large scale analysis of the combined measurements, which is described next. Tstat flow measurements are combined with two other types of measure-

ments: (i) external data coming from geo-localization services such as MaxMind[1] and IP-address analysis services such as Team Cymru Community Services[2], 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 [29] and BGP measurements coming from Route-Views[3].

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 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.

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, we introduced a new approach, based on flow-level measurements. This approach is fully detailed in D4.3. However, for the sake of consistency, we briefly describe it next. 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. As we show in D4.3, no stallings are observed for $\beta > 1.25$, and user experience is rather optimal (MOS > 4). On the other hand, stalling occurs for values of $\beta < 1.25$, resulting in bad quality (MOS ≈ 2.5) around $\beta = 1$.

3.1.2 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

[1] https://www.maxmind.com
[2] https://www.team-cymru.org/
[3] http://routeviews.org/

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[4] – 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.

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

- **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. 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 geo-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 a 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? → check the occurrence of re-configuration events related to the corresponding access networks or slots.
3. Equipment failure related? → 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 topological scope of the monitoring layer.

3.1.3 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. 5 depicts an exemplifying decision graph, integrating some of the previous diagnosis rules. As explained in A.4, 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. 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.

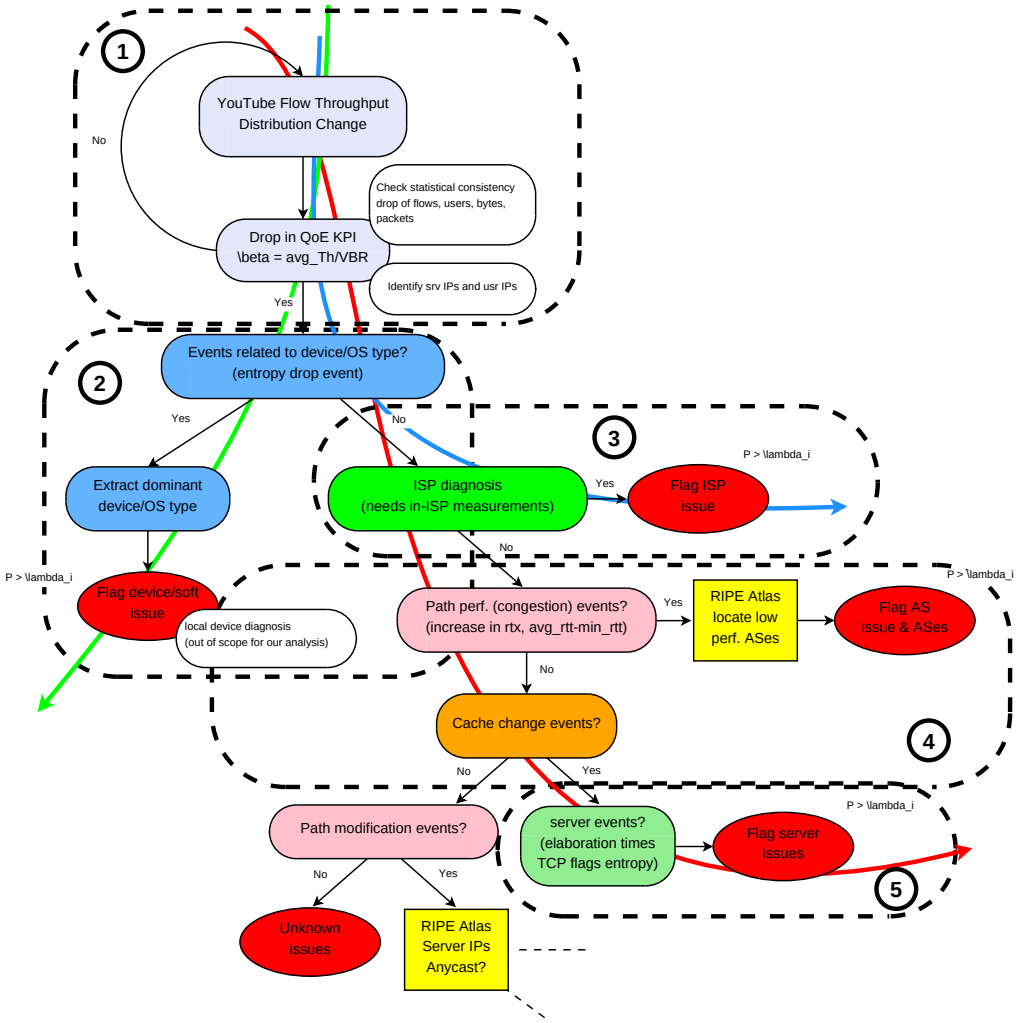


Figure 5: Diagnosis graph associated to the detection and troubleshooting support of large-scale QoE-based anomalies in YouTube.

3.2 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.

3.2.1 The Role of the Reasoner

Fig. 6 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 collect IP-level features that can be accessed from the thin-client connection while it is running, such as packet size, rate,

