mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Cross-Check of Analysis Modules and Reasoner Interactions

Author(s):	Author names
POLITO	U. Manferdini, S. Traverso, M. Mellia
FUB	E. Tego, F. Matera
ALBLF	Z. Ben Houidi
EURECOM	M. Milanesio, P. Michiardi
ENST	D. Rossi, D. Cicalese, D. Joumblatt, J. Auge
NEC	M. Dusi, S. Nikitaki, M. Ahmed
TID	I. Leontiadis, L. Baltrunas, M. Varvello
FTW	P. Casas, A. D'Alconzo
ULG (editor)	B. Donnet, W. Du, G. Leduc, Y. Liao
TI	A. Capello, F. Invernizzi
A-LBELL	D. Papadimitriou

Document Number:	D4.3
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 presents an extended set of Analysis Modules, including both the improvements done to those presented in deliverable D4.1 as well as the new analysis algorithms designed and developed to address use-cases. The deliverable also describes a complete workflow description for the different use-cases, including both stream processing for real-time monitoring applications as well as batch processing for "off-line" analysis. This workflow description specifies the iterative interaction loop between WP2, WP3, T4.1, and T4.2, thereby allowing for a cross-checking of the analysis modules and the reasoner interactions.

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

mPlane consists of a Distributed Measurement Infrastructure to perform active, passive and hybrid measurements. It operates at a wide variety of scales and dynamically supports new functionality.

The mPlane infrastructure is made of several components: The *Measurement component* provides a geographically distributed network monitoring infrastructure through active and passive measurements. The *Repository and data analysis component* is in charge of storing the large amount of data collected and processing it prior to later analysis. The *Analysis component* provided by the supervision layer allows mPlane to extract more elaborated and useful information from the gathered and pre-processed measurements. Finally, the *Reasoner component* is the mPlane intelligence. It allows for structured, iterative, and automated analysis of the measurements and intermediate analysis results. It orchestrates the measurements and the analysis performed by the probes, the large-scale analysis repositories and the analysis algorithms, actuating through the Supervisor to interconnect with the other mPlane components.

In this deliverable, we update analysis algorithms provided in Deliverable D4.1 [73] (Chapter 2). In particular, we inspect each use cases and discuss improvements and validation results obtained since the early stages of the mPlane project. In addition, we provide an insight into more generic analysis algorithms that are related to network topology and routing.

We next show how each algorithm behaves in the whole mPlane workflow (Chapter 3), i.e., how various mPlane modules are used by each algorithm, thereby allowing for a cross-check of the analysis modules discussed in Chap. 2 and the reasoner interactions.

Finally, Chapter 4 concludes this deliverable.

2 Analysis Module

In this chapter, we provide improvements to algorithms first provided in mPlane deliverable D4.1 [73]. We also provide new analysis algorithms designed and developed to address use-cases, and additional generic algorithms.

The first series of proposed analysis algorithms are presented in the context of specific use-cases. They enable to:

- find the cause of Quality of Experience (QoE) degradations,
- estimate the future popularity trends of services and contents for network optimization,
- classify and promote interesting web content to end-users,
- assess and troubleshoot performance and quality of multimedia stream delivery,
- diagnose performance issues in web and identify the segment that is responsible for the quality of experience degradation,
- find root cause of problems related to connectivity and poor quality of experience on mobile devices,
- detect and diagnose anomalies in Internet-scale services (e.g., CDN-based services),
- verify SLAs.

The second series of analysis algorithms have a generic nature and are therefore presented separately. They enable to :

- find measurement probes near some point of interest in the network,
- discover network topology (presence of MPLS tunnels and of middleboxes, inference of IGP weights),
- detect anycast services, enumerate and geolocalize the replicas.

These algorithms thus cover a large range of functionalities, such as:

- classification and filtering (e.g., of flows, applications, content),
- estimation/prediction (e.g., of Quality of Experience (QoE), popularity, path metrics, topology),
- detection (e.g., of anomalies, threshold-based changes, interfering middleboxes, hidden relationships between policy rules),
- correlations (e.g. between measurements and QoE, traffic directions and caches/servers),
- diagnosis (e.g., of QoE or web degradation, lack of connectivity).

2.1 Supporting DaaS Troubleshooting

2.1.1 Use Case Reminder

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.

2.1.2 Statistical Classification Module

The role of the reasoner in this use case is to combine the information about the kind of application on top of a RDP connection given by the statistical classification module, with the information about the delay on the end-to-end path, so to instrument the network on the troubleshooting action to take to overcome poor QoE issues. Detecting the kind of application on top of the connection is therefore key.

In D4.1 [73], we presented several analysis algorithms we considered to detect the application on top of a given thin-client connection. The main goal there was the design and tuning of an effective statistical classification technique which can effectively take advantage of the available features provided by the mPlane probes.

In D3.3 [73], we described the robustness of the statistical classification technique based on SVM to network conditions, when the training has been done without considering any network impairments, whereas the testing includes traces with impairments such as packet loss and packet delay.

In this deliverable we collected all the previous results and carried out additional evaluations, to determine the algorithms and the classification parameters that allow us to achieve best accuracy, also considering the variety of network impairments that we introduced when collecting the dataset used for training and testing.

We provide a detailed description of the technique and its application to inferring users' QoE in [29]. Details about the testbed are provided in D5.1 [73].

2.1.3 Results and Lessons Learned

Our results support the idea that considering the SVM algorithm to classify Remote-Desktop-Protocol (RDP) connections over a ten-second time-window allows to infer with high accuracy (up to 99% in some cases) the class of applications that is running over the thin-client connection. Table 2.1 reports on the results achieved with the SVM algorithm on a ten-second time-window. Given that, we focused on the SVM algorithm only for the following experiments.

We investigated how the accuracy of the statistical classification techniques changes when the training conditions differ from the testing ones. Table 2.2 and Table 2.3 report on the composition of our training set and testing set, respectively.

Our experiments show that the accuracy in detecting flows carrying *Video* drops by 60% in terms of epochs (40% of bytes) when traces used for training are collected only under the optimal case,

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

Table 2.1: Classification results (percentage) for the SVM algorithm with a ten-second time window. On the left, accuracy by epoch. On the right, accuracy by byte.

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

Table 2.2: Training set composition. AdR stands for adobe reader, while PPT for Powerpoint presentations. VLC is marked as VLCa and VLCv when used for generating audio and multimedia content, respectively. Same consideration holds for WMP.

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

Table 2.3: Testing set composition. WebF stands for web pages with embedded Flash videos, WebB for other kinds of web pages. VLC is marked as VLCa and VLCv when used for generating audio and multimedia content, respectively. Same consideration holds for WMP.

i.e., without introducing any impairment on the network, whereas traces used for testing experience a bandwidth squeezing of a factor of four (from a downlink/uplink rate 6Mbps/1Mbps to 1.5Mbps/256Kbps).

We believe this result is important for two reasons. First, it practically shows the rate by which the accuracy of SVM decreases as we force the technique to classify traffic for which it did not receive any training, both in terms of application and network conditions. Second, it points out that the misclassified epochs are actually the ones that carry few bytes, which means that the bandwidth squeezing we apply is so that it alters the values of the features upon which we trained our classifier, thus altering the behavior of the application inside the thin-client connection, e.g., the multimedia streaming is bursty. As long as the conditions return similar to the optimal ones, such as in the case where the testing traces have a downlink/uplink rate of 3Mbps/512Kbps, the classifier can keep up and is still able to detect the *Video* category with an accuracy of bytes around 80%.

To prove this thesis, we further analyzed whether there is room for improvement in the classification of the testing set in case also the training set includes RDP sessions collected under some forms of network impairments, such as with different bandwidth conditions. Preliminary tests show that the accuracy increases on average of 25% (15%) in terms of epochs (bytes) for multimedia content. It is worth noting that although these results may justify a motivated service provider to collect training traces in different network conditions to achieve better accuracy, they open to the potential over-specialization (over-fitting) of the training set against the testing set.

2.2 Estimating Content and Service Popularity for Network Optimization

2.2.1 Use Case Reminder

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

2.2.2 An Almost Reasoner-less Approach

Differently from other use cases that include troubleshooting and where iterative reasoning is almost mandatory, the role of Reasoner is basic for the content popularity estimation use case. Indeed it orchestrates the two different analysis modules that monitor and estimate the popularity evolution of contents observed in certain portions of the network. The only reasoning task which may require some iteration is the identification of the network portions in which contents are labelled as potentially popular.

In its current status, the Reasoner gets the list of expected popular contents from the analysis modules, that run continuously on the repositories, together with information about the network portion (i.e., the probe) in which such content was observed. In the scenario of a hiercarchical Content

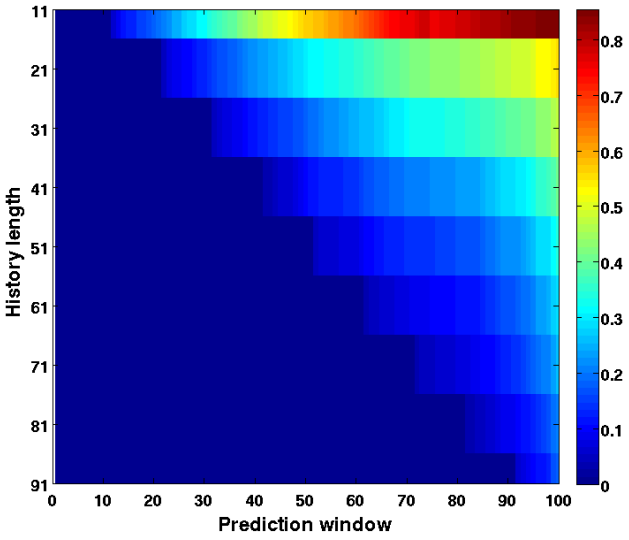


Figure 2.1: Mean percentage reconstruction error when predicting the future popularity of the video based on the history length (bins represent days). Note how the reconstruction error increases as we predict further in the future.

Delivery Network, it would be useful to get the additional knowledge about the location of the contents that are expected to be more popular. Consequently, this information could be utilized in order to identify the caching level in the hierarcical CDN at which proactively prefetching the corresponding content.

2.2.3 Preliminary Evaluation Results

Here we describe the results in inferring the evolution of content requests over time by means of the predictor as described in D4.1 [73]. In particular, we run our prototype on a commercial ISP anonymized trace reporting the requests to YouTube videos watched by a population of 28,000 users.

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

The goal of this first evaluation is to assess the ability of the technique to accurately predict the popularity of a given content. We believe this is the first step into the implementation of caching strategies: it constitues the input for optimizing the cache usage given its size, and for reducing the amount of bytes that we have to transfer if required to be transfered when we know which content should be kept into the cache as it will become popular in the future.

We started investigating the accuracy of our algorithm in predicting the future popularity of the videos, given that the algorithm has seen the first X data samples of the requests of the videos over time, where X ranges from 11 to 91, with step 5. Fig. 2.1 reports the results in terms of Mean Percentage Error (MPE), that is the ratio between the absolute estimation error (the difference

between the estimated and the real requests) and the number of requests that the video actually gets in the data sample we are trying to predict.

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

As future analysis, we plan to assess how the accuracy of the prediction algorithm brings benefit to the overall reduction of the amount of traffic that has to be transfered over the network across different levels of cache.

2.2.4 Other Approaches

In our ongoing efforts we are analyzing other prediction approaches and evaluating their scalability with respect to the current implementation.

2.3 Passive Content Curation

2.3.1 Use Case Reminder

We remind that the content curation use case aims at providing a service that helps users identifying, fast, relevant content in the web. This use case monitors various probes in the network, detects URL clicks (called *user-URLs*) out of the streams of HTTP logs observed on the probes, and performs some analysis on these clicks in order to pinpoint the set of URLs that are worth recommending to users.

In D4.1 [73], we presented two offline analysis modules that aim at detecting (1) *user-URLs* (URLs that were clicked intentionally by users) out of a stream of HTTP logs, and (2) *interesting-URLs* (user-URLs that are likely to be interesting to recommend to users).

In D3.3 [73], we enhanced our user-URLs detection heuristics and modified them to run online at high rates of HTTP requests (up to 5 million per hour). Since this algorithm needs to run continuously on HTTP logs streamed from the mPlane probe, we decided to make it a scalable data-analysis algorithm that runs on the repository instead of an analysis module on its own. The output of this algorithm will be user-URLs, together with their timestamps, referrer and a flag saying wether they contain or not a social plugin.

In D4.2 [73], we sketched how we modified the structure of the analysis modules to work online, and we present in this document two new analysis modules which rely on the output provided by WP3: (1) the content versus Portal module and (2) the content promotion module.

2.3.2 Content versus Portal Analysis Module

As also described in Sec. 3.3, our use case needs a *Content versus Portal* module which focuses on discriminating interesting-URLs corresponding to web portals from those pointing to specific con-

tent. We use the term *web portal* or *portal-URL* to refer to the front page of content providers, which mostly has links to different pieces of content (e.g., `nytimes.com/` and `wikipedia.org/`); whereas a *content-URL* refers to the web page of, e.g., a single news or a wikipedia article. We thus engineer an analysis module that is a classifier that distinguishes between web portals and content-URLs.

2.3.2.1 Features

We use five features to capture both URL characteristics and the arrival process of visits users generate.

URL length. This feature corresponds to the number of characters in the URL. Intuitively, portal-URLs tend to be shorter than content-URLs.

Hostname. This is a binary feature. It is set to one if the resource in the URL has no path (i.e., it is equal to "/"); and to zero, otherwise. Usually, requests to portal-URLs have no path in the resource field.

Frequency as hostname. This feature counts the number of times a URL appears as root of other interesting-URLs. The higher the frequency, the higher the chances that the URL is a portal.

Request Arrival Process (RAP) cross-correlation. The URL request arrival process is modelled as a vector in which each element counts the number of visits in five-minute bins. We noticed that users often visit portal-URLs following some diurnal periodic pattern. Intuitively, the more the request arrival process signal of a given URLs is "similar" to that of a well known portal, the higher the chances that the URL corresponds to a portal. To capture such a similarity, we compute the maximum cross-correlation between (1) the request arrival process of a tested URL and that of (2) well-known portals (e.g., `www.google.com` or `www.facebook.com`). The cross-correlation, a well known operation in signal processing theory, measures how similar the two signals are, as a function of a sliding time lag applied to one of them. The higher the value of the maximum of the cross-correlation, the larger the chance of a URL being a portal.

Periodicity. This is a binary feature that captures the fact that users visit portals with some periodicity. We use the Fast Fourier Transform (FFT) on the discretized aggregate visit arrival process for a given URL. If the visit arrival process shows one-day periodicity (principal frequency of 1/24h), then we set periodicity to one; zero, otherwise.

Note that the last two features would intuitively work only for popular-enough portals that get enough clicks to exhibit the periodic diurnal cycle. However, as we will see in Sec. 2.3.3, we only promote content that has captured a sufficient amount of attention. As such, unpopular portals are less likely to be promoted.

Finally, we opt for a supervised machine learning approach to build a classifier based on the above features. Given the heterogeneity of the URL characteristics, we choose the Naive Bayes classifier. This classifier is simple and fast, which is important for online implementation. As we will see, it achieves good performance, not calling for more advanced classifiers.

2.3.2.2 Accuracy

To evaluate the accuracy, we observe a stream of interesting-URLs (extracted from http logs of a commercial ISP) and by visiting such URLs, we manually pick among them 100 *content-URLs* and 100 *portal-URLs*. We use this set for both training and testing. In particular, we randomly divide the 200 URLs into two sets: two thirds of the URLs for training and one third for testing. We use a ten-fold validation, averaging results from 10 independent runs.

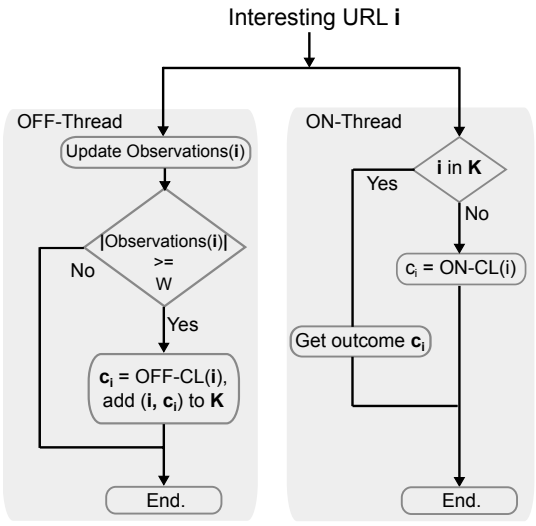


Figure 2.2: Workflow of the online content vs. portal URL classifier

We build several combinations of features and choose the one that shows the best tradeoff between precision and recall. Our results show that we achieve the best accuracy by combining the URL length and the periodicity. This combination achieves 100% precision and 93% recall for web portals, and 94% precision and 100% recall for content-URLs, which translates into an overall accuracy of 96%. This means that we have a very high precision whenever we tag a URL as Portal. However, around 6% of what we tag as content-URLs are in reality portals.