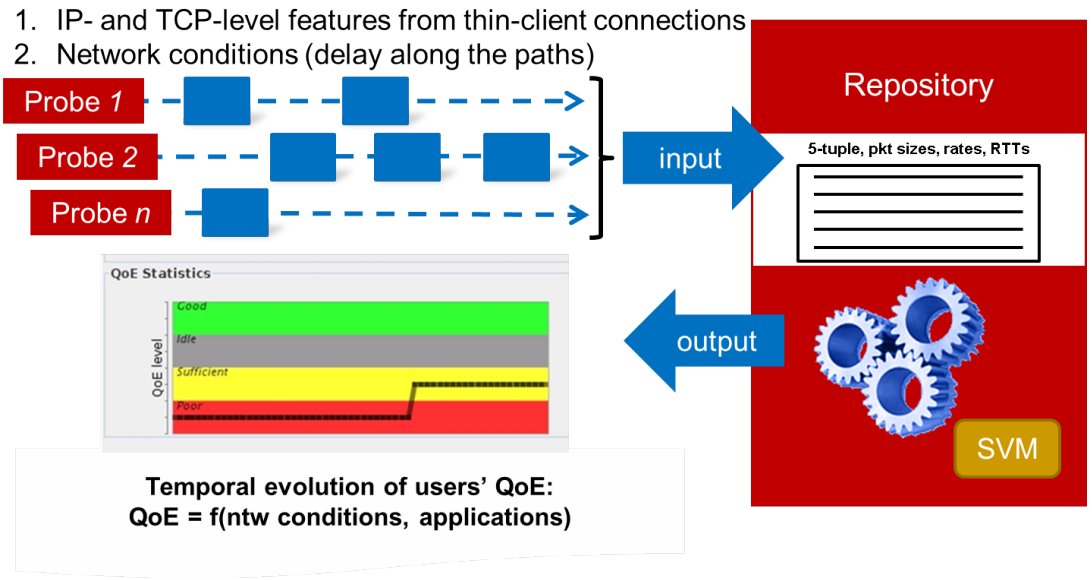


Figure 6: mPlane components for the use case "Supporting DaaS troubleshooting". Given a thin-client connection, the Reasoner observes as input the evolution of the user experience over time, and triggers actions based on how the quality evolves.

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.

3.2.2 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.

To this purpose, the analysis algorithm periodically generates events as follows:

```
<flow_type, 2013-10-21-12:30:00, 2013-10-21-12:35:00,
DaaS(SVM), {flow_tuple, RTT, ...}>
```

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.

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.

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.

By comparing the flow type with the current value of RTT, the Reasoner can work on symptom events of the following kind:

```
<poor_QoE, 2013-10-21-12:30:00, 2013-10-21-12:35:00,
DaaS(), {flow_tuple}>
```

3.2.3 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. Events that the Reasoner receives are as follow:

```
<end_point, 2013-10-21-12:30:00, 2013-10-21-12:35:00,
traceroute, {(hop,delay),...}>
```

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. 7. Each decision will lead to a generation of the corresponding diagnosis event.

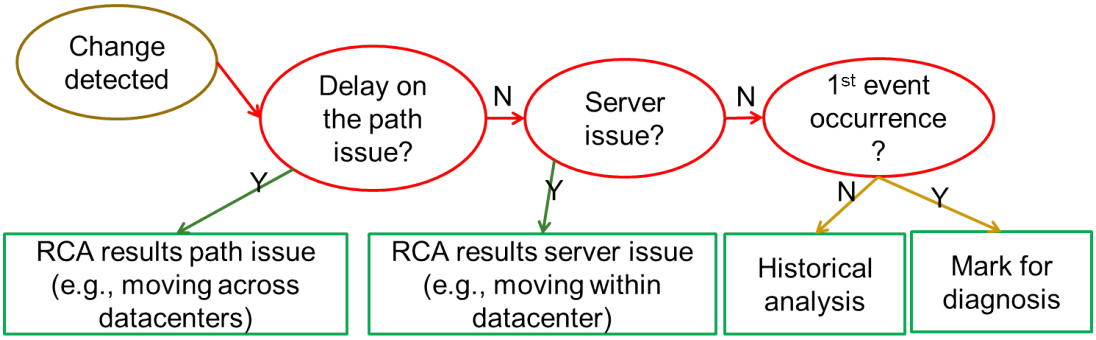


Figure 7: Desktop-as-a-Service troubleshooting: Diagnosis analysis graph. A change is detected in case of poor QoE of a user running a thin-client connection.

3.3 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 logs about the traffic flowing in the network, and extract the contents the users download in several points in the network. The acquired information is exploited to predict the popularity of the contents and suggest efficient caching replacement strategies to the Reasoner.

3.3.1 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 popularity evolution of contents observed in the traffic and raises alarms to notify which contents are becoming popular and which network portions. Such information can be exploited, e.g., to optimize the caching of labeled-as-popular contents in a Content Distribution system with hierarchical infrastructure.

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 an identifier of the content (e.g., the URL to access it), the associated timestamp and the network location of the probe. Based on these features, the analysis modules are responsible for predicting the future popularity of each requested content at each part of the network (i.e., the probe location).

This task is accomplished by employing two different analysis modules: the first one, named Popularity Classifier, takes as input the popularity history for a content, it generates a signature for its request arrival process using Gaussian Mixture Modelling techniques, and classifies such content using a methodology based on the Latent Dirichlet allocation, whose classification result is stored at the repository. In parallel, for each observed content the Online Predictor explores the classification results to find the popularity pattern which maximizes a likelihood function. Once the most similar 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 to notify which contents are becoming popular and where in the network.

In its current status, the Reasoner gets the list of (estimated) popular contents from the analysis modules that run continuously, together with information about the network portion (i.e., the probe) in which such content was observed. Following the terminology defined in Sec. 2 these correspond to symptom events containing the following parameters:

- `Symptom event:` Increasing Popularity
- `Probe location`: indicates the location of the probe at which the content has been labeled at popular.
- `Time span`: indicates the time period at which the content is expected to be popular.
- `Predictor`: the name of the analysis module running the predictor algorithm.
- `URL`: contains the URL to the content labeled as popular.