2.3.2.3 Online Classifier

Our analysis module needs to run online, that is to decide as soon as it is seen whether an interesting-URL is a content-URL or a portal-URL. We now describe the online version of the classifier we use to distinguish content-URLs from portal-URLs. It takes as an input a stream of tuples <interesting-URLs, timestamps>, and it outputs binary labels as <interesting-URLs, label (content-URL/portal-URL)>.

Despite we achieve the best accuracy when combining the features URL-length and periodicity, this latter complicates the design of the online classifier, as it has to observe each interesting-URL for some time, e.g., a week. Therefore, we design the online classifier based on the observation that the features can be split in two categories: those that analyze the structure of the URL (URL Length and hostname), and those that rely on the visit arrival process (RAP cross-correlation, periodicity and frequency as hostname). For the first set we can take decisions on-the-fly, while the latter requires to collect observations for a few days before being able to correctly take a decision. Thus, we split the classification workflow into two threads, as depicted in Fig. 2.2. The left thread, named *OFF-Thread*, collects observations for the features RAP cross-correlation, periodicity and frequency as hostname. As soon as the interesting-URL i has been observed for enough (W) times (being each observation corresponding to a user visit to i), i is classified using features URL-length and periodicity (the most accurate in our experiments, method *OFF-CL* in Fig. 2.2), and the outcome c_i is stored in the *Knowledge Database*, **K**. The right thread, named *ON-Thread*, performs classification on-the-fly based on the per-URL "online" features (method *ON-CL* in Fig. 2.2). In particular, we test our classifier on a groundtruth trace that we build by visiting the top 100 websites in Alexa ranking. We observe that combining URL length and hostname gives us a 93% accuracy on the groundtruth

trace.

Therefore, for every interesting-URL i, we first rely on the *OFF-CL* result, if available, i.e., we verify the presence of i in **K**. Otherwise, we use the faster, but less precise, *ON-CL* result.

2.3.3 Content Promotion Analysis Module

This analysis module is useful to decide which content-URLs to promote and show to the users, it takes as an input content-URLs, computed by the previous analysis module. In the current status of the use case, we test four different promotion mechanisms to use for as many sections (or tabs) in the front end website that we built for a first prototype of our use case.
Hot. This mechanism is an adaptation of Reddit's Hot ranking algorithm [109], which promotes URLs that are both popular and recent. The algorithm behind Reddit's Hot ranking assigns each URL a score based on users' votes. We replace such votes with the number of visits, and modify Reddit's formula to obtain the following:

$$Score = log(N_{views}) + \frac{T_{first} - T_{start}}{T_P}$$

N_{views} reports the number of views, T_{first} is the time corresponding to the first time the content-URL has been observed (i.e., visited), and T_{start} is the time corresponding to an absolute reference, i.e., the start of the system. Finally, T_P represents the normalization factor that we use to define a "freshness period", and that we set to 12 hours. Intuitively, this formula finds a balance between the content popularity (the number of views) and its freshness (its age with respect to the absolute reference). When a content-URL stops getting attention, its ranking starts decreasing due to its age.
Top. This mechanism produces a simple ranking of URLs depending on the number of views. In the current version, in order to keep the memory usage of the system steady, this ranking accounts only for one week of history.
Fresh news. This mechanism focuses on one category of content-URLs, namely news, and aims at promoting the freshest news seen in the network. In order to detect if a content-URL corresponds to a news, we rely on a predefined list of traditional news media websites in Italy: if the hostname of a content-URL belongs to this list, and if the content-URL has never been seen before, we tag it as news and promote it as fresh news. We construct the list of news websites by observing Google News, a popular news aggregation and indexing system that uses an active approach based on web-crawlers to collect and promote news. More precisely, Google servers regularly query a predefined set of popular news portals, looking for new articles to push on the front page [38]. To construct the list of news portals, we crawl the Google News front page every 20 minutes for a period of one week, looking for new websites. This allowed us to obtain more than 500 distinct news websites.
Live news stream. It simply promotes the news, as defined earlier, as soon as they get attention from users in the network, i.e., in a continuous manner.

Finally, when a content-URL receives a visit, this analysis module updates the score of the URL and its number of views. The content promotion analysis module periodically recomputes the ranking and updates, if necessary, the database used to store the contents in the Hot and Top categories.

2.3.4 Preliminary Prototype and Online Deployment

We build a preliminary prototype of this use case.

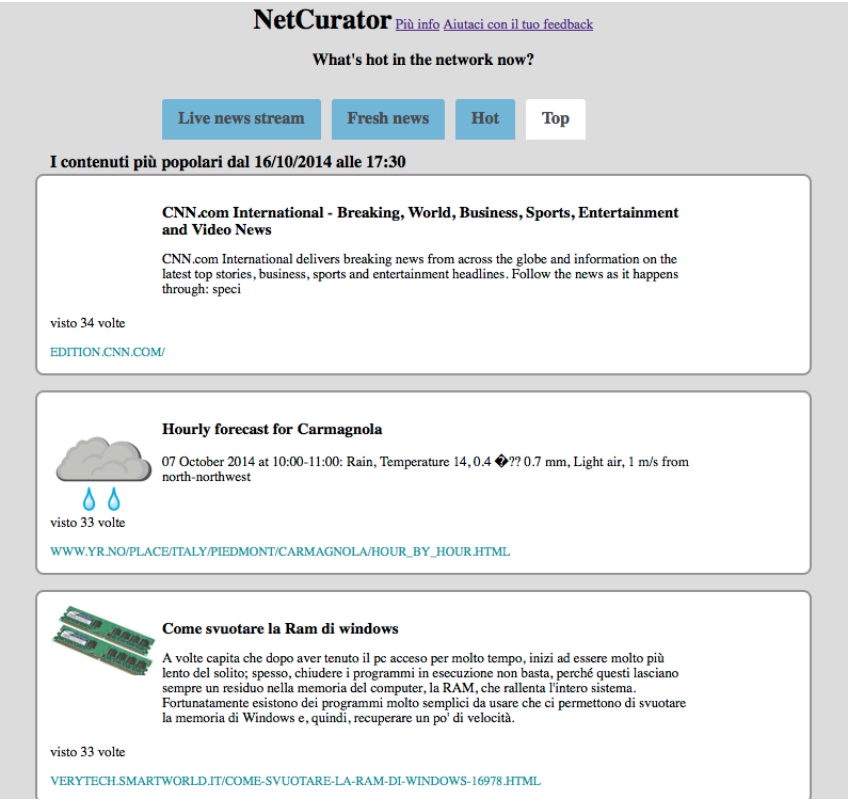


Figure 2.3: Screenshot of the website running the Content Curation prototype available at `http://tstat.polito.it/netcurator/`

By feeding the online algorithms described in previous sections with a live stream of HTTP requests that has been made available from Politecnico di Torino, we deploy a complete prototype of the system running in an actual operational network.

Thanks to a Tstat probe installed at the egress vantage point of the campus network and obtain HTTP requests generated by around 15,000 users, i.e., students and personnel regularly accessing the Internet. On average, users generate 7M HTTP requests per day. Out of these the system extracts on average 55,000 user-URLs, corresponding to 5,000 interesting-URLs. The backend server of the system receives the stream of HTTP requests and processes it to detect, first, interesting-URLs, then, content-URLs, and, finally, among those the URLs to promote on the website, that we named NetCurator: `http://webrowse.polito.it/`.

As depicted in Fig. 2.3, the website consists in four tabs, one for each promotion method. Each tab contains a content feed whose design is inspired by the "wall" implemented in popular social networks such as Facebook and Twitter, and URLs that make it to the feed are presented with a preview image, a title, and a description when available. In our ongoing efforts we are working to make all the system components mPlane-compliant.

2.3.5 Evaluation

This section evaluates various aspects of our prototype. First, to evaluate its efficiency in processing HTTP logs, we apply it on the passive HTTP log trace that have been collected in a commercial European ISP. This trace reports the HTTP activity of 19,000 users regularly accessing the Internet

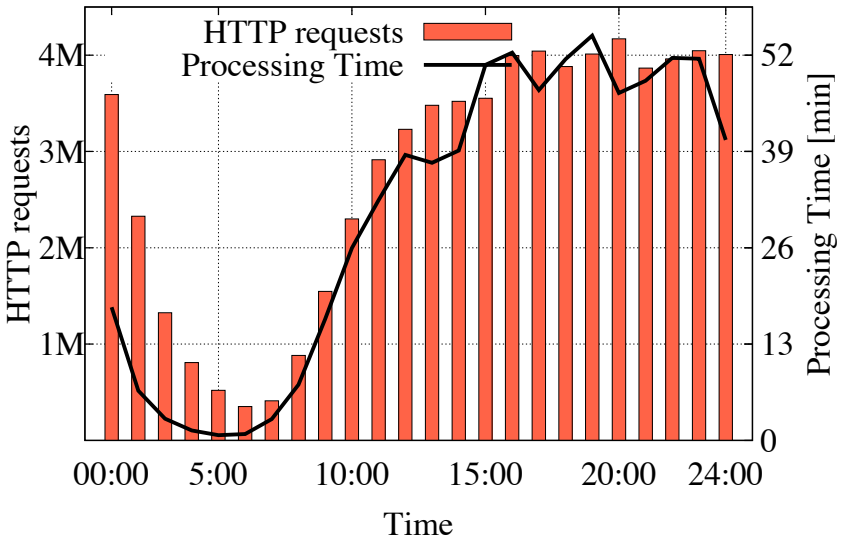


Figure 2.4: HTTP requests rate (left y axis) and processing time (right y axis) over time for one day extracted from *HTTP-ISP*

through ADSL/FTTH access technologies. The trace, that we name *HTTP-ISP*, reports in total more that 190M HTTP requests that users have generated over a period of three days of April 2014. As we described in D4.2, Sec. 2.3.5.1 evaluates the performance of the prototype when we feed it with the passive trace *HTTP-ISP*. Finally, Sec. 2.3.5.2 focuses on the Fresh News tab (see Sec. 2.3.4 for a detailed description) to see how efficient it is in detecting fresh news.

2.3.5.1 Performance

In order to understand the performance of the prototype, we evaluate how it behaves when it processes *HTTP-ISP*, whose per-hour rates of HTTP requests are much larger with respect to the live deployment we described in Sec. 2.3.4. We show how the prototype performs against the day in *HTTP-ISP* that has the largest peak hourly rate of HTTP requests. We split the one-day trace in 1-hour long sub-traces, and use them to feed our prototype. For each hour, we measure the time that the prototype spends to end the processing. For this experiment, we run the prototype on a six-core Intel 2.5GHz CPU and 32GB RAM server.

Fig. 2.4 reports the amount of HTTP requests (left-hand y axis) and the corresponding processing time (right-hand y axis), for all the 1-hour long bins in the day of *HTTP-ISP*. The figure shows that the implementation of the prototype is able to finish all the one-hour traces in less than one hour. This demonstrates that it can sustain the processing of such large rates of HTTP requests on a rather standard server like the one we picked above. Finally, we note that its memory footprint is minimal, as less than 5% of memory was used throughout the experiment.

2.3.5.2 How fast can our Content Curation system Detect Content?

Finally, this section assesses what we expected to be a nice property of our prototype when we designed it: fast discovery of Internet content. Indeed, if Google News has robots that actively look

for new content to index, our system can rely on an "army" of users who explore the web looking for fresh news. In this section, we set the bar high and leverage our online deployment to compare the two approaches and assess how fast is our content curation system in discovering news content. Since the content in both "Fresh news" and Google News relies on the same list of traditional news providers, the only difference between them is that ours uses the crowd of users to discover content, while Google's uses robots.

To compare the two approaches, each time our prototype promotes a content-URL to "Fresh news", we check if Google has already indexed it,[1] and if it has, we measure since when (this "age" is an information available below each link returned by Google Search). We run this experiment for a period of one day (after that, Google has banned our IP network because of the extensive probing).

We observe that despite the small sample of users that our prototype has in its current live deployment, it was able to find few not-yet indexed news URLs. Fig. 2.5 shows the number of not indexed news in each hour of the day. We put this number in perspective with the total number of "Fresh news" per hour. The figure shows, not surprisingly, that the more views we have, the higher the chances that we find not-yet-indexed content.

We conclude that with such a small number of users, our content curation system cannot compete on the speed of content discovery, especially when the space of news to discover is limited (a predefined list of news portals). Indeed, our analysis of the age of news shows that in 96% of the cases, our prototype is more than 1 hour later than Google News in discovering a news URL.

However, an interesting observation is the age of the news articles that are consumed by users in the network (and that are therefore promoted by the prototype). In fact, although platforms like Google News promote on their front page very fresh news (from few minutes to two hours old), our experiment shows that 44% of the consumed news were published one to several days before (according to when they were first indexed by Google). This means that although content freshness is important for news, a big portion of news article are still "consumable" one to few days after their publication.

2.4 Service Provider Decision Tree for Troubleshooting Use Cases

2.4.1 Use Case Overview

The reference scenario, as depicted in Fig. 2.6, is based on a typical SP's infrastructure. The SP has built and runs a completely private instantiation of the mPlane infrastructure. This means that all the probes, the repository, the supervisor and the reasoner are under the exclusive control of the SP.

In order to evaluate the QoE of a specific service (e.g., video streaming) a continuous monitoring infrastructure built upon a set of passive probes is used. Passive probes are placed in multiple vantage points and extract from traffic flows data useful to estimate the QoE. For instance, the average throughput per flow or the Round Trip Time can be usefully used to build performance parameters associated to the monitored service. Besides the set of passive probes, the reference scenario also considers the availability of a set of active probes. The active probes are located directly behind a subset of access nodes (e.g. DSLAMs, OLTs, etc.), within the PoP site and next to the Internet Gate-

[1] We use several instances of a headless browser to do a "site:" search on Google Search for each news-URL the prototype detects.

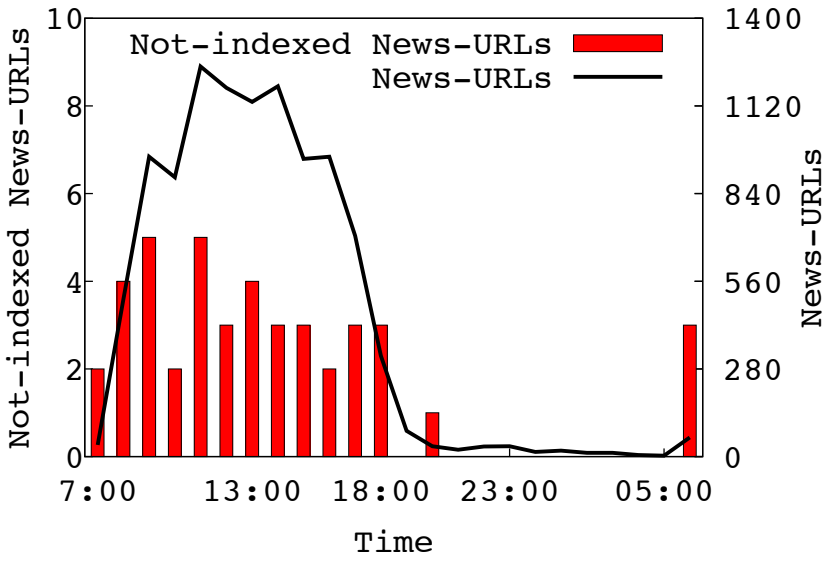


Figure 2.5: Number of not-indexed news-URLs (left y axis), and number of news-URLs (right y axis) during one day (Online deployment)

ways. These probes are able to perform active measurements: for instance, Ping/`traceroute` or actively requesting video contents and taking measures from Over The Top services (e.g. YouTube). The active probes work on demand, following triggers coming from the Reasoner (through the Supervisor).

2.4.2 Diagnosis Algorithm

Fig. 2.7 depicts the diagnosis algorithm.

The Analysis module is the trigger of the overall troubleshooting process. When the performance measures are below the threshold, a "NOT OK" response is sent to the Reasoner which starts the process to find the cause of the problem, according to the actions described in the graph. The algorithm is described in detail in deliverable D4.2 [73]. It is based on a very basic decision tree that highlights the iterative interaction with the probes. More sophisticated algorithms or machine learning approaches could be used to speed-up the process and reduce the communication with the probes. Anyway, at this stage, we prefer to propose an algorithm that is easy to understand for people working in network operation teams because it is based on an expert-driven approach, the same they use daily to solve the issues. In this way, the purpose of each step of the algorithm is clear. At the same time, the overall process, including the communication with the probes, is completely automatic, boosting the efficiency of the analysis.