Hence, a possible example of symptom event signaling the occurrence of a popular content in the network could be:

```
<Popularity-event, PoP_1, 2014-10-21-12:30:00, 2013-10-21-12:35:00,
GMM-LDA, {url=http://path.to/acme/hotcontent.jpg}>
```

3.3.2 Domain-knowledge based Iterative Rules

Given the above symptom events the decision rules we define for this use case are straightforward. Essentially, we define several decision rules to measure the spatial breadth in the network of the popularity of each content. This is particularly welcome for hierarchically structured Content Delivery Networks (CDNs): intuitively, the spatially wider the popularity of the content, the higher the position in the hierarchy of the server which should cache it. Conversely, if the content shows a localized popularity, it should be cached in leaf server in the CDN hierarchy. Therefore, for this use case, each decision rule aims at identifying the different locations at which a content has been labeled as potentially popular, as depicted in the following example:

$$\text{if} \left\{ \begin{array}{l} \textbf{Popularity\_increase}: \\ \texttt{Content Popularity Increase in PoP\_1, ISP\_A} \\ \quad\quad\quad\quad \& \\ \textbf{Popularity\_increase}: \\ \texttt{Content Popularity Increase in PoP\_2, ISP\_A} \\ \quad\quad\quad\quad \& \\ \textbf{Popularity\_increase}: \\ \texttt{Content Popularity Increase in PoP\_1, ISP\_B} \\ \quad\quad\quad\quad \& \\ \textbf{Popularity\_increase}: \\ \texttt{Content Popularity Increase in PoP\_2, ISP\_B} \end{array} \right\} \rightarrow \begin{array}{l} \texttt{Spatial Popuparity breadth:} \\ \texttt{All PoPs.} \end{array}$$

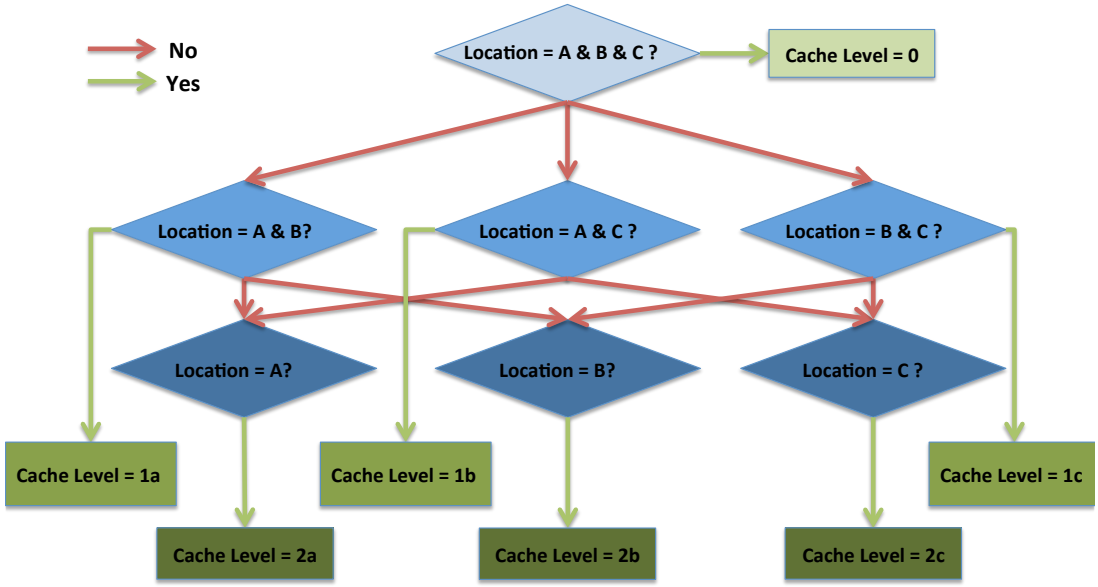


Figure 8: Content Popularity Estimation: Diagnosis analysis graph. The graph determines the popularity spread in the network for each labeled-as-popular content.

3.3.3 Diagnosis/Iterative Analysis Graph

As rules aim at identifying the network locations at which a given content has been labeled as popular, the diagnosis graph simply determines the level in the CDN hierarchy at which the caching scheme should store the content and the network location(s). A simplified example of the diagnosis graph is depicted in Fig. 8. As shown, if the content is labeled as potentially popular at all the probes, then it worths being cached in the highest caching level. Otherwise, it has to be cached at lower, more localized levels of the hierarchy.

Finally, we remark that this use case does not need any iterative reasoning to, and its working flow does not need any "learning capabilities" to be expanded automatically. However, the reasoner can improve its prediction performance as the LDA-based classifier has been designed to improve its accuracy over time.

3.4 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 interesting content from the crowd's browsing behavior. We believe that content 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.

3.4.1 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 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 Reasoner's analysis modules get as an input **events** corresponding to the detection of *interesting URLs* that are getting the attention of the crowd (detection done by the scalable data analysis algorithms 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. Therefore, considering the terminology defined in Sec. 2, a symptom event will contain the following information:

- `Symptom event:` Interesting URL
- `Probe location`: indicates the location of the probe at which the URL has been elected.
- `Time span`: indicates the time period at which the interesting URL has been captured.
- `Interesting URL detector`: the name of the analysis module running the detector algorithm.
- `URL`: contains the interesting URL.

Thus, an example of symptom event reporting the occurrence of an interesting URL may be structured as follows:

```
<Interesting-URL, PoP_1, 2014-10-21-12:30:00, 2013-10-21-12:35:00,
F-Ref+F-type detector, {url=http://path.to/acme/interesting-url/}>
```

Since not all interesting URLs are worth recommending to the users, the Reasoner leverages a set of analysis modules to classify and pinpoint the interesting URLs whose content may represent a good candidate to be promoted. In particular, content aggregation web pages (e.g., the home page of a popular online newspaper that promotes a wide number of news) are popular but do not pertain to a specific news or article. The Reasoner, with the additional help of the *content vs portal analysis module* (that will be detailed in next deliverable), must decide whether if an interesting-URL corresponds to a content or to a portal. In order to do so, the Reasoner continuously updates a *knowledge database* filled with URLs that correspond to Portals (which are URLs that we do not want our system to recommend). The Reasoner, thanks to the content vs portal analysis module, learns from the past time series of a given interesting URL to infer such an information. Intuitively, if the URL timeseries shows a daily periodicity, then it corresponds to a portal. The content versus portal analysis module continuously pulls symptom events containing interesting URLs from the repository, if the URL is present in the portal knowledge database, it is discarded, otherwise an online heuristic (less precise) is applied to guess whether this unknown URL corresponds to a Portal. If not, a **content URL detection event** is created. The Reasoner then appends this URL to the list of followed URLs. By chaining several analysis modules as the one above described, we forms a simple primitive diagnosis graph whose role is to diagnose which URLs correspond to content URLs and which ones should be tagged as portal URLs.

Finally, for the followed URLs, the Reasoner, leveraging the *Track rising content* module, and instrumenting the supervisor, asks the repository to get their popularity and timeseries. Our promotion algorithms currently elect three types of contents. The first is a live stream of public news that are attracting the attention of the crowd. The second promotes a mixture of content that is at the same time fresh and getting popular. The third simply promotes top popular content.

Advantageously, the Reasoner can infer 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 nation-wide. 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.

3.4.2 Domain-knowledge based Iterative Rules

Given the symptom events reporting the interesting URLs detected by continuous scalable algorithms running at the repository, the Reasoner uses some basic rules to diagnose whether the interesting URLs correspond to content URLs (thus worth recommending) or to Portal URLs (URLs that correspond to well known home pages, and thus not worth recommending).

A first set of rules, that we will detail in the next deliverable as part of the *Content vs Portal* analysis module, can be divided into two types. The first concerns rules that can be applied online immediately after the happening of an interesting URL detection event. The second are rules that can be applied only after an observation period for a given interesting URL. An example for the first family are rules that rely on the size or the structure of the interesting URL names. An example of the second type of rules are rules that rely on the history of the visits of a given interesting URLs (e.g. of rules: if a URL shows one day periodicity, then it is likely to be a portal). An example of

decision rule that is responsible for filtering out interesting URLs corresponding to portals and can take decisions online is structured as follows:

$$\text{if} \left\{ \begin{array}{l} \texttt{Interesting\_URL\_length}: \\ \texttt{URL length above threshold} \\ \& \\ \texttt{!Interesting\_URL\_path}: \\ \texttt{URL has no path} \end{array} \right\} \rightarrow \begin{array}{l} \texttt{Interesting URL:} \\ \texttt{Portal-URL.} \end{array}$$

Additional rules, need to be applied then following the event of detecting a content-URL. These rules are needed to decide which content-URLs to promote to the final promotion web page. These rules are simple at this point of the work, and simply rely on assigning scores to each content-URL taking into account, either their popularity, their freshness, their type (news, blogs) or a combination of all these criteria. An example of decision rule that is responsible for detecting content-URLs corresponding to news articles is structured as follows:

$$\text{if} \left\{ \begin{array}{l} \texttt{Content\_URL\_News}: \\ \texttt{URL's hostname is news portal} \end{array} \right\} \rightarrow \begin{array}{l} \texttt{Content-URL:} \\ \texttt{News-URL.} \end{array}$$

3.4.3 Diagnosis/Iterative Analysis Graph

Diagnosing content URLs, and discarding Portal URLs, can be represented by the simple diagnosis graph depicted in Fig. 9

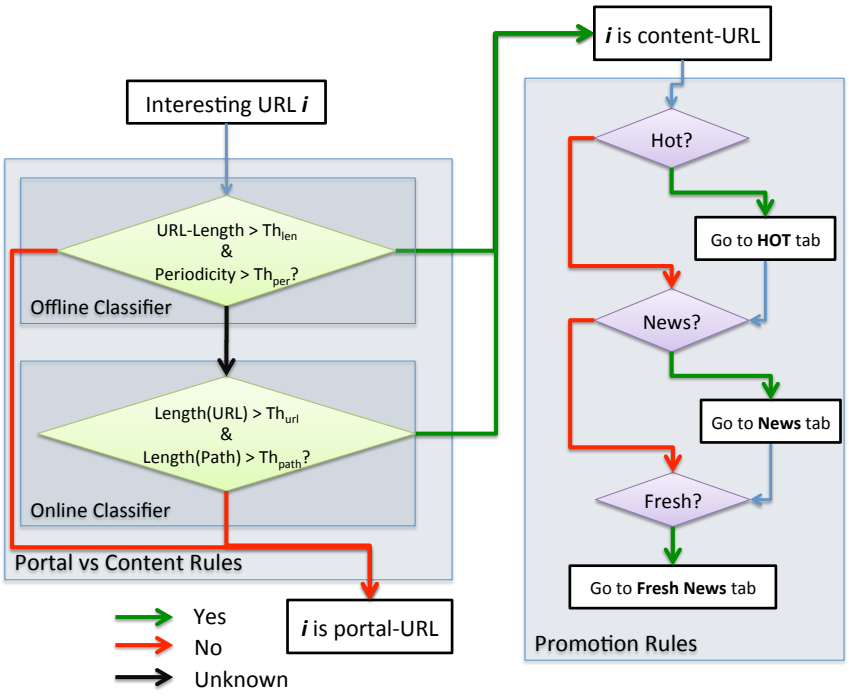


Figure 9: Interesting URL diagnosis graph.