Currently, part of the analysis module has been implemented. Specifically, the software that aggregates the QoE parameters measured by the passive probes is available. The aggregation is performed on a per-DSLAM basis, in order to have a sufficient granularity to trigger the Reasoner process. The QoE parameters taken into account are the bandwidth and the Round Trip Time (RTT). The passive probes calculate these quantities for each session and store the values into the Repository. The analysis module takes these values, aggregates them per-DSLAM, and calculates the average. The obtained result is compared to a pre-defined threshold: if the result is below the threshold,

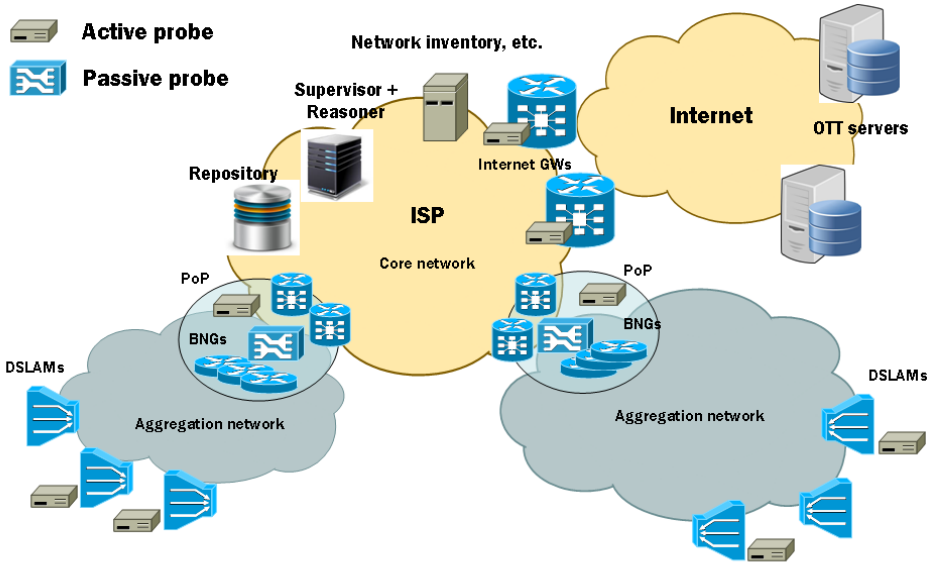


Figure 2.6: Reference scenario

the analysis module interprets it as a QoE degradation and triggers the Reasoner process to find the cause of the problem. An example of the average bandwidth per-DSLAM is shown in Fig. 2.8, where the value is calculated every 10 minutes. After each calculation, the analysis module sends a feedback to the Reasoner process: "OK", if all the DSLAMs are over the threshold, "NOT OK", if one or more DSLAMs are below the threshold. In both cases, the values are sent to the Reasoner.

Fig. 2.9 shows an example of RTT calculated from the sessions of a specific DSLAM. In this case, the values are classified also on the basis of the profile of the access line. This option can be enabled also for the bandwidth calculation.

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

2.5 Quality of Experience for Web browsing

2.5.1 Use Case Overview

The QoE for Web browsing use case aims at identifying the root cause of a poor performance in browsing (i.e., high Web page loading time). The identification of the root cause exploits measurements taken on distributed probes, both actively and passively. The probes passively records HTTP time measurements from an instrumented headless browser (i.e., phantomJS) and couples them

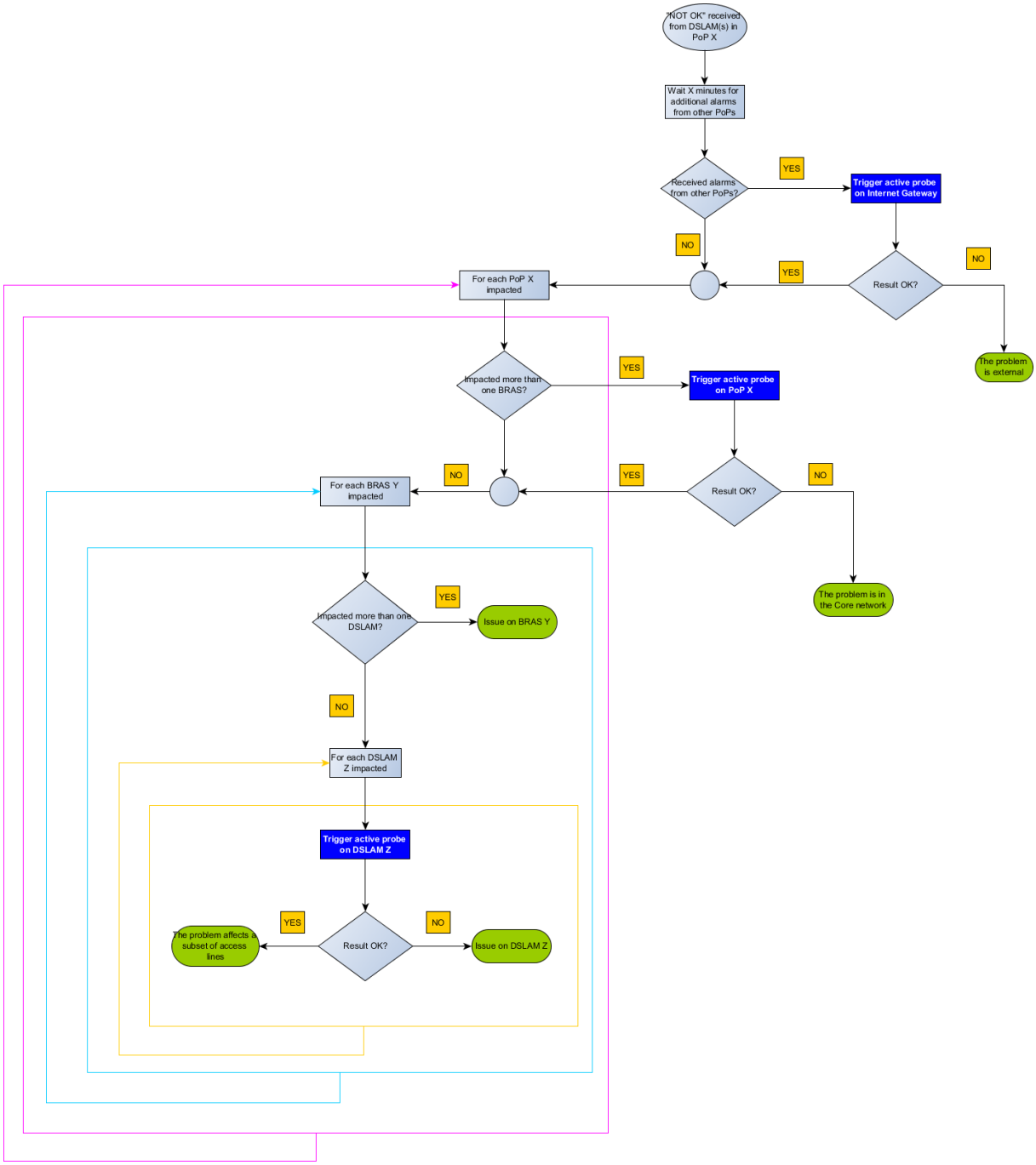


Figure 2.7: Diagnosis algorithm

with the log collected at the TCP level by Tstat. The probes perform then active measurements (namely, Ping and Traceroute) over the collected IP addresses. The result of the diagnosis is an information on the location of the root cause (e.g., local host / lan, home gateway, remote web server, and so on).

In D4.1 [73], we presented the preliminary algorithm and scenario aiming at identifying the segment that is responsible for the high Web page loading time. In D4.2 [73], we further detailed the

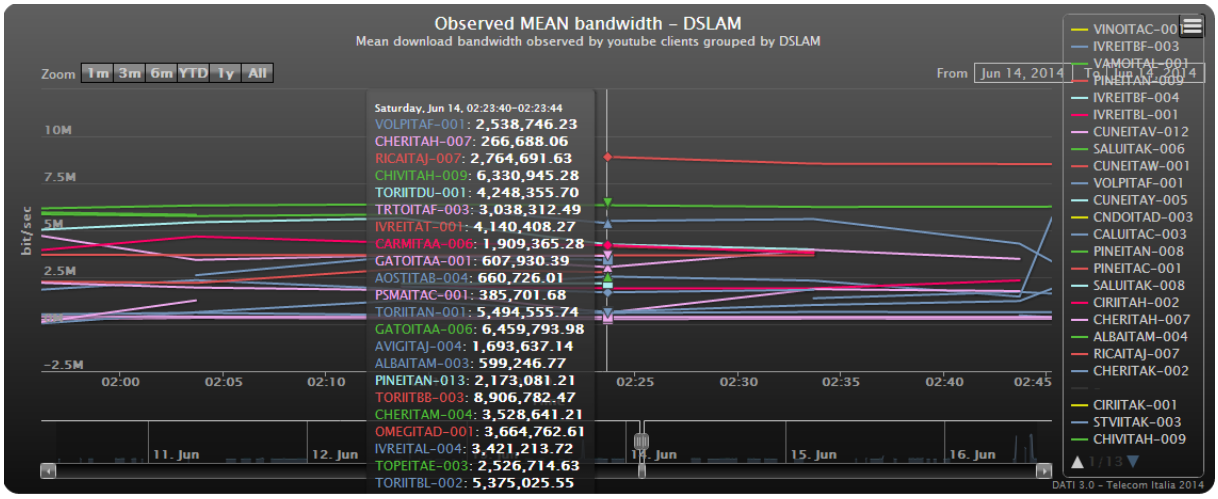


Figure 2.8: Average bandwidth aggregated per-DSLAM

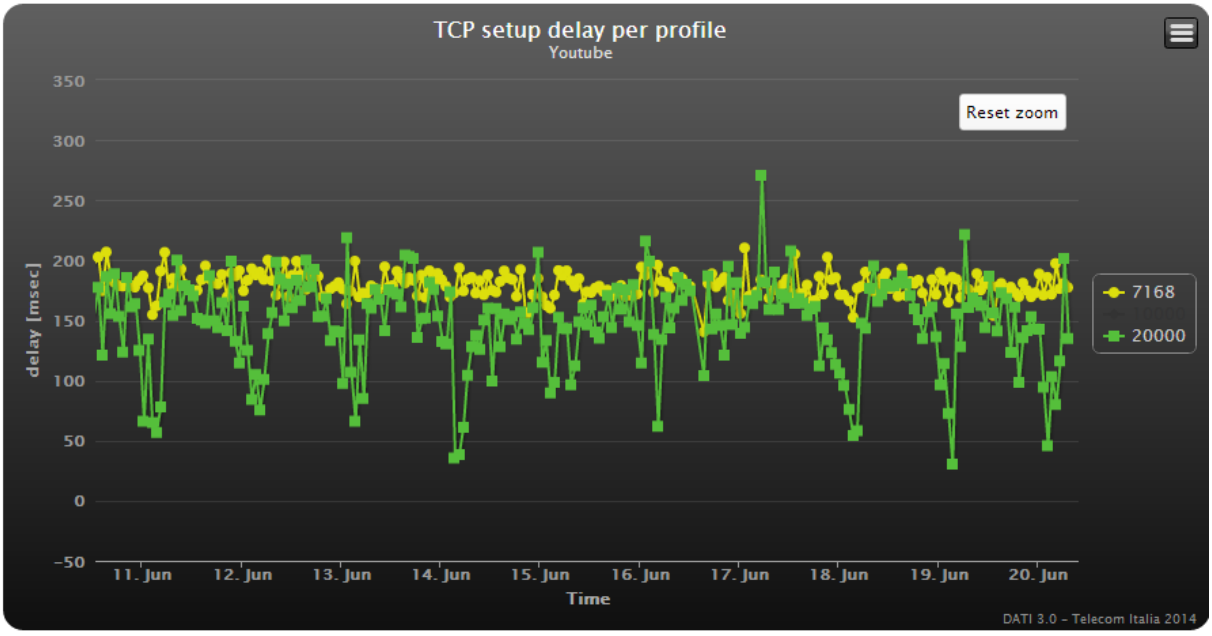


Figure 2.9: Average RTT aggregated per-DSLAM

Symbol	Metric	Passive or Active
T_{nhop}	RTT to the n^{th} hop	Active
Δ^n	$T_{(n+1)hop} - T_{nhop}$	Simple Computation
T_{idle}	Client idle time	Passive
T_{tot}	Total web page downloading time	Passive
T_{DNS}	DNS response time	Passive
T_{tcp}	TCP response time	Passive
T_{http}	HTTP response time	Passive

Table 2.4: Metrics used in the diagnosis algorithm for the Web QoE Use Case

workflow for the iterative analysis at the Reasoner level.

Now we present the complete picture, relying both on the probe measurements and on the repository data.

2.5.2 The Diagnosis Algorithm

We remind all the collected metrics that are exploited by the diagnosis algorithm in table 2.4.

We can identify different segments where the problem of a high page loading time could be located (each segment is indicated with a question mark in Fig. 2.10, taken from Deliverable D4.2 [73]): (1) at the probe side, i.e., local probe, local network, gateway; (2) at the domain side, i.e., middle boxes (if any), DNS server; (3) the backbone network; and (4) the remote web server.

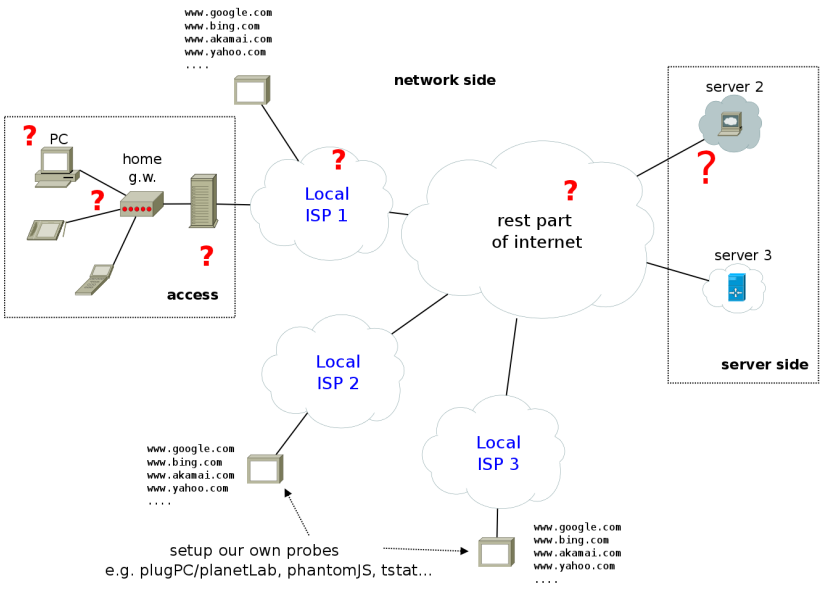


Figure 2.10: Network Scenario

The main diagnosis algorithm is described in Algorithm 1.

Lines 1-9 are executed on data coming from the single probe to infer the status of the probe (e.g., CPU usage) and browser level measurements[2] (lines 3-9).

If the total time of retrieving all the objects in the web page is under a certain threshold (test on line 3) then we investigate the remote web server part of the path, namely the page dimension, the time for setting up a TCP connection and the time taken to resolve the URL. On the contrary, if the total time of fetching all the items on a web page is high, there can be three distinct cases: all the other devices in the same local network are experiencing some problems, none of the other devices is experiencing any problem, and just some of the other devices are experiencing some problems (lines 11-34).

In case all the different devices are experiencing some problems (lines 12-18) the algorithm can directly exclude that the problem is due to the remote server (assuming that not all the devices are contacting the same remote server). The algorithm assumes then that most probably the problem is located close to the devices (otherwise probably not all the devices would experience problems) and thus begins this diagnosis phase by checking the gateway and the local network (Algorithm 2), if the problem is neither in the gateway nor in the local network, it checks the middle boxes (Algorithm 3), and finally, if the problem is neither in the middle boxes, it concludes that the problem is in the backbone network (probably in the portion of the backbone that is close to the local network, given that all the local devices are traversing it).

Let us dive into the check-gateway-lan algorithm (Algorithm 2). First of all (lines 1-5), the algorithm verifies if the CUSUM [97] (see Sec.2.5.3) applied to the T_{1hop} (where $1hop$ is the gateway) exceeds an appropriate threshold. If this is the case, this can justified by either the fact that the local network is congested or by the fact that the gateway is overloaded and the PING response time is "anomalous". To discriminate between this two cases the probe also checks the CUSUM applied to T_p (where p is another device of the network) and if it is "anomalous" too, it concludes that the problem resides in the local network, which is probably congested, otherwise it concludes that the problem is in the gateway, which is probably overloaded.

Else, if the T_{1hop} is "normal", the algorithm cannot yet exclude the overloaded gateway case (because the relation between the ping response time and the machine load is not always significant, as usually PING messages are directly managed by the network card) and performs a check on the CUSUM applied to Δ^1. Note that this metric, from a practical point of view roughly represents the sum of the time needed to traverse the gateway, the time needed to go through the first link outside the gateway, and the time required by the second hop to process the PING request. If this is "anomalous", the algorithm also checks the CUSUM applied to Δ^2 and in case it is "anomalous" too it concludes that there is congestion on the first link outside the gateway, which is reported as backbone network problem (note that if there are middle boxes the algorithm instead proceeds to the next phase), otherwise it concludes that the problem is in the gateway that is overloaded.

In case Δ^1 results to be "normal", the algorithm can exclude the overloaded gateway case, and proceeds by checking the middle boxes, if any. The verification of the middle box (Algorithm 3) is based on a process that is very similar to the one used to check the gateway. The algorithm checks the CUSUM applied to T_{nhop} (where n is the middle box). If this is "anomalous", it can conclude that the

[2] The current implementation of the probe with the headless browser does not allow the capturing of DNS times and idle times. The first, because we can not distinguish DNS requests from the Tstat logs, the second because on a headless browser there is no rendering of the web page. This choice was driven by the need to lighten the probe on low-power hardware, and to deploy the probe as a standalone box on which take the measurements. User interaction (i.e., the explicit warning that a high page loading time is experienced) is always collected by a "full browser" plugin.

Algorithm 1 Web QoE diagnosis main algorithm

1: **if** $\frac{T_{idle}}{T_{tot}} > \psi_1$ or $CPU\ usage > \psi_2$ **then**
2: return local client ▷ otherwise exclude local client
3: **if** $T_{http} < \psi_3$ **then** ▷ exclude any kind of network problem
4: **if** $PageDimension > \psi_4$ **then**
5: return page too big
6: **if** $T_{tcp} > \psi_5$ **then**
7: web server too far
8: **if** $T_{DNS} > \psi_6$ **then**
9: return DNS ▷ otherwise exclude the DNS problems
10: **else** ▷ the problem is somewhere in the network
11: check other probes in the local network
12: **switch** Do they have problems? **do**
13: **case** They are ALL experiencing problems ▷ exclude the remote server
14: **if** CHECK GW/LAN **then**
15: return GW/LAN
16: **if** CHECK MiddleBox **then**
17: return Middle Box ▷ exclude gateway and local network
18: return network ▷ the problem can be only in the net (near portion)
19: **case** NONE is experiencing problems ▷ exclude GW, LAN, Middle boxes
20: **if** $CUSUM(T_{http} - T_{tcp}) > \psi_6$ **then** ▷ reload page and check again
21: return remote web server
22: **else**
23: return network ▷ the problem can be only in the net (far portion)
24: **case** SOME are experiencing problems ▷ exclude gateway, local network, middle box, and almost certainly the remote server
25: return network
26: **case** No Other Probe Available
27: **if** CHECK GW/LAN **then**
28: return GW/LAN
29: **if** CHECK MiddleBox **then**
30: return Middle Box ▷ exclude gateway and local network
31: **if** $CUSUM(T_{http} - T_{tcp}) > \psi_6$ **then** ▷ reload page and check again
32: return remote web server
33: **else**
34: return network ▷ the problem can be only in the net (far portion)

problem is in the middle box, otherwise it checks if any anomaly is present in Δ^n: if not, it excludes the middle box and, in case, goes to the next middle box, otherwise it also check Δ^{n+1}, concluding that the problem is in the middle box, if the latter is "normal", or in the congested network, if not.

Note that this phase is apparently simpler than the one responsible for checking the gateway, just because it can exploit all the information already obtained from the previous phases: if the algorithm cannot locate the problem neither in the gateway and local network, nor in the middle boxes, it concludes that the problem resides in the near portion of the backbone network.

Back at the main algorithm, let us analyze now the case in which none of the other devices of the local network is experiencing any problem (Algorithm 1, lines 19-23). In this case, we can easily exclude the gateway, the local network, and the middle boxes, restricting the possible results to either the remote web server or the backbone network. Hence, the first check is performed on the remote server (that is assumed to be more probable than the backbone network, given that the only device that is experiencing problems is the one navigating that remote server).

To perform such a check, the algorithm verifies if the CUSUM applied to the metric $T_{http} - T_{tcp}$ is

Algorithm 2 CHECK GW/LAN

```
1: if CUSUM(T_1hop) > ψ then
2:     if CUSUM(T_p) > ψ then
3:         return Local
4:     else
5:         return GW                          ▷ optionally I can ask another probe to verify
6: else
7:     if CUSUM(Δ^1) > ψ then
8:         if CUSUM(Δ^2) > ψ then
9:             return Network (backbone) congestion
10:        else
11:            return GW
                                              ▷ exclude GW and LAN
```

Algorithm 3 CHECK MiddleBox

```
1: to apply for each middle box
2: consider middle box n                      ▷ for the first middle box after the gw, n = 2)
3: if CUSUM(T_nhop) > ψ then
4:     return MiddleBox n                     ▷ note that CUSUM(Δ^{n-1}) < ψ
5: else
6:     if CUSUM(Δ^n) > ψ then
7:         if  thenCUSUM(Δ^{n+1}) > ψ
8:             return network congestion
9:         else
10:            return Middle box n
11:    else
12:        check next Middle Box
                                              ▷ exclude MiddleBoxes
```

"anomalous" or not[3]. If yes, the algorithm concludes that the problem is located in the remote web server, otherwise that it is located in the backbone network.

If some of the local network devices are experiencing some problems and some are not (lines 24-25), the algorithm directly concludes that the problem is in the backbone network.

Note that, despite the quite complicated description of the algorithm, the number and the type of the operations made by the probe make it suitable for being used on a background task, without significantly affecting system performance. Indeed all the checks are performed by either simply comparing some passive measurements to a threshold or computing the CUSUM statistics (CUSUM is well-known for being suitable for all kind of real-time applications) and comparing it with a threshold.

Finally (lines 26-34), if no other probe are available in the local network, then we fall back to the case in which all probes in the network are experiencing problems.

2.5.3 Cumulative Sum

To discover anomalies, we exploit a well knows technique named Cumulative Sum (CUSUM) [97], also known as cumulative sum control chart. It is a sequential analysis technique,q typically used for monitoring change detection. Let us suppose to have a time series, given by the samples x_n from a process, the goal of the algorithm is to detect with the smallest possible delay a change in

[3] Note that this metric roughly represents the time needed by the remote server to process the HTTP `GET` request, being T_{tcp} almost independent on the server load, when the server is not in the local network

the distribution of the data. The assumption of the method is that the distribution before and after the change ($f_{\theta_1}(x)$ and $f_{\theta_2}(x)$) are known. As its name implies, CUSUM involves the calculation of a cumulative sum, as follows:

$$\begin{aligned} S_0 &= x_0 \\ S_{n+1} &= \left(S_n + log\left(\frac{f_{\theta_2}(x)}{f_{\theta_1}(x)}\right)\right)^+ \end{aligned} \tag{2.1}$$

The rationale behind the CUSUM algorithm is that, before the change the quantity $log\left(\frac{f_{\theta_2}(x)}{f_{\theta_1}(x)}\right)$ is negative, whereas after the change it is positive: as a consequence, the test statistics S_n remains around 0 before the change, and it increases linearly with a positive slope after the change, until it reaches the threshold ξ when the alarm is raised.

Note that the assumption about the knowledge of the two distributions $f_{\theta_1}(x)$ and $f_{\theta_2}(x)$, implies that CUSUM is only able to decide between two simple hypotheses. But, in case of network problems we cannot suppose that the distribution after the change is known (usually neither the distribution before the change is known). This implies the need of using the non parametric version of the algorithm [97], which leads to a different definition of the cumulative sum S_n. In more detail in this work we have used the non parametric CUSUM (NP-CUSUM), in which the quantity S_n is defined as:

$$\begin{aligned} S_0 &= x_0 \\ S_{n+1} &= (S_n + x_n - (\mu_n + c \cdot \sigma_n))^+ \end{aligned} \tag{2.2}$$

where μ_n and σ_n are the mean value and the standard deviation until step n, while c is a tunable parameter of the algorithm.

As far as the estimations of μ and σ are concerned, we can use the Exponential Weighted Moving Average (EWMA) algorithm defined as:

$$\begin{aligned} \mu_n &= \alpha \cdot \mu_{n-1} + (1-\alpha) \cdot x_n \\ \sigma_n &= \alpha \cdot sigma_{n-1} + (1-\alpha) \cdot (x_n - \mu_n)^2 \end{aligned} \tag{2.3}$$

where α is a tunable parameter of the algorithm.

2.5.4 Exploiting Analysis Modules at the Repository Level

The analysis modules computing statistical distributions and values at the repository level are provided as tools to check this (see Sec. 3.5). At the repository level, the Reasoner can check:

- historical values of the RTTs to a particular IP address,
- changes in the traceroutes to the same host,
- other probes in the same network (i.e., same 1st hop in traceroute: same gateway),
- other probes in the same area (i.e., geo-location tools),
- particular segments with high RTTs (e.g., `traceroute` from different probes intersecting two subsequent hops),

- the target web server against different probes geographically distributed,

In order to present consistent data to the Reasoner, (i) *cleaning*, (ii) *normalization*, and (iii) *transformation* are performed at the Repository level. The first is the process of detecting and correcting corrupt or inaccurate records, caused, for example, by user entry errors or corruption in transmission or storage; the second aims at reducing data to its canonical form, to minimize redundancy and dependency; and the third converts a set of data values from the data format of the source into the data format of the destination system (in our case we store JSON objects into HDFS).

After this preprocessing, the Repository presents the capability of computing simple statistic functions to be applied to the collected data (mean value, standard deviation, median, etc.) with different time granularities (e.g., hour, day, week, month), as well as the capability to cluster the collected data about users, flows, servers, ISP or geographical locations. As a result, we can dive in more details in all cases in which the main algorithm returns a generic "network" problem.

As for now, all the analysis modules on the Repository must be explicitly invoked, while our goal is to automate them as soon as new data arrive from the probes (i.e., moving from batch processing to stream processing).

2.6 Mobile Network Performance Issue Cause Analysis

2.6.1 Use Case Reminder

This scenario focuses in identifying the cause of possible problems that are related to experiencing video-on-demand on mobile devices.

Video stream delivery on mobile devices is prone to a multitude of faults originating either from device hardware constrains, failures in the wireless medium or network issues occurring in different points along the data path. Although well established video streaming QoE metrics such as the frequency of stalls during playback are a good indicator of the problems perceived by the user, they do not provide any insights about the nature of the problem nor where it has occurred. Quantifying the correlation between the aforementioned faults and the users' experience is a challenging task due the large number of variables and the numerous points of failure. To address this problem, we are developing a root-cause diagnosis framework for video QoE issues. With the aid of machine learning, our solution analyzes metrics from hardware and network probes deployed on multiple vantage points to determine the type and location of the problems.

2.6.2 System Model Overview

In a typical scenario where a user streams a video on a mobile device from a popular service like YouTube, a request to the content server is made to receive the video data. When the server receives the request, it sends the data either directly from the content server or through a Content Distribution Network (CDN). Then the data stream enters the Internet Backbone until it arrives to the client's ISP network. If the client is connected on a cellular network, then the data is delivered to the mobile from the client's serving cellular tower. If the device is connected from a Wi-Fi home network, then the video is delivered over a broadband access link to the home gateway and finally to the mobile device.

Each hop of the data path may suffer from impairments that can affect the smooth delivery of the video and therefore the user's experience. Congestion or bandwidth bottlenecks in the local or remote network segments, high load on the devices and problems in the wireless medium are some of the most significant issues that cumber the performance of video streaming services and contribute in the user's QoE degradation.

To detect the types of failures that may cause issues during the video playback, we need to place measurement probes at multiple vantage points (VP) so that we can extract performance metrics from different segments and devices along the path. In an ideal configuration, probes in all the intermediate devices of the path would provide us with measurements regarding the performance of each individual hop.

However, our approach only requires probes at the mobile device, the home router and the content server. We only use these three points as they allow us to capture issues at the boundaries of each of the three important entities in the video delivery path, the user, the ISP and the content provider.

With the mobile and the server probes we are able to collect measurements from both the data receiver and sender's point of view, which correspond to the endpoints of the connection. The home gateway acts as an intermediate VP capable of acquiring metrics from both the local (LAN) and the wide are network (WAN).

In this work, we focus on understanding the contribution of each VP when detecting problems, their type and location and with what accuracy. We also examine the benefits of combining the data from multiple VP when they cooperate and in which scenarios the combination becomes more beneficial.

2.6.3 Description of the Probes

Since the majority of popular video streaming services deliver content over TCP, statistics of TCP flows are key to analyze the network metrics that could reveal problems deeper in the data path. For the purpose of collecting network metrics, we use the TCP statistics tool `tstat`. Tstat is capable of periodically generating logs based on statistics extracted from the TCP flows that are observed during runtime, such as delay, re-transmissions, window size and time-outs.

The mobile probe. We developed the mobile probe for the Android platform to measure TCP, network interface and hardware metrics and monitor the system's log for events relevant to the progress of the video sessions and the performance of the playback. The probe is capable of collecting three types of metrics, network, hardware and system events.

In more detail, the network metrics that we obtain are the parameters and TCP flow statistics collected by tstat. From the logs created by the tool we extract 113 metrics and statistics about a single TCP flow, including RTT, number of packets, flow duration, window sizes, out-of-order packets and re-transmissions.

In addition to tstat, the probe is configured to log all incoming and outgoing packets at the NIC as well as the errors and dropped packets. The hardware metrics are required for providing information about the available resources on the device and the state of its connectivity. These metrics are directly correlated to the performance of video streaming and video decoding.

The hardware metrics on the smartphone capture the percentage of CPU utilization, the amount of free system memory in KB and the wireless connectivity status (RSSI). These three parameters give us important information about the hardware state and the amount of load of the device. Other hardware parameters were also considered but we concluded that the ones presented here are the

most significant for describing the device's performance.

Finally, the mobile probe monitors the system event log for information about the start-up delay of the video and the number and duration of rebuffering events. The rebuffering events indicate the depletion of the video buffers that force the playback to pause until the buffers fill up to a certain threshold. These interruptions experienced by the user, are indicators of problematic sessions and the QoE of the user. Except from stalls due to buffer outages we also consider stalls caused by high load on the device that do not allow the proper decoding of the video stream.

The router probe represents the first hop of the connection between the device and the server. It resides on the edge of both the LAN and WAN segments and therefore it is an important VP for measuring the performance of both sides. Although in our model we use a home router, in real scenarios it could be placed at any point that provides wireless connectivity such as a cellular base station.

The router probe configured to capture network metrics in a similar manner to the mobile probe. There is an instance of tstat running on the device and at the same time the probe logs statistics regarding the number of packets seen arriving or leaving the wireless interface and the number of packets that were dropped. In addition, the probe is capable of monitoring the RSSI of the wireless interface of the connected clients.

The server probe. The selection of the server for the placement of the probe allows the measurement of flows from multiple clients connecting from a large number of devices. It additionally serves as a measuring point with a view from the endpoint of the connection that helps identify issues like bottlenecks and slow response times.

The probe on the server is as well instrumented with tstat for measuring network parameters and with a tool to monitor and log statistics about the machine's NIC using the same approach as the router probe.

In this section we presented probes that were developed to be compatible with different platforms depending on the architecture and operating system of the host device. Hence, in our system we have versions of these probes for Android, OpenWRT and Linux.

2.6.4 Detection System

It is difficult to measure the correlation between QoE metrics such as playback stalls and hardware or network performance metrics, due to their non-monotonic and some times counter-intuitive relation. Established methods for identifying network or hardware faults do not return information on whether nor how these problems affect the viewer's experience.

For that purpose, we use machine learning methods to learn the correlations between performance and QoE metrics and to create a predictive model for detecting and characterizing the root cause of playback problems. Before applying the machine learning tools, we employ two techniques, feature construction (FC) and feature selection (FS) that help improve the performance of the classifier.

Feature Construction involves the creation of new features by processing the already acquired data as a means of increasing the information related to a given problem. To accomplish this task, instead of using the raw number of bytes or packets, we generate more general statistics that reflect performance issues regardless the size of the payload or the duration of the flow.

Specifically, for the better classification of sessions suffering from congestion and shaping, we initially constructed the average bitrate per direction from the received and transmitted bytes of each

device's NIC divided by the duration of the session. Based on the the maximum bitrate observed for each interface we then calculate an estimation of the interface's utilization.

In order to create more general metrics that can be used to compare sessions regardless the size or duration of the video, we normalize all the parameters which are expressed in bytes or packets with the respective total number of bytes or packets that were exchanged during the session. This way we obtain ratios of the initial parameters based on the size of the payload. We apply the same approach for the duration of the video which is normalized with the duration of the video session.

Feature Selection: To increase the performance of the algorithm in terms of both accuracy and execution time, it is important to significantly reduce the size of the feature space. For that purpose we apply the Fast Correlation-Based Filter method, which we use in conjunction with an attribute set evaluator. The reduction of the total number of features used to train the algorithm minimizes the over-fitting problem that adds noise and reduces the overall accuracy.

After applying FS, the number of features is reduced from 354 to 22. Among the remaining 22 features, those with higher rankings were the 3 hardware metrics from the mobile device: the free memory, the CPU utilization and the RSSI. There are also many of the constructed features we presented above, such as the upstream and downstream utilization of the interfaces and the normalized values of re-transmitted, reordered and out-of-order packets as well as the number of packets with payload. More parameters that received high ranking during the FS process are the average and standard deviation of RTT, the minimum segment size observed, the arrival time of the first payload packet and the receiver and sender window sizes.

Machine Learning: For the data processing and analysis we use version 3.6.10 of Weka. Weka is a collection of machine learning algorithms and tools for processing, classification, regression and clustering. From the variety of algorithms offered by Weka we select Decision Trees to perform the classification of the instances. Our classifier of choice for the data analysis is J48 which is an implementation of the popular C4.5 algorithm.

Given that the datasets we obtain from our experiments are comprised of a large number of features that often have a non-linear relation between them, decision trees are well suited for our predictive model since the performance is not affected by such non-linear relations.

2.6.5 Controlled Experiments

In our system we use two approaches to collect measurements for our datasets. First, we collect a set of measurements in a controlled environment in the lab. The second approach involves real-world tests in a real environment. We perform the controlled experiments to set the ground truth and evaluate the performance of the system, and later we deploy it in a real scenario to validate it under realistic conditions.