The diagnosis graph takes as an input a stream of symptom events containing interesting URLs and outputs, first, diagnosis events as <interesting-URLs, label (content-URL/portal-URL)> and, if

labeled as content, the graph runs further analysis modules to understand the nature of the content. Depending on the outcome, a diagnosis event is generated to drive the URL to the most suitable curation platform section (e.g., a website tab or page).

As explained above, the rules to distinguish portals from content-URLs can be divided in two categories: those that rely on the history of visits for a given URL – typically more accurate –, and those that analyze the structure of the URL (URL length and path) – usually less precise. The first rules require to collect observations for a few days before being able to correctly take a decision, while the latter can be applied to take decisions on-the-fly. Thus, we split the *Portal vs Content* classification workflow into two steps, as depicted in Fig. 9 in which we report an example of the diagnosis graph for this use case. First, we run the rules composing the *Offline Classifier*, which collects observations about all detected interesting URLs and exploit this "knowledge" to distinguish URLs corresponding to portals to those pointing to actual content. In a nutshell, as soon as the interesting-URL i has been observed enough (W) time (each observation corresponding to the event of a distinct visit to the URL i), i can be classified using the offline domain knowledge rules (which is the most accurate since more informed), and the outcome c_i is stored in the *Knowledge Database*, **K**. When the information about the URL contained in **K** is too poor, the Offline classifier returns an "unknown" outcome, and the diagnosis graph relies on a simpler, but less accurate, set of classification rules. The *Online Classifier* in Fig. 9 performs classification on-the-fly based on the rules which do not need any knowledge to execute. As shown above, these are the length of the interesting URL and the presence of a path. These base on the intuition that URLs pointing to portals are usually shorter and do not contain a path.

Therefore, for every interesting-URL i, we first rely on the *Offline Portal vs Content classifier* result, if available, i.e., we verify the presence of i in **K**. Otherwise, we use the faster, but less precise, *Online classifier* result.

Finally, once we get the outcome on this classification, interesting URLs pointing to portals are discarded, while remaining content-URLs undergo the *Promotion Rules* branch of the diagnosis graph. The objective of this further analysis is to infer the most suitable promotion section in the content curation platform, that could be, e.g., a website containing multiple per-content-kind tabs or pages. The promotion rules investigate the nature of the content by using different means, and generate diagnosis events which will be processed by the back-end of the content curation platform. For instance, we rely on a slightly modified version of the promotion rule employed by Reddit to determine which contents are "Hot" in the network, and we distinguish URLs pointing to news articles comparing their hostnames against a pre-compiled set of news portals (see the example above). Finally, if a URL is classified as news and it has never been seen before, it is labeled as Fresh.

4 Conclusions

This deliverable presented the design and specification of the different logical components of the mPlane Reasoner. The Reasoner-relevant concepts were exemplified through the realization of the iterative analysis process for selected use cases, such that the reader can get a more tangible picture of how the overall mPlane Reasoner approach applies in the practice. Along with these specific Reasoner instantiations, the deliverable provided a set of domain knowledge based rules which give 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 introduced (in the appendix) different learning approaches for extending and/or generating new analysis rules within the Knowledge Structure.

The design and specification of the Reasoner would be complemented with the overall, final release of the complete Supervisor components, including all the necessary interfaces. This is planned for the final WP4 deliverable, D4.4. Still, the core of the system as presented in this deliverable already brings all the different logical components required for the complete mPlane to operate as envisioned.

A 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 subspace clustering techniques as well as an approach for exploratory data analysis based on association rule mining techniques. 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.

A.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.

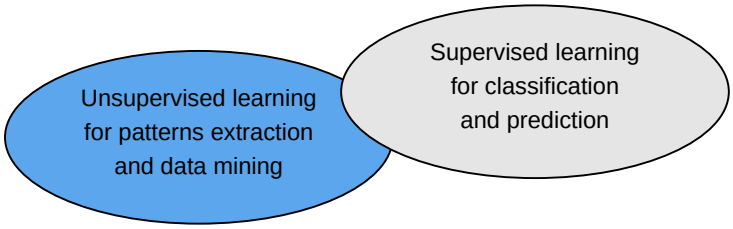


Figure 10: Different learning approaches for extracting useful information and detect the occurrence of specific patterns in large amounts of measurements.

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. In particular, we devised a robust multi-clustering algorithm based on a combination of Sub-Space Clustering (SSC) [26], Density-based Clustering [11], and Evidence Accumulation Clustering (EAC) [13] 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.

A.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 [26] 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 aggregating traffic measurements). Unfortunately, even if hundreds of clustering algorithms exist [17], it is very difficult to find a single one that can handle all types of cluster shapes and sizes.

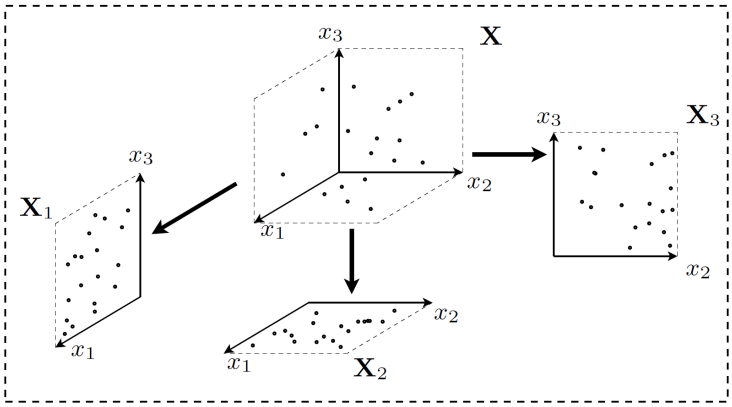


Figure 11: Sub-Space Clustering: 2-dimensional sub-spaces $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$ are obtain from a 3-dimensional feature space $\mathbf{X}$ by simple projection. Units in the graph are irrelevant.