In order to model realistic conditions in the controlled experiments, we use measurements collected in the wild from mobile devices which were instrumented to run YouTube videos and collect the related network parameters. From the obtained measurements, we extract the distributions for the delay, loss, throughput and download rate for video sessions both on WLAN and mobile broadband connections and we then apply these values when simulating different network conditions in the lab.

Here, we describe the methodology for the controlled experiments and the process of generating the corresponding dataset.

2.6.5.1 Setup

We use the controlled environment of the lab to simulate a range of problematic scenarios in different segments of the data path that potentially cause interruptions in the playback and QoE degradation. We will later discuss about the accuracy of the algorithm in real-world experiments.

We set-up a simple testbed with a video server, a router/AP (Access Point) and an Android phone. The mobile is connected to the Wireless LAN of the AP and the server is in turn connected via an Ethernet cable to the router. A wired client acting as a background traffic generator to introduce variability, is also connected to the router. We use `tc` and `netem` to simulate a DSL link by shaping the downstream of the link between the server and the router to 7.8Mbit/s and to 5.22Mbit/s to simulate a mobile broadband link. We obtain these values from analyzing the video sessions in the real-world experiments.

For the DSL link we apply 50ms delay and 1.4% loss both following a normal distribution between $\pm$ 20ms and $\pm$ 1% respectively. 100ms $\pm 30ms$ delay and 0.75% $\pm 0.5\%$ was added for the mobile link (Fig. 2.11).

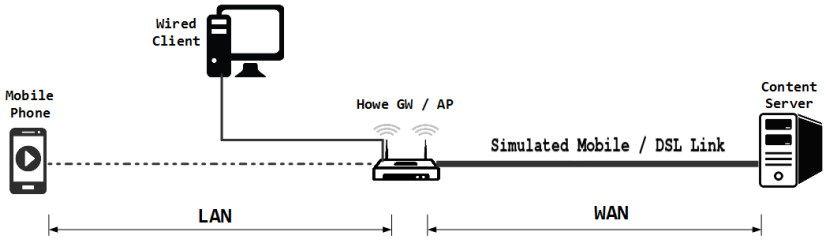


Figure 2.11: Device setup in the testbed

The video server operates on Linux with the Apache HTTP server installed. The videos from the top 100 most viewed list [110] have been previously downloaded from YouTube to the server in either Standard or High Definition to ensure the diversity of the collection. Furthermore, for some of the experiments real YouTube servers and existing streaming configurations have been used (see details below).

For the router/AP we used a Netgear WNDR3800 running OpenWRT. The access point of the device was configured to work on the 5GHz band in order to minimize interference from surrounding sources and we verified that no other devices were operating in the same frequency. For the mobile client, we used a Samsung Galaxy S II running Android 4.4.2.

The mobile application that we developed is responsible for performing HTTP requests to the server and open the returned video stream using the default Android media player. Similarly, the application can download videos from other services such as YouTube, etc. As soon as the playback finishes the application repeats the process by launching another random video from the list.

2.6.5.2 Background Variations

To generate the dataset for the controlled experiments, for each scenario we perform multiple iterations with random videos and gradually increment the intensity of the problem until we observe frequent re-buffering events. At the same time, attempting to create more realistic network conditions, we introduce variance to the system during every experiment by adding synthetic traffic

workloads of different patterns and at random intervals. For that purpose, we use the Distributed Traffic Generator [13] that supports traffic generation based on different applications such as Telnet, FTP, gaming, VoIP and more. We also use ApacheBench (ab) to create realistic HTTP traffic. Both tools were set to create traffic at random intervals and for random periods during the experiments.

2.6.5.3 Simulated Problems

Apart from constant background variations, we generate a set of specific problems to label our measurements. Specifically, we create scenarios for the controlled experiments that represent real-world problems and we compile a list of common faults that we will simulate to cause stalling during the video playback. These scenarios can be grouped in three basic categories, networking, device hardware and wireless medium issues.

The generation of the problematic conditions is performed in an automated fashion to easier control the experiments without the need of supervision. In more detail, when a new experiment starts, the background traffic sources and sinks are dispatched. At the same time, the experiment controller script sets up the environmental parameters that are specific to the problematic scenario that we want to simulate.

Next, the hardware and network monitoring probes are launched in the mobile, the router and the server. In the following step, a random video is selected and launched from the top 100 list. Based on the duration of the video a timeout is calculated so that we can detect when a video has finished playing plus some extra delay due to stalls and startup time. If the video has finished or timed out, we collect and aggregate the statistics from all the VPs for the given video session.

After the playback of 10 consecutive videos, the experiment controller script will modify the value of a parameter that contributes to the specific problem. This parameter and its value are defined according to the type of the problem we want to simulate and the intensity of the problem. Depending on the scenario we are simulating, the parameter could either stand for the CPU load, the traffic generated to create congestion, the amount of shaping of a link or the number of interference sources.

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

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

The traffic shaping scenarios are simulated with different bandwidth caps, delay and loss on the downstream of the related link. For the LAN shaping we select to limit the available bandwidth based on the data rates offered by common 802.11 standards such as a, b, g and n that are capable of providing rates per stream ranging from 1 up to 70Mbit/s. In these scenarios we apply delay with a normal distribution around 1ms and zero loss.

For the WAN traffic shaping we apply caps, delay and loss according to the distributions of the respective values observed in the measurements in the wild. For clients connected over mobile broadband we obtained throughput values normally distributed around 5.22Mbit/s, delay of 100ms and loss of 0.75%. For the WLAN connections the respective values observed were 7.8Mbit/s, 50ms

delay and 1.4% loss.

Mobile Load. In the second category we examine cases where the high load on the device hardware does not allow the proper decoding and playback of the video. This type of problems are more common on handheld devices that come with limited hardware capabilities. The hardware load simulation is performed with the Linux workload generator tool `stress` that allows CPU, I/O, memory and disk workload generation in order to stress-test the host system.

Low RSSI. The last category deals with the simulation of two faults common in 802.11 networks. In the first scenario (low RSSI) we simulate poor signal reception by placing the phone far from the AP and blocking the line-of-sight with physical objects. As a result, there is degradation in the wireless link's SNR and the available data rate. noticed high re-buffering frequency with RSSI values of -89dB and lower and link speeds less or equal to 2Mbps.

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

2.6.5.4 Dataset

All the collected metrics that correspond to a single video session are aggregated to one instance in the dataset. Each instance in the dataset is comprised of 354 metrics out of which there are 117 network metrics for each of the three VPs, the total number of rebuffering events and from the hardware measurements of the mobile we get the maximum CPU utilization, the minimum amount of available memory and the minimum value of the RSSI. It is important to note that the rebuffering events are only used for labeling the instances and not as a feature.

Overall, the dataset from the controlled experiments consists of 3919 instances in total out of which 3125 are labelled as good, 450 as mild and 344 as severe.

2.6.5.5 MOS-based Labelling

Before performing the analysis, the instances in the dataset need to be labelled appropriately in order to be identified and evaluated by the classifier. Our labelling system is required to express the quality of the video session in terms of user satisfaction so that we can correlate problematic videos with the QoE.

For that purpose, we convert application performance metrics such as startup delay and the frequency and duration of stalls to Mean Opinion Score (MOS) ratings based on the work of Mok et al. [72] who derived an equation for calculating the MOS from performance metrics by means of regression analysis.

Based on the obtained scores, we label instances with MOS greater than 3 as 'good', instances with scores between 2 and 3 as 'mild' and those with MOS lower than 2 are labelled as 'severe'. Good instances represent video sessions with fast startup times and close to zero stalls. The instances that were labelled as mild correspond to sessions with increased number and duration of stalls and initial delay, while instances marked as severe imply sessions with large amount of stalling and/or startup times usually longer than 5 seconds.

For the detection of the location of the problem, we create six new labels 'wan mild', 'wan severe', 'lan mild', 'lan severe' and 'mobile mild', 'mobile severe' based on the locality and severity of the problem. In label 'wan' we merge wan congestion and wan shaping problems, 'lan' contains instances from lan congestion, lan shaping, wifi interference and low rssi scenarios and finally in the 'mobile' we place the problematic instances that correspond to mobile load. For the evaluation of the algorithm when detecting the exact problem, we label problematic instances according to the type.

2.6.6 Evaluation

In this section, we evaluate the system's performance for three tasks, detecting the existence of problems, detecting the problem's location and finally for identifying the exact problem. In Sec. 2.6.9 we only evaluate for the detection of problems since we lack the ground truth for the type of problem. In the following sections we will also examine the performance of the algorithm in the wild.

Apart from the overall accuracy of the algorithm we additionally present the accuracy results using the Precision and Recall metrics. Precision expresses the ratio of True Positives (TP) over TP and False Positives (FP) and Recall the ratio of TP divided by TP and False Negatives (FN).

2.6.6.1 Who Can Detect a Problem?

In this part, we examine which VP or combination of them is performing better when identifying problems. We prepare the data by merging all the labels from problematic instances to labels 'mild' and 'severe', while preserving all 'good' labels. We consecutively evaluate for every VP separately and finally with all the points combined. The overall accuracy for the mobile is 88.1%, for the router 86.4%, for the server 85.6% and for the combination of all 88.8%. Although the server VP is performing slightly worse than the other two when used separately, we observe improvements in the classification of all the labels when we take measurements from all probes combined.

We can already see from these results that the system performs extremely well even when using a single VP while there is up to 3.2% improvement when using them combined. Moreover, the mobile VP outperforms the router and the server and can help achieve a classification accuracy almost equal to the combination of the VPs.

Fig. 2.12 (a) and (b) present the performance of the algorithm per VP in terms of Precision (P) and Recall (R), where we see that all VPs alone or combined perform extremely well for detecting good instances. This calls for instrumentation closer to the mobile terminals where the majority of the problems occur.

2.6.6.2 How does Severity Affect Detection?

With slightly lower accuracy, we are able to detect instances labelled severe from the mobile and the combination of the VPs while there is loss due to false negatives for the case of the router and the server.

The accuracy of the algorithm for all VPs decreases when identifying instances with mild problems as a result of the increase of false positives and false negatives as shown by the Recall values for the server and the router. The increased number of false negatives is attributed to 'mild' instances classified as 'good' due to the subtle differences between network and hardware conditions in se-

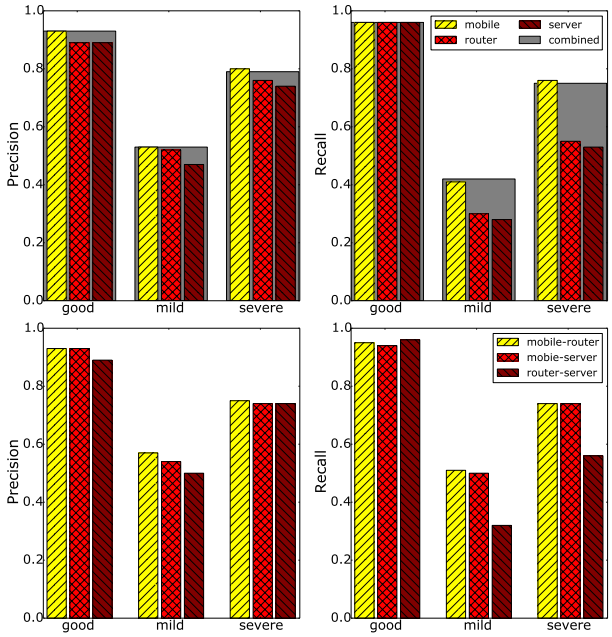


Figure 2.12: Precision and Recall for problem detection (a), (b) per VP and per VP pair (c), (d)

vere instances that the classifier is unable to detect. In a similar way, the increase of false positives occurs because of the often similar conditions with the good instances.

A possible approach to tackle this issue in the future, is to increase the number of labels for the problematic scenarios in order to provide a more fine grain division of the intermediate conditions between mild and good and mild and severe.

Our findings above show that either end users, ISPs or content providers can benefit from the system to detect problematic video sessions and the intensity of the given problem without having to exchange information with each other. In this way, providers can remotely evaluate their services and the client's QoE.

2.6.6.3 What are the Benefits of Vantage Point Pairs?

Next we examine the advantages when VPs cooperate in pairs. In Fig. 2.12 (c) and (d) we find that the combination of VPs in pairs yields improved accuracy when detecting mild problems as compared to the single VP approach. The greatest improvements can be observed for the mobile-server pair where the higher Recall numbers show the decrease of the number of mild cases which are falsely detected as severe and vice versa. There are also better results for both Precision and Recall for the mobile-router pair and smaller improvements for the respective values for the router and server combination.

The overall improvement is a good indication that the collaborative use of two VPs can help reduce the number of FN and therefore the amount of mild problems that are classified as severe. Two entities can collaborate to more effectively identify issues that affect the users' experience. Moreover, this approach can be applied to improve problem detection in cases where agreements to share information among more than two parties is not feasible.

2.6.7 Detecting the Location

2.6.7.1 Who Can Detect a Problem's Location?

In the next step we aim in verifying the algorithm's accuracy when identifying in which part of the data path the problem has occurred.

The percentage of the correctly classified instances increases to 89.21% in this evaluation case. As expected the accuracy for identifying good instances remains approximately the same as for good/mild/severe classification. In the related accuracies shown in Fig. 2.13 (a) and (b), we observe that the combined use of all three VPs is beneficial for the prediction of all the labels. As previously, the algorithm performs better for severe problem classification than for mild in all the cases.

Therefore, while a single VP can detect the existence of a problem, finding the exact problem might require some collaboration between two or more parties.

2.6.7.2 Does Proximity Improve Detection?

When identifying wan faults, the server outperforms the other two VPs, while the use of the mobile VP guarantees higher accuracy for the mobile problems. However, the detection of problems that occur in the lan is more accurate with the mobile instead of the router. This is attributed to the lack of RSSI measurements that is paramount for detecting faults that are attributed to low signal reception.

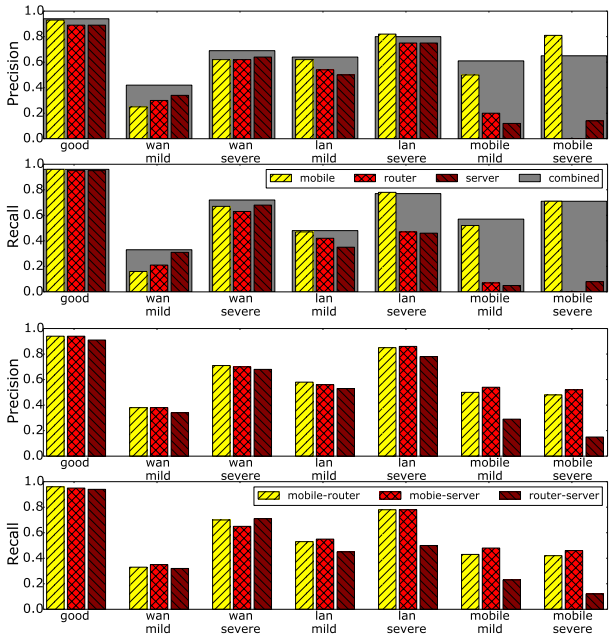


Figure 2.13: Precision and Recall for location detection per VP (a), (b) and per VP pair (c), (d)

From the values of Precision and Recall, we see that the algorithm is incorrectly identifying many problematic instances as good and vice versa. This is a result of the large number of false positives and false negatives for the problems that are located in the lan segment.

Similar observations can be made with regards to the mobile problems which are very difficult to detect from the router or the server due to the lack of information about the device's CPU and memory resources. However, in the case of mild mobile problems, the joint use of the three VPs can give small improvements in accuracy when compared to just using the mobile. These improvements mainly derive from the delayed acknowledgments that arrive from the mobile to the router and from the router to the server.

The results above show that each VP is performing better when identifying problems in the same location or segment of the path. This is a good indicator that users as well as providers can verify if the problem has occurred in their part of the network or not by performing measurements on their own networks and equipment.

The raw metrics which contributed more for detecting problems in the mobile are the CPU, the free memory, the RSSI, the TCP window size and the time the first packet with payload arrived to the router. For the lan detection, the parameters with the highest weight are the RSSI, the maximum RTT on the router, the time first acknowledgement was received and the TCP window size of the mobile and the router. Finally, for the wan label we observed that the most significant metrics were the RTT on the server, the minimum MSS of the server, the number of packets carrying video data, and the time the first packet with payload arrived to the server.

2.6.7.3 What are the Benefits of Vantage Point Pairs?

We again evaluate the benefits of using pairs of VPs instead of one at a time. The Precision and Recall values for the respective combinations can be found in Fig. 2.13 (c) and (d). The improvements are again significant for the mild cases but we can also observe accuracy increase for severe problems as well. Here we can consider the two VPs of the Mobile-Router and Router-Server pairs as the endpoints of the respective segments. From this point of view we notice the the Router-Server pair provides better results for wan problems, while the Mobile-Router increases performance for the problems in the lan.

From the obtained results we conclude that similar to the previous section, the usage of VP pairs contributes to a more accurate detection of the segment in which the problem has occurred. Moreover, the combination of the endpoints of a segment will result in better performance for the localization of the problems that take place in that segment.

2.6.8 Detecting the Exact Problem

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

From the output of the classifier we get 88.95% correctly identified instances when using all three VPs, while different labels are classified with different accuracies as seen in Table 2.5.

2.6.8.1 Which Problems are Easier to Detect?

In more detail, we observe higher performance for the wifi interference and low rssi related problems. This is attributed to the bigger weight of the hardware metrics that we extract from the mobile device that help the algorithm classify these instances with higher accuracy.

The main reasons of loss is miss-classifications between mild and severe cases of the same type of problem such as mild and severe congestion or mild and severe shaping. Moreover, mild problems are falsely identified as good or non-problematic instances are marked as problematic.

In both scenarios of loss described above, the classifier has trouble distinguishing the mild problems from good and severe because of marginal cases where network and hardware conditions have many similarities. For the marginal cases the probability of a miss-classification is further amplified from the different patterns of noise generated from the background traffic.

	Precision	Recall
good	0.94	0.97
wan congested mild	0.43	0.34
wan congested severe	0.63	0.6
lan congested mild	0.65	0.62
lan congested severe	0.84	0.67
wan shaped mild	0.42	0.33
wan shaped severe	0.59	0.58
lan shaped mild	0.66	0.6
lan shaped severe	0.76	0.69
mobile load mild	0.54	0.52
mobile load severe	0.73	0.67
low rssi mild	0.7	0.86
low rssi severe	0.87	0.93
wifi interference mild	0.71	0.43
wifi interference severe	0.79	0.88
Weighted Avg.	0.88	0.89

Table 2.5: Accuracies for root-cause detection in controlled experiments.

2.6.8.2 Who Can Detect the Exact Problem?

In terms of overall accuracy, the mobile is better than the router VP which in turn is better than the server, with respective accuracies 88.18%, 85.74% and 84.2%. These numbers demonstrate the system's high performance when carrying out the task of identifying the exact problem.

Fig. 2.14 (a) and (b) offer a more detailed view of the Precision and Recall metrics per VP and their combination. We can see that the VP on the mobile is able to detect with higher accuracy problems in the LAN segment and issues of the wireless medium. The router performs well when detecting lan congestion and lan shaping while the problems the server can identify with better accuracy than the other VPs is wan shaping and wan congestion.

With regards to faults of the wireless medium, both the mobile and the router VPs are capable

of accurately identifying wifi interference problems but low rssi cases are better detected by the router. These results are intuitive and expected since the server cannot obtain information about errors or performance issues that take place in the wireless channel.

Moreover, similar to the previous evaluation examples, the mobile problems are more accurately identified when using the mobile device as a VP. Again here the hardware metrics from the handset are playing a decisive role in the correct classification of these instances.

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

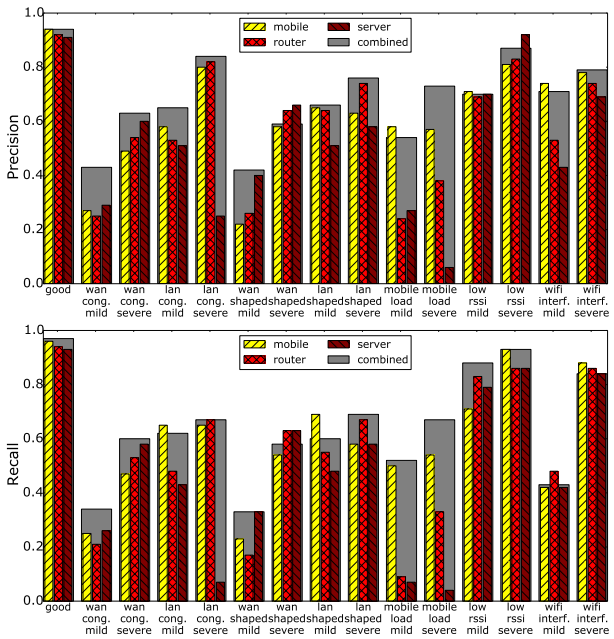


Figure 2.14: Precision (a) and Recall (b) for exact problem detection per VP

2.6.8.3 Do Feature Construction and Selection Help?

The objective of this section is to show the improvement we obtained with Feature Construction (FC) and Feature Selection (FS). We repeat the evaluation in Sec. 2.6.6 initially without FC and FS. Next, we apply FC and FS and compare the resulting accuracy. The analysis shows that when applying both methods we are able to get up to 2.27% improvement in accuracy.

It is evident that both FS and FC play a significant role in improving the system's accuracy and that these benefits can be amplified when the two methods are used together. The reduced feature space that results from FS minimizes overfitting problems and reduces the execution time. FC on the other hand, introduces more relevant information to the feature set and thus it results in better classification results.

2.6.9 Real World Deployment

The set of experiments in this section are performed outside the lab in real use cases and network conditions. More specifically, we distribute five Galaxy S II devices to equal number of users for a period of one month. The phones are again equipped with an application that automatically launches random videos from the top 100 list, while coordinating the network and hardware probes. The users were instructed to carry the phones with them in order capture measurements from a wide range of different networks.

In these experiments the videos are streamed from both our private video server and from YouTube with probabilities 0.25 and 0.75 respectively and the devices are configured to use both the Wi-Fi and mobile broadband connections to access the videos. We select these probabilities so that we end up with a dataset where the majority of measurements corresponds to YouTube sessions and a smaller part to streams from our server. The later set of measurements will be used to further train the algorithm and improve its accuracy.

We run a server probe to collect network statistics for the sessions that are streamed from our private server. With this approach we are able to collect measurements from two VPs (mobile, server) when the user is streaming the video from the private video server and one point (mobile) when streaming from YouTube.

2.6.9.1 Dataset

For the real-world experiments, although all hardware measurements as well as the number of re-buffering events are always available for each instance in the database, the number of network metrics varies depending on the number of VPs that were used. Therefore, in the final dataset we have instances with either 117 for one VP or 234 network parameters when using two. The real-world dataset contains 3495 instances from which 2940 are good 555 bad. problems and 144 with severe.

2.6.9.2 Results

The purpose of the analysis in this section is to validate the results we observed in the controlled experiment with measurements we obtained from experiments in real-world conditions.

Training the Classifier: To train the classifier we use the ground truth that we established in the controlled experiments evaluation. Therefore, instead of applying cross-validation, we use again J48 and train it with the dataset from the controlled experiments. With this approach the classifier is able to predict labels based on the labeled instances in the controlled dataset.

Detecting Problems: In contrast to the controlled experiments, in this dataset we only have measurements from the mobile and the server probes. Moreover, since the experiments are done in the wild, we are unaware of the root cause behind the stalls and the startup delay that were experienced by the user. Therefore we can only mark instances as good and bad based on these events using the MOS scale as previously. For that reason in this section we evaluate the dataset for good/bad classification alone.

2.6.9.3 Does it Work in Real Conditions?

We initially evaluate for the mobile VP only, consecutively for the server and finally for the two combined. The total accuracy is 82.89% for the mobile probe, 81.34% for the server and 83.70% for the combination. Fig. 2.15 shows the Precision and Recall for each case where we see that the combination of VPs improves the detection of videos without problems but performance is worse when finding problematic ones.

Similar to the controlled experiments, we find that the mobile probe is a better choice than the server for identifying both good and problematic instances while the combined use improves the system's accuracy. However, the main reason for loss here is the large number of false positives and false negatives when classifying bad instances. We attribute this phenomenon to the different patterns of noise that exist in real scenarios from the ones we generate in the lab. The different noise variations cause miss-classifications when dealing with mild problems that are more difficult for the algorithm to identify. Note that once such problems are troubleshooted by the ISP or the content provider they can be added to the training test, expanding the possibility of identifying them.

Overall, the results from the real experiments evaluation verify that the algorithm can detect problems in the wild (82.9% accuracy) even with fewer VPs available while the detection of non-problematic instances is performed with higher accuracy.

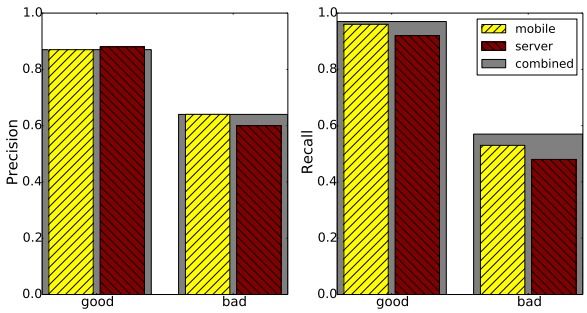


Figure 2.15: Precision and Recall for problem detection per VP pair in the real-world deployment

2.6.10 Practical Implications

Based on the findings in previous sections, providers should take advantage of installing probes in their part of the data path not only to identify problems but more importantly to tell if these problems have occurred inside their networks and there is need to take action. This is an important task as it supports SLA agreements, it providers awareness about the status of their network and it provides feedback from the end-devices, something that most providers are still missing.

We demonstrated that only a few network metrics such as delay, loss and throughput need to be extracted from residential gateways or content servers to provide details about the issue affecting the user.

When compared to the other two parties, the VP at the content provider's side had inferior detection capabilities. Although its performance is adequate for identifying problems at its side, the accuracy is reduced for mobile and lan fault detection. The mobile VP showed overall better performance in

the scenarios we evaluated. This advantage is attributed to the device's hardware metrics measured by the probe that contribute in detecting a wide range of local problems.

It is clear that the instrumentation of the mobile device is paramount for all the parties in the video delivery system. Moreover, content and service providers need to access measurements from the mobile device in addition to their own networks to infer the type of problems that affect QoE with high accuracy. Although these measurements do not contain sensitive information, some users might be reluctant to share these. Whenever providers are not able to get the users' consent, our results revealed that improvements can be achieved from the collaboration of content and service providers.

2.6.11 Limitations

One of the limitations of our system, is the inability to detect faults that it has not been trained for yet in the lab. These would not only include new problems such as middleboxes and DNS or routing miss-configurations but also the co-occurrence of problems that jointly affect video QoE.

Another limitation that in some cases affects the system's performance, is the miss-classification of instances with mild problems as either severe or non-problematic. This issue is more prevalent in marginal cases where the hardware or network conditions are very similar.

As discussed in the previous section, collaboration and exchange of information between parties is required in some cases in order to improve the system's performance. Such collaborations are often difficult to establish due to privacy concerns and fear of sensitive information leaks.

2.7 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 detection is based on statistical approaches, thus we stress the fact that the target is the detection of significant statistical changes in the analyzed measurements. In this sense, the detection of anomalies in a per-flow or per-customer basis is out of scope. Also, the analysis is done on top of passive measurements reflecting the activity of a large number of users, thus we consider passive measurements at vantage points located in Points of Presence (PoPs) of high traffic aggregation. These measurements are done at either the packet or the flow level.

The use case particularly focuses on the most popular web-based HTTP services (e.g., YouTube, Facebook, Google Services, Apple 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.

To detect and diagnose anomalies in web services, which additionally impact the end users, certain

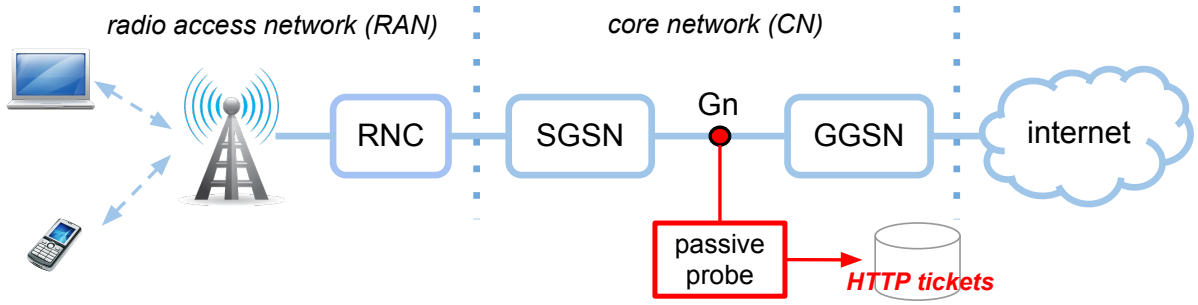


Figure 2.16: HTTPTag deployment in an operational 3G Network

pre-conditions must be satisfied: firstly, one must be capable of discriminating among the different web services in the stream of monitored packets/flows, and secondly, metrics reflecting the experience of the end users and methods to map network measurements to user experience have to be applied.

Next we present four different algorithms which allow to integrally tackle the detection and diagnosis of anomalies in web-based services, spanning (i) the automatic classification of traffic flows running on top of HTTP, (ii) the monitoring of HTTP services from a Quality of Experience (QoE) perspective, (iii) the detection of traffic anomalies based on statistical measurements analysis techniques, and (iv) the diagnosis of such anomalies.

2.7.1 On-line HTTP Traffic Classification through HTTPTag

HTTP is doubtlessly the dominating content delivery protocol in today's Internet. The popularity of services running on top of HTTP (e.g., video and audio streaming, social networking, on-line gaming, etc.) is such that they account for more than 75% of today's residential customers traffic. In this scenario, understanding HTTP traffic composition and usage patterns is highly valuable for network operators, with applications in multiple areas such as network planning and optimization (e.g., content caching), traffic engineering (e.g., traffic differentiation/prioritization), marketing analysis (e.g., heavy-hitter applications), just to name a few of them.

To classify web-based applications we have developed HTTPTag, a flexible on-line traffic classification system for analyzing applications running on top of HTTP. The approach adopted for the HTTP classification is based on *tagging*, i.e. associating a set of labels or *tags* to each observed HTTP request, based on the contents and service being requested. This association is performed by simple regular expressions matching, applied to the `host` field of the corresponding HTTP request's header. HTTPTag currently recognizes and tracks the evolution of more than 280 services and applications running on top of HTTP, including for example tags like `YouTube`, `Facebook`, `Twitter`, `Zynga`, `Gmail`, etc. Due to the highly concentrated traffic volume on a small number of heavy hitter applications, the current list of services spans more than 70% of the total HTTP traffic in the 3G network of a leading European provider.

HTTPTag works with packet data, passively captured at the vantage point of analysis. Fig. 2.16 shows current deployment of HTTPTag in the network of a major European mobile operator. Packets are captured on the Gn interface links between the GGSN and SGSN nodes. HTTP packets are detected and analyzed on the fly: every new HTTP transaction is parsed and the contacted `hostname` (extracted from the URL) is compared against the defined regular expressions or *patterns*, see

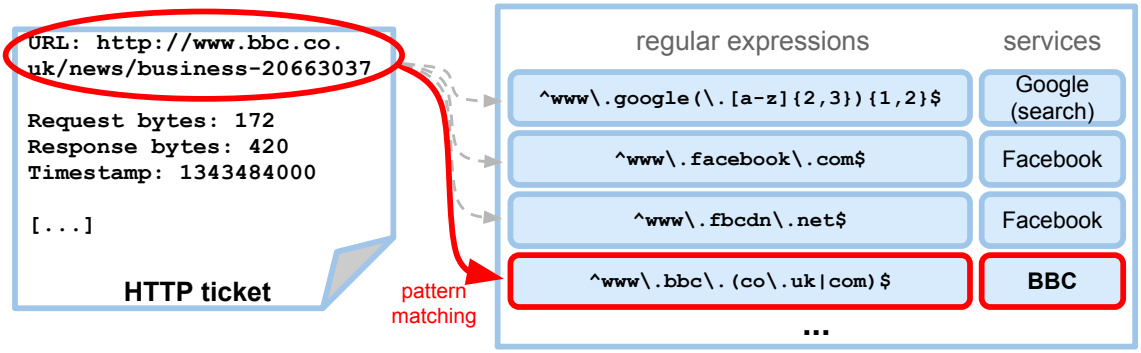


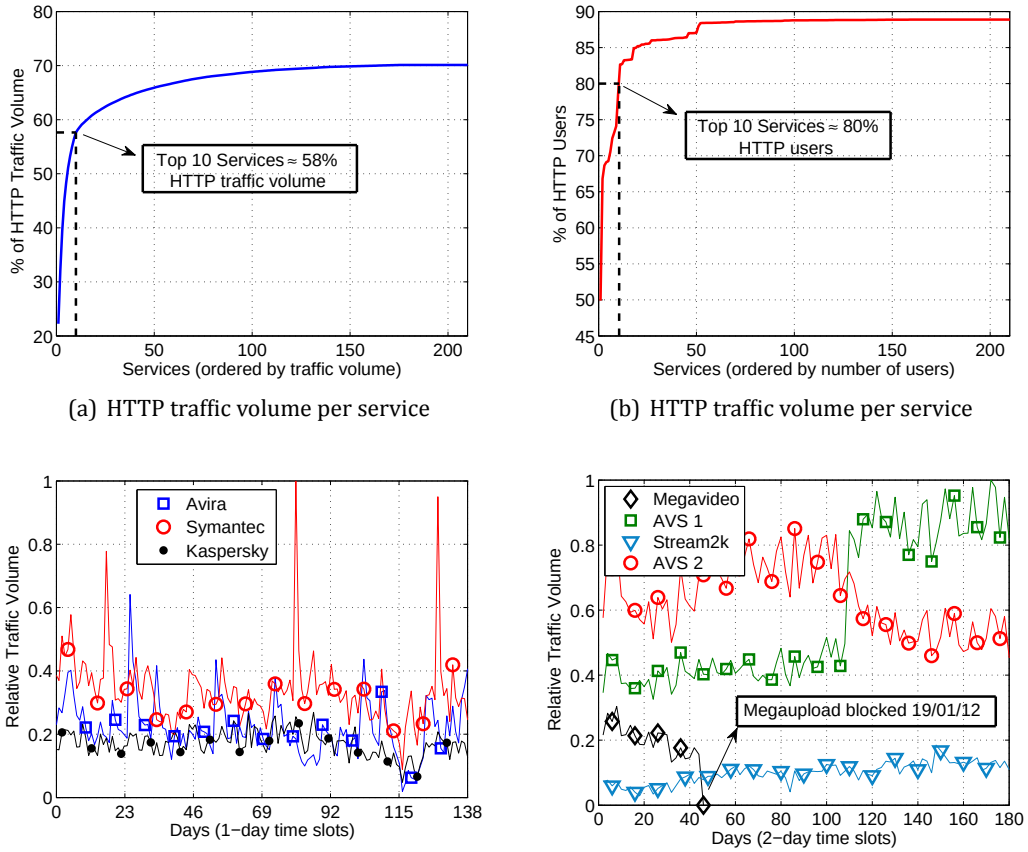
Figure 2.17: Matching URLs and Hostnames with patterns and services

Fig. 2.17. If a matching pattern is found, the transaction is assigned to the corresponding service.