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) [26] to produce multiple data partitions, doing density-based clustering in N different sub-spaces $\mathbf{X}_i$ of the original space.

A.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. 11 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 [17], 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 [17], 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 [13], 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.

A.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 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

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})

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) [16], 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.

A.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

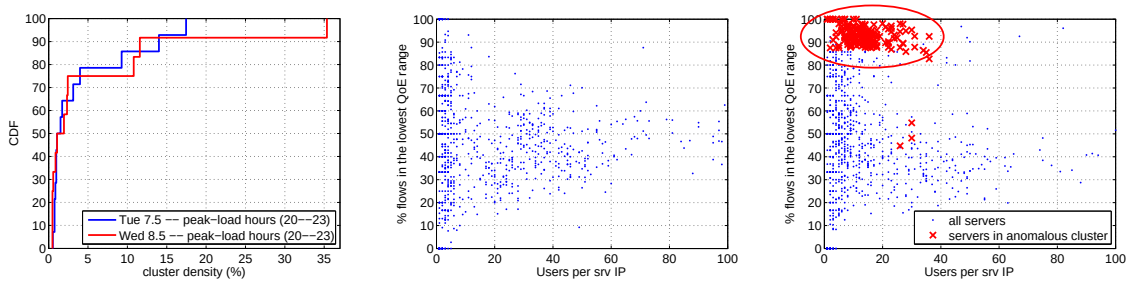
to analyze. δ_i is set as a fraction of the average distance between patterns in sub-space $\mathbf{X}_i$, which is estimated from 10% of the patterns, randomly selected. This permits to fast-up computations. The weighting factor w_i is used as an outlier-boosting parameter, as it gives more relevance 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 $\mathrm{d_M}$ 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.

A.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. 12(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. 12(b) and 12(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 12: Unsupervised detection of the anomaly through clustering. There is a clear shift in the cluster density during the hours of the anomaly.

A.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.

A.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 [37, 36, 23, 14] and with a distributed architecture [27, 10, 22, 38]. 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.

A.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 $\mathrm{c}(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$ [25].
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 [25], 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 [25]

$$\mathrm{lift(X,Y)} = \frac{\mathrm{c}(X \rightarrow Y)}{\mathrm{s(Y)}} = \frac{\mathrm{s}(X \rightarrow Y)}{\mathrm{s(X)s(Y)}} \tag{A.1}$$

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.

A.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

A.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, 24], 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 [24]. 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.

A.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 2.

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 2: Pre-processing example

A.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 3 and 4. Table 3 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 4, as obtained by the sample transactions in Table 3.

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 3: 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 4: Sample items

A.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 4, a result of this job is reported in Table 5, 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 5: Sample itemsets

A.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 A.2.2. Table 6 reports the rules extracted from the itemsets of the running example (see Table 5).

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 6: Sample rules

A.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 A.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 7.

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 7: Sample rules, sorted and aggregated

A.3 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.

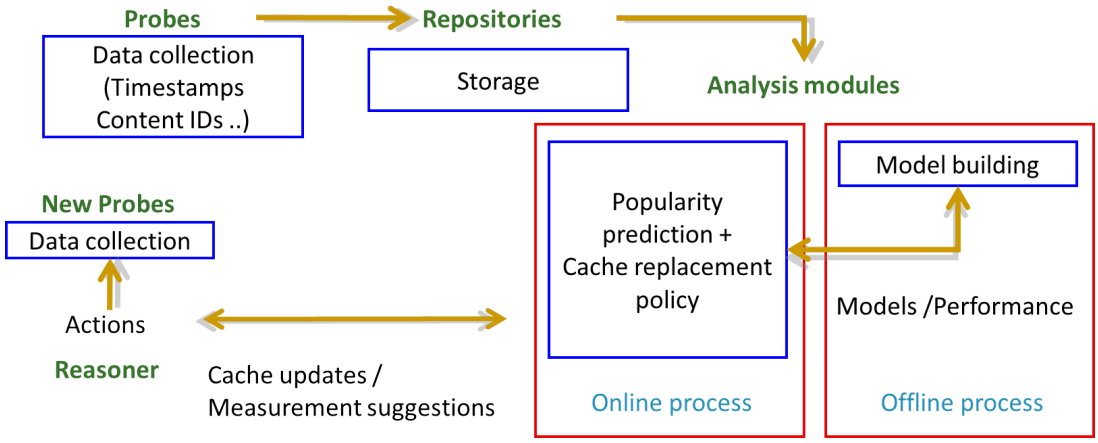


Figure 13: Correlating offline and online data in the use case "Estimating content and service popularity for network optimization".

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. 13 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.

A.4 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. 14, 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. 14 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.