HTTPTag runs on top of DBStream, a fast and scalable parallel database system tailored for large-scale network monitoring. For every new HTTP transaction analyzed by HTTPTag, a summary ticket is stored and indexed in DBStream, providing long term traffic analysis capabilities to the system. Each ticket contains a timestamp, the IP address of the contacted server, the requested URL, and volume stats (i.e., transferred bytes up/down). To improve pattern matching, patterns are ordered by probability of occurrence, which are computed from the history of successful matches. Many other different optimizations are performed at each of the steps of HTTPTag, including fast data imports within DBStream, efficient indexing for fast data access, and flexibility in the data query to allow multiple types of analysis on the tagged services.

HTTPTag tagging approach is based on manual definition of tags and regular expressions, which might a priori impose scalability issues. Indeed, there are millions of websites on the Internet and it would be impossible to define enough patterns to classify every requested URL. However, the well known mice and elephants phenomenon also applies to HTTP-based services, and limiting the study to the most popular services already captures the majority of the traffic volume/users in the network. While the initial definition of tags is a time-consuming task, regular expressions identifying applications tend to remain stable in time, basically because they are associated to the name of the application itself and thus recognized by the end-user. This is specially true for popular services, which at the same time carry the most of the traffic. In the practice, an initial effort in classifying the top 50 sites combined with weekly updates ensures a high classification rate. HTTPTag provides a GUI-based exploring system to identify the top host names responsible for the largest non-classified traffic volume and number of visitors, easing the tagging of new services. HTTPTag does not recognize HTTPS traffic, since the requested URLs are encrypted. To actually identify services running on top of HTTPS, HTTPTag applies the same tagging procedure on top of the DNS queries. In a nutshell, every time a user issues a DNS request for certain Full Qualified Domain Name (FQDN), HTTPTag assigns a tag to the corresponding FQDN, and creates an entry mapping this user to the server IPs provided in the DNS reply. Each entry is time stamped and expires after a time-out based on the TTL of the DNS reply. Using these mappings, all the subsequent flows between this user and the identified servers are assumed to belong to the service assigned by HHTPTag to the requested FQDN.

To show some examples of how HTTPTag performs, Fig. 2.18(a) and 2.18(b) depict the distribution of HTTP traffic volume and number of users covered by HTTPTag in a standard day. Using about 380 regular expressions and 280 tags (i.e. services) manually defined, HTTPTag can classify more than 70% of the overall HTTP traffic volume caused by more than 88% of the web users in an operational



(a) HTTP traffic volume per service

(b) HTTP traffic volume per service

(c) 5-months anti-virus services tracking

(d) 6-months video services tracking

Figure 2.18: HTTPTag classification coverage and some long-term tracking examples revealing different events of interest in an operational 3G network

3G network. As previously mentioned, a small number of heavy hitter services dominates the HTTP landscape: the top-10 services w.r.t. volume account for almost 60% of the overall HTTP traffic, and the top-10 services w.r.t. popularity are accessed by about 80% of the users. These results reinforce the hypotheses behind HTTPTag: focusing on a small portion of the services already gives a large traffic visibility to the network operator.

Fig. 2.18(c) and 2.18(d) depict two long-term tracking applications of HTTPTag. In 2.18(c) we track the traffic generated by three popular antivirus services (Symantec, Kaspersky, and Avira) over a four months period (from the 26/05/12 to 15/10/12). Analyzing the traffic patterns on a sufficiently long period gives a good image on the different approaches the three companies use to manage software and virus-definition updates. While `Kaspersky` shows a quite constant behavior, both `Symantec` and `Avira` present important peak volumes on specific update periods, which might heavily load the network. This information could be directly used by the network operator to define routing, load balancing, or prioritization/shaping policies. Fig. 2.18(d) depicts a comparison of four video streaming services on a 6-months period (from the 1/12/11 to the 25/05/12): Megavideo, Stream2k, and two adult video services (AVS 1 and 2). After 46 days from the starting tracking day, `Megavideo` traffic completely disappears, which correlates to the well-known shutdown of the Megaupload services on the 19/01/12. Part of the video streaming volume provided

by `Megavideo` was taken by a direct competitor, `Stream2k`, which shows a slow yet constant growth on the following months. Finally, we observe a drastic shift in the consumed volume from the two AVS services after 3 months and a half of steady traffic. We do not have a direct answer for this shift, but a change in the charging policy to access the content (e.g., free to subscription-based access) could explain such a variation. Having a complete picture of these popularity/usage modifications gives the operator the chance to better react to them (e.g., by defining specific content caching policies to reduce the load on the core links).

In the case of detecting anomalies in web-based HTTP services, HTTPTag allows to focus the analysis on specific services and/or specific CDNs, which is paramount to isolate issues and drill down into them.

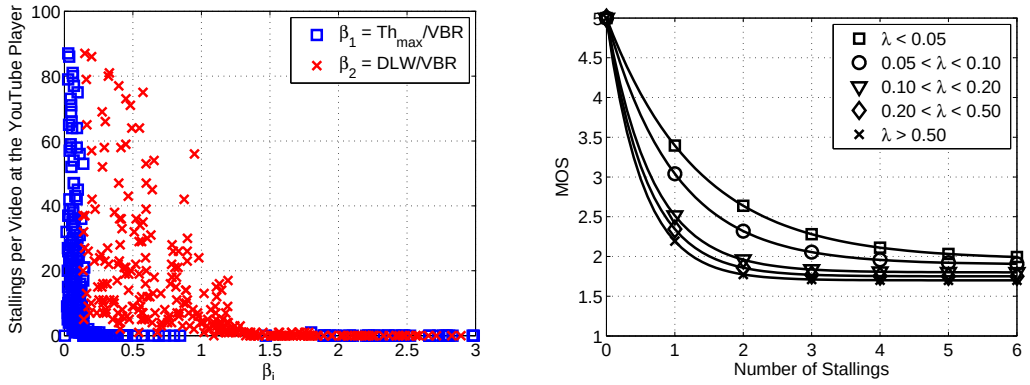
2.7.2 YouTube QoE-based Monitoring from Traffic Measurements

As we said before, with HTTPTag it is possible to identify the specific services one is interested in to analyze and in particular, to detect anomalies. YouTube is the most popular application in today's Internet, accounting for more than 30% of the overall traffic worldwide. For this reason, analyzing YouTube and detecting anomalies associated to this service is highly relevant, specially for network operators, who need to engineer their systems to handle the huge volume of traffic in efficient ways.

When it comes to the analysis of anomalies in highly popular services such as YouTube, the most important aspect to consider is the end-user of the service. While there are many different standardized approaches and Key Performance Indicators (KPIs) for QoE-based traffic analysis in voice and video over UDP-like services, standardization efforts regarding progressive download, HTTP-based video streaming are still ongoing, with some relevant yet under discussion results provided by ITU-T study group SG12 under questions Q13/12, Q14/12 and Q17/12. Another limitation of these approaches is that their practical application for real network monitoring is not always really discussed nor evaluated, offering basic models but lacking the how-to-get input parameters; check for example the recent P.1201 recommendation series [49].

For this reason, and despite not being part of the original DoW, we have devoted a lot of effort to the problem of QoE-based YouTube monitoring, developing, among others, two different QoE-based monitoring approaches for YouTube HTTP video streaming. The former approach is capable of estimating the quality experienced by users watching YouTube videos from network traffic, packet-level measurements only. We have called this YOUQMON. The latter is a simplified, light-weight version, and operates with flow-based measurements, making it easier to scale up to millions of customers (still, as we show next, we have been using YOUQMON with large-scale traffic at a 3G operational network, showing excellent results in terms of scalability).

YouTube QoE monitoring is about measuring the quality impairments perceived by the user at the application layer. From previous studies we know that the dominant QoE influence impairments of an HTTP video streaming service like YouTube are the number and the duration of the playback *stallings*, i.e., the events when the player stops the playback. Stallings occur due to the depletion of the player's buffer at the user's terminal. Some approaches to detect YouTube stallings rely on browser plugins or apps installed at the user's terminal. However, this approach is not highly extensible to large user communities and relies on the cooperation of the end-user. Hence, the big challenge and interest for a network operator willing to monitor the YouTube QoE of his customers is to detect such stalling patterns exclusively from the network side, being completely transparent to the end-users. This is exactly the approach followed by YOUQMON.



(a) Number of stallings vs β

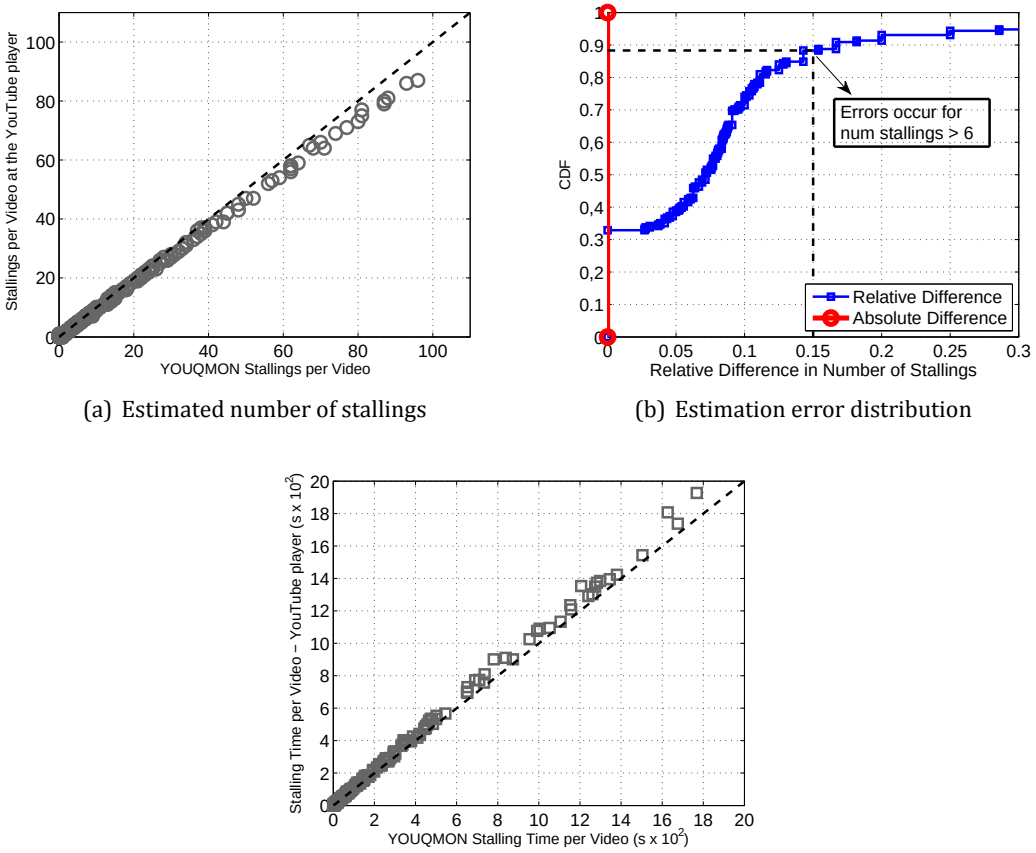
(b) MOS vs. n and λ

Figure 2.19: Stallings and QoE in YouTube.

YOUQMON works with packet data, passively captured at the vantage point of interest. In this specific description of the algorithm, packets are captured on the Gn interface links. Intuitively, when the downlink bandwidth (DLW) or the throughput achieved by the YouTube flows (Th_{max}) are lower than the corresponding video bitrate (VBR), the player buffer becomes gradually empty, ultimately leading to the stalling of the playback. However, measuring DLW and/or Th_{max} is not always enough to get a precise picture of the user experience. Fig. 2.19(a) shows the number of stallings measured at the YouTube player and it's correlation with the ratios $\beta_1 = Th_{max}/\text{VBR}$ and $\beta_2 = \text{DLW}/\text{VBR}$, for hundreds of different YouTube videos streamed to a local host with traffic shaping capabilities. While some marked relations can be observed for the average set of videos (e.g., no stallings if $\beta_2 > 1.3$), it is difficult to estimate the exact number of stallings for each specific video stream. For this reason, YOUQMON's QoE estimation principle is based on reconstructing the complete stalling pattern (i.e., number and duration of stallings) of a YouTube video from the analysis of the headers in the corresponding video packets.

YOUQMON is highly optimized to run on-line in standard 3G networks, as is its capable of decoding the complete 3GPP protocol stack in real-time. Nevertheless, the techniques apply to any IP-traffic vantage point. YOUQMON is fully developed in C for improved performance, and supports the two most popular video formats currently used by YouTube, namely Adobe-Flash and MPEG4.

The traffic analysis consists of two steps: (i) identify the beginning of every new YouTube video flow by HTTP header inspection, and (ii) extract the playtime offsets of the corresponding video frames to estimate the buffered video playtime at the YouTube player. The latter is achieved by inspecting the headers and tags of the video container (e.g., FLV or MP4). To preserve user privacy, any user related data are removed on-the-fly, and payload content is ignored. Using the extracted frame time offsets, YOUQMON tracks the amount of video playtime τ so far downloaded. The difference between τ and the current time of the video t corresponds to the remaining buffered video playtime $\Delta = \tau - t$. The video time t is computed as the difference between the actual time and the time t_p when the player starts to play the video; t_p is the time when the first packet containing actual video content is observed, plus an initial buffering time α_{play}. The video playback starts when the buffered playtime $\Delta > \alpha_{play}$, and stalls when $\Delta < \alpha_{sta}$. Both α_{play} and α_{sta} are estimated from large measurement campaigns. By tracking the evolution of Δ, YOUQMON can estimate on the fly when the YouTube video is playing or stalled. Every 60 seconds, YOUQMON computes a new ticket containing the number of stallings n and the fraction of stalling time λ for each detected video. The



(a) Estimated number of stallings

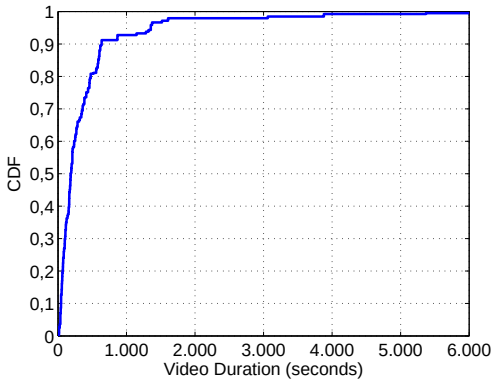
(b) Estimation error distribution

(c) Estimated stalling time

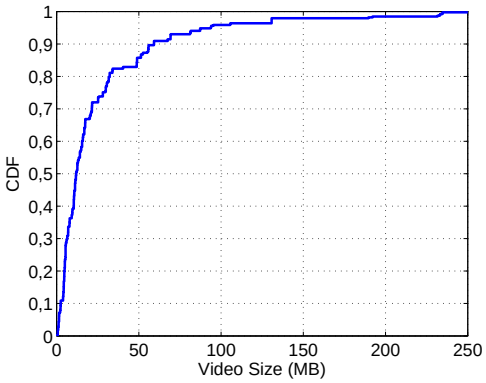
Figure 2.20: YOUQMON validation. Estimated (a) number of stallings, (b) distribution of errors, and (c) duration of stallings for 386 YouTube videos streamed from `youtube.com` through a bottleneck link

final step consists in mapping the resulting stalling pattern into a Mean Opinion Score (MOS) value. Using both lab and field subjective studies on YouTube QoE, we have conceived a model that maps n and λ into a user experience measure: $\text{MOS}(n)_i = a_i \cdot e^{-b_i \cdot n} + c_i, \quad \forall i = 1, \ldots, 5$. Each of the 5 MOS functions $\{a_i, b_i, c_i\}$ corresponds to a different fraction gap λ_i (e.g., $\lambda_1 < 5\%, 5\% < \lambda_2 < 10\%$, etc.). Fig. 2.19(b) depicts the corresponding QoE estimation model. All the aforementioned techniques are based on YouTube as video streaming service, but can be easily extended to other HTTP-based streaming applications, simply by re-computing the player's buffering thresholds.