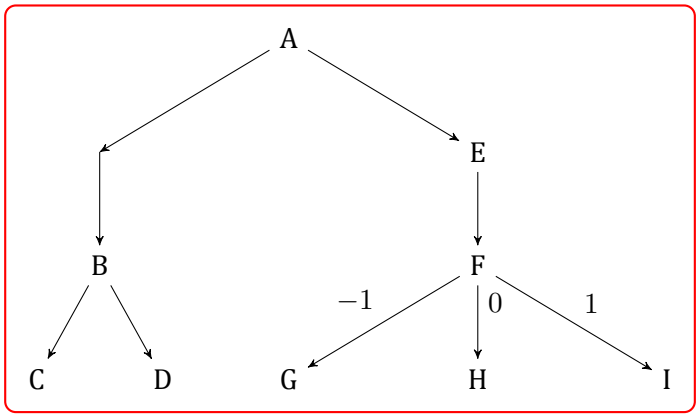


Figure 14: Example of diagnosis and measurement scheduling tree

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.

A.4.1 Scheduling tradeoff

We assume for simplicity that measurements have no conditional execution[3], and represent three scheduling strategies as shown in Fig. 15.

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.

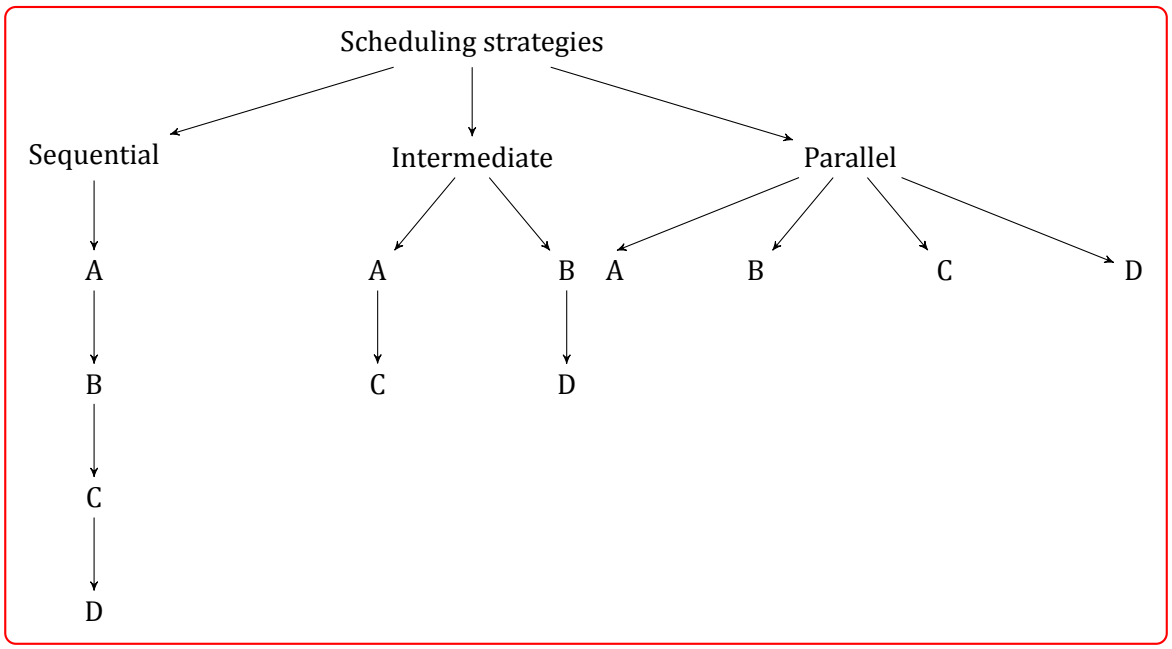


Figure 15: Scheduling tradeoff

[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. 14 are encoded with the ternary operator $condition?ifclause : elseclause$).

A.4.2 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. 15).

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.

A.4.3 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 [15, 34], vs. Probe Rate [18, 28]) models, but there are also studies that compare the precision and accuracy of different bandwidth measurement techniques [32, 21, 33, 31]. 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

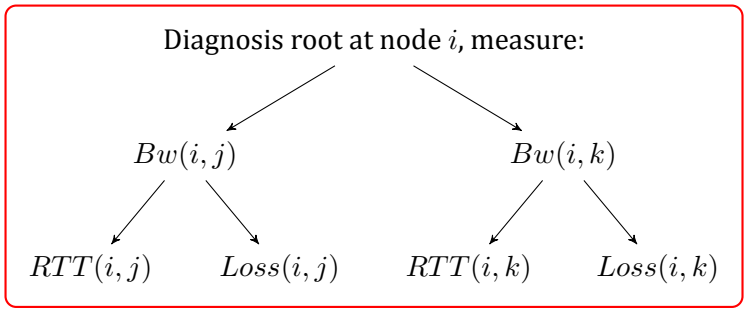


Figure 16: Interaction of homogeneous measurements: $Bw(i,j)$ and $Bw(i,k)$ may mutually interfere

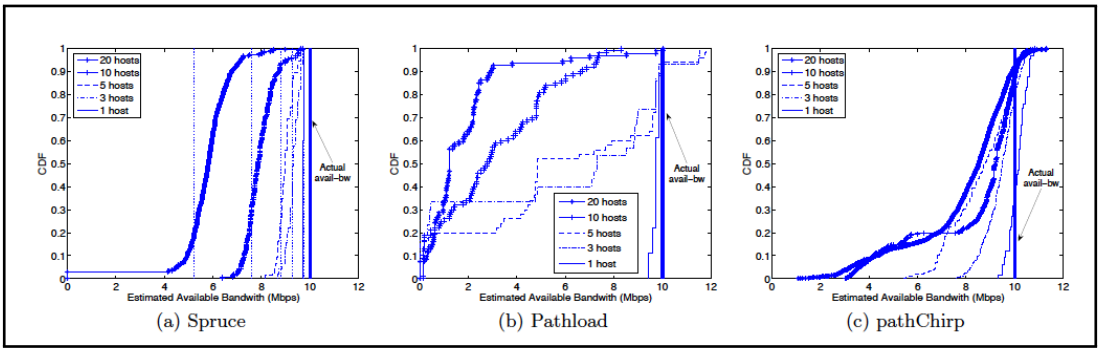


Figure 17: Interference between available bandwidth estimation tools (courtesy of [8])

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 A.4.2. Simultaneous measurements of a bottleneck link from a number of hosts [8] show that the accuracy of current available bandwidth estimation tools, namely Spruce [34], Pathload [18], and PathChirp [28] drops significantly due to mutual interference. As shown in Fig. 17, 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. 16 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-

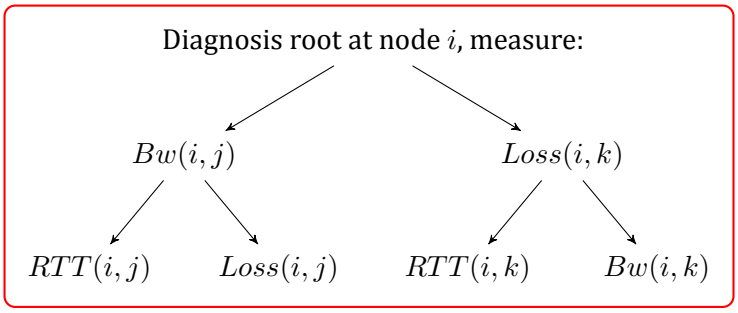


Figure 18: Interaction of heterogeneous measurement: $RTT(i,j)$ may be affected by $Bw(i,k)$

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.

A.4.4 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. 18.

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-

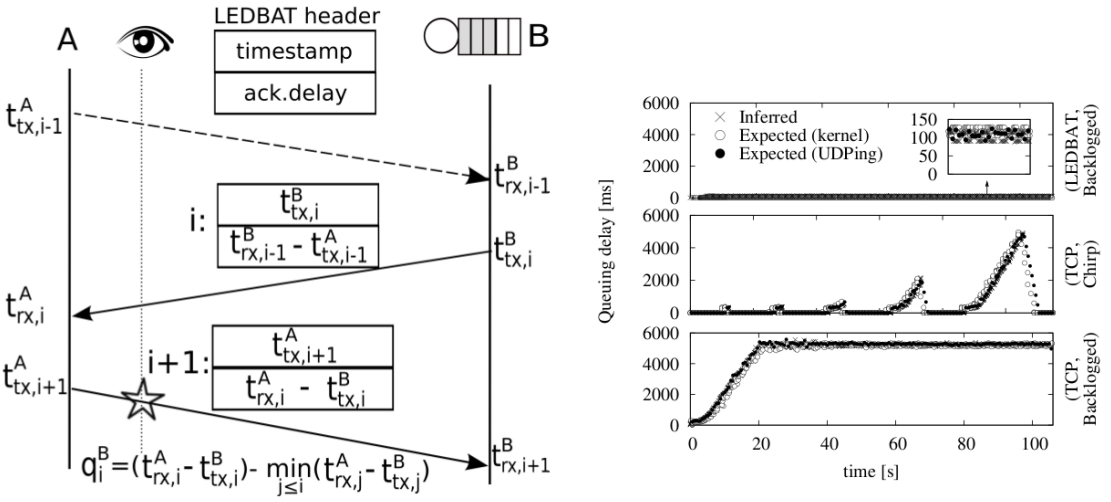


Figure 19: Interference of latency and load measurement: methodology to infer queueing delay (left) and validation with controlled background traffic (right).

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. 18, $RTT(i,k)$ 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. 19. This is a classic approach used in congestion control to drive congestion window dynamics, starting from Jain's pioneering work in the late 80s [19], to the widely known TCP Vegas [2] in the late 90s, to ultimately the LEDBAT [30] 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[20], 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,

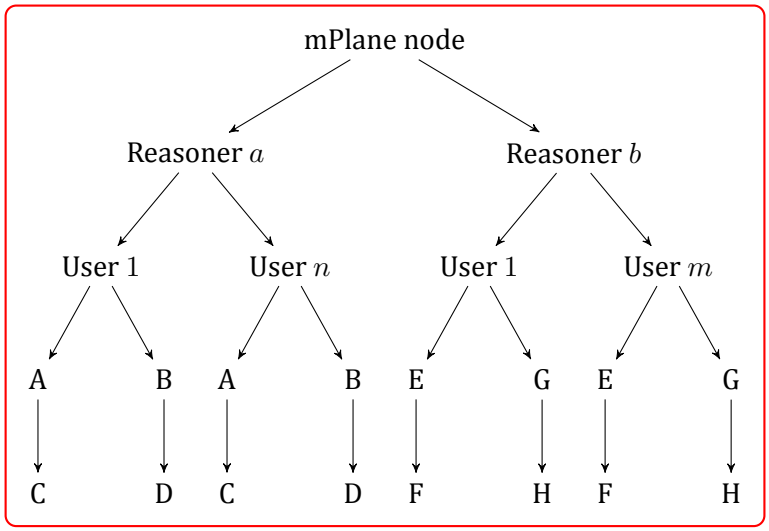


Figure 20: Multiple levels of dependencies: per-user, per-reasoner, per-node

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. 19, 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.

A.4.5 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. 20. 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.

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] R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

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

[14] 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.

[15] 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.

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

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

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

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

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

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

[22] 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.

[23] 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.

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

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

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

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

[28] 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.

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

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

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

[32] 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.

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

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

[35] 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.

[36] 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.

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

[38] 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.