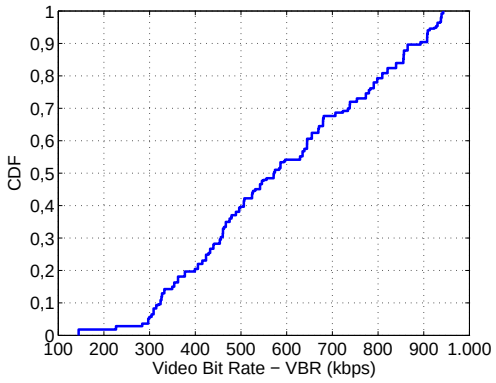
Fig. 2.20 reports some validation results of YOUQMON. Results correspond to 386 YouTube videos streamed from `youtube.com` through a bottleneck link of controlled capacity (from 128kbps to 20Mbps). Fig. 2.21 depicts the characteristics of the corresponding videos, in terms of 2.21(a) video duration, 2.21(b) video size, and 2.21(c) VBR. Fig. 2.20(a) and 2.20(c) show that the number and total duration of stallings per video computed by YOUQMON are highly consistent with the stallings measured at the YouTube player. Fig. 2.20(b) shows that for 131 videos, the number of stallings is zero and the absolute difference between the estimated (n_e) and the real (n_a) number of stallings is 0. For the 255 remaining videos, the relative difference $\frac{|n_e - n_a|}{n_a}$ is still 0 for 30% of the cases, and below 15% for about 90% of the videos. Hence, for more than 93% of the 386 tested videos, the estimation is either exact or there are errors for $n_a > 6$. According to the QoE model depicted in



(a) Video duration.



(b) Video size.



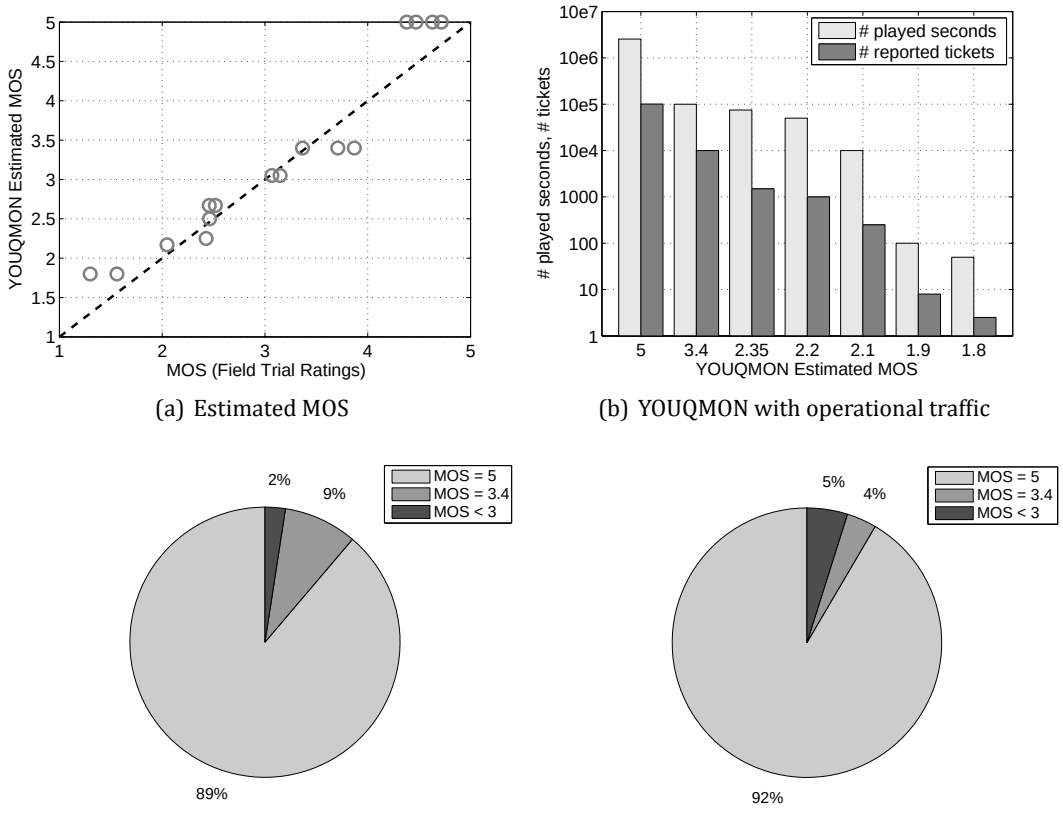
(c) Encoding video bitrate.

Figure 2.21: Characterization of the videos used in the validation of the stalling detector. More than 80% of the videos are shorter than 10 minutes, 90% are smaller than 60 MB, and the average VBR is 590 kbps

Fig. 2.19(b), MOS differences for $n > 4$ are negligible, showing that YOUQMON is actually performing highly accurately in the validation dataset.

To validate the QoE estimation properties of YOUQMON, we replay one day of the network packet traces captured in a field trial study, for which the MOS values declared by the users are provided as ground truth. Fig. 2.22(a) compares both the declared MOS and the YOUQMON MOS values for 16 different videos which experienced different stalling patterns in the field trial. Obtained results are very accurate and close to the MOS values actually declared by the participants, with only minor differences at the edges of the rating scale, which arise from rating saturation effects (e.g., in the field trial, ratings for 0 stallings correspond to MOS values around 4.5, while the model gives a MOS value of 5 in such cases) and are therefore not an issue.

Fig. 2.22(b) depicts an histogram on the number of reported tickets and the total played seconds of YouTube videos at the different estimated QoE levels, for one hour of real traffic monitored at the live 3G network previously mentioned. These results show that using YOUQMON it is actually possible to have a clear view of the performance of the mobile network as regards the satisfaction of the customers consuming YouTube videos. As reported by the charts in Fig. 2.22(c) and 2.22(d), the resulting YouTube QoE in this network is excellent (i.e., MOS = 5) for about 90% of the issued



(a) Estimated MOS

(b) YOUQMON with operational traffic

(c) MOS distribution (tickets)

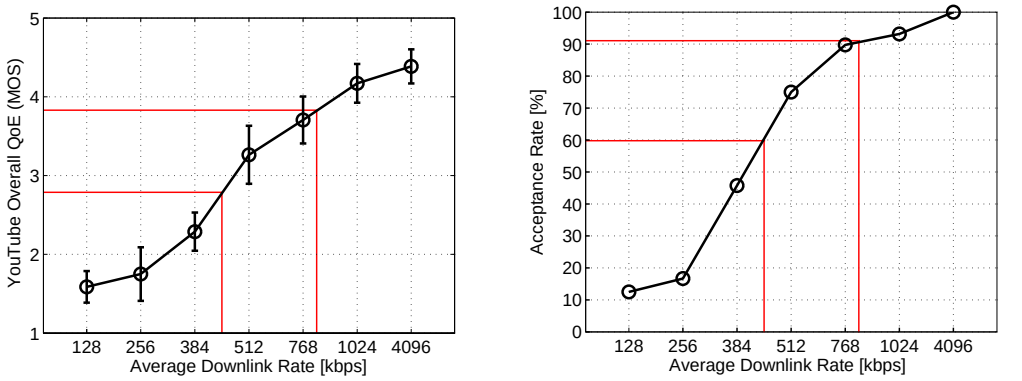
(d) MOS distribution (video time)

Figure 2.22: System validation in a mobile network

tickets and of the video time consumed during the analyzed hour. For 9% of the issued tickets and 4% of the total video time, the quality achieved was average (i.e., MOS = 3.4 in this case). Regarding bad quality events, YOUQMON can not say whether bad quality events come from problems on the network or in any other part of the end-to-end path (the customer terminal, the YouTube servers, a bad SNR, etc). Diagnosing such bad QoE events is part of the overall mPlane troubleshooting support tasks.

We come back now to the analysis of the β_2 indicator, which conforms the second approach we use for QoE-based YouTube monitoring. Even if it is not possible to estimate the exact number of stallings a video suffers from the computation of β_2, this indicator can still be used as a measure of how good is YouTube doing from a QoE perspective in terms of stallings/no-stallings. As depicted in Fig. 2.19(b), already 2 stalling events degrade the QoE to bad user experience (i.e., MOS < 3); as such, having a binary indication of the occurrence or not of stallings tells already a lot about the QoE at the end-user level. This is exactly the type of information one can extract from the computation of β_2.

Fig. 2.23 reports the overall QoE and the acceptance rate as declared by users watching YouTube videos during a field trial test, both as a function of the average downlink rate. During this one-month long field trial test, about 40 users regularly reported their experience on surfing their



(a) YouTube overall QoE versus downlink rate (b) YouTube acceptability versus downlink rate

Figure 2.23: YouTube overall QoE and acceptability in terms of average downlink rate. The curves correspond to a best-case scenario, in which only 360p videos are considered. In a more general case with higher resolution videos (e.g., 1080p HD), the download rate has an even stronger effect on the user experience.

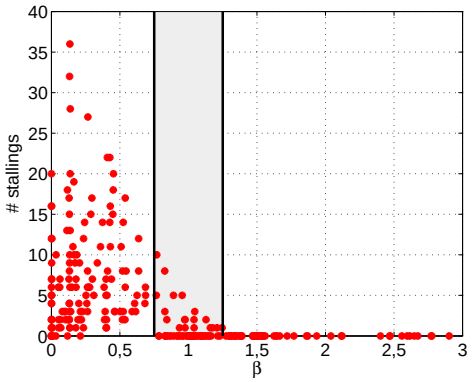
preferred YouTube videos under changing network conditions, artificially modified through traffic shaping at the core of the network. Both curves correspond to a best-case scenario, in which only 360p videos were watched by the users. Using the results of Fig. 2.23, we evaluate the possibility of using β_2 as a metric reflecting QoE degradations. Fig. 2.24 reports (a) the measured number of stallings events and (b) the QoE user feedbacks as a function of β_2. In particular, no stallings are observed for $\beta > 1.3$, and user experience is rather optimal (MOS > 4). As a direct application of these results, if we consider standard 360p YouTube videos, which have an average VBR = 600 kbps, a DLW = 750 kbps would result in a rather high user QoE, which is the value recommended by video providers in case of 360p videos. Fig. 2.24(c) additionally shows how the fraction λ = VPT/VD (ratio video played time over video duration) of the video time viewed by the end users increases when β_2 increases, specially above the β_2 = 1.3 threshold.

As a general conclusion of the two potential QoE-based monitoring approaches (i.e., YOUQMON or directly computing β_2), we can say that YOUQMON is more accurate in the exact estimation of the number of stallings of a video, which translates in a more precise QoE assessment. Still, β_2 represents an easy and very light weight means of knowing if throughput degradations are actually impacting the QoE of the end-users.

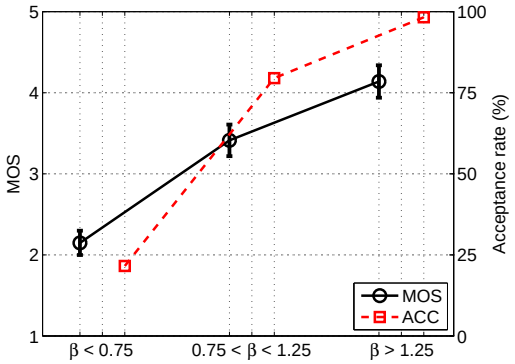
2.7.3 Statistical Anomaly Detection

Based on the previously introduced analysis modules, it is possible to identify the traffic of specific web applications running on top of HTTP/HTTPS, as well as assessing the QoE undergone by the customers using such services, and in particular YouTube. We now introduce an analysis module capable of detecting anomalies in the provisioning of such services.

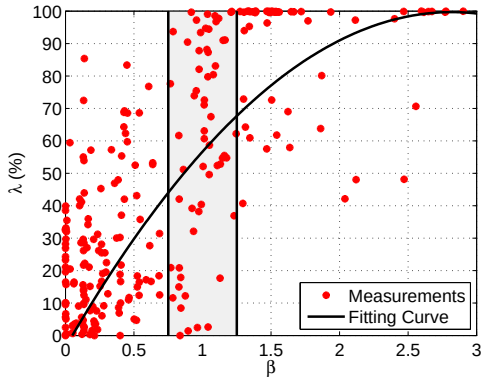
As we explained before, large-scale web services are usually provisioned through CDNs. The intrinsic distributed nature of CDNs allows to better cope with the ever-increasing users' content demand. Popular applications and contents are pushed as close as possible to the end-users to reduce latency and improve QoE. Load balancing policies are commonly used to limit servers load, handle



(a) # stallings vs. β_2



(b) MOS and acceptance vs. β_2



(c) User engagement (λ) vs. β_2

Figure 2.24: Users have a much better experience and watch videos for longer time when $\beta_2 > 1.3$, corresponding to a DLW = 750 kbps in 360p videos

internal outages, help during services migration, etc. Unfortunately, all these control policies are typically very dynamic and the details of their internal mechanisms are not publicly available. If on the one hand the highly distributed server deployment and adaptive behavior of large CDNs allows them to achieve high availability and performance, on the other hand they pose important challenges to the ISPs. The traffic served by CDNs can shift from one cache location to another in just minutes, causing large fluctuations on the traffic volume carried through the different ISP network paths. As a result, the traffic engineering policies of the ISP might be overruled by the CDN caching selection policies, potentially resulting in extra transport costs for the former. Finally, there might be cases in which the strategies in place result non optimal for end-users' QoE.

To detect such unexpected traffic shifts and more in general, in order to detect traffic anomalies, we present a novel network Anomaly Detection (AD) approach, specialized on CDN traffic. The goal of the AD algorithm is to detect macroscopic anomalies in the aggregate traffic served by CDNs, meaning events that involve multiple flows and/or affect multiple users at the same time. For this purpose, we resort to the temporal analysis of the entire probability distributions of certain traffic descriptors or features. In a nutshell, the proposed statistical non-parametric anomaly detection algorithm works by comparing the current probability distribution of a feature f to a set of reference distributions describing its "normal" behavior. The specific types of features we use capture both the intrinsic and dynamic CDNs mechanisms (e.g., number of flows and bytes served by each

CDN server IP address), and end-users experienced performance (e.g., flow download throughput, or the aforementioned β_2 QoE-based feature). Features are computed on a temporal basis, considering time bins of fixed length, referred to as time scale. We describe the algorithm next.

Given a certain traffic feature f (e.g., flow counts), we define $c_i^\tau(t)$ as a generic counter associated to f. The i-th counter can be associated to the client IP, to the server IP/network of a CDN, or finally to the i-th (quantized) throughput value. The symbol τ indicates the size of the time bin, and t is the time index. For example, $c_i^\tau(t)$ could be the number of flows served from IP i at time bin t of length τ minutes. The length of τ defines the timescale of the data aggregation, which in turn defines the timescale of the observable anomalous events. Given a certain time scale τ, the set of non-zero counters $\mathcal{C}^\tau(t) = \{c_i^\tau(t),\, i = 1, 2, \ldots, N^\tau(t)\}$ can be used to derive the empirical distribution of the feature f, denoted by $X^\tau(t)$, where the cardinality $N^\tau(t)$ could be for example the number of IPs serving traffic in the t-th time bin. As the following analysis can be done independently of the specific selected time scale, we omit the superscript τ from now on.

The anomaly detection algorithm consists in computing the degree of similarity between current distribution at time t, and a set of references distributions computed from past measurements at times $t_j < t$. To construct this reference set, we introduce the notion of *observation window* $\mathcal{W}(t)$, which is simply a sliding window containing past time bins: $\mathcal{W}(t) = \{t_j : a(t) \leq t_j \leq b(t)\}$, where $a(t)$ and $b(t)$ are the oldest and the most recent time bins that can be considered to evaluate the distribution $X(t)$ at current time t. The reference time bins set is denoted as $\mathcal{I}(t) \subseteq \mathcal{W}(t)$, and corresponds to the set of time bins selected from $\mathcal{W}(t)$ by running the *reference set identification algorithm* briefly described next. This algorithm identifies the set of past time bins with the most similar anomaly-free distributions to the current one. Given two distributions $X(t_i)$ and $X(t_j)$, of the same feature and timescale, at times t_i and t_j, we define $L(t_i, t_j)$ as a divergence metric accounting for the degree of similarity between the two of them. The choice of divergence metric is discussed next. The comparison between the current distribution $X(t)$ and the associated distributions reference set $\{X(t_j), t_j \in \mathcal{I}(t)\}$ involves the computation of two compound metrics based on the divergence $L(\cdot, \cdot)$. The first one, called *internal dispersion* and denoted by $\Phi_\alpha(t)$, is a synthetic indicator derived from the set of divergences computed between all the pairs of distributions in the reference set. Formally, $\{L(t_i, t_j), t_i, t_j \in \mathcal{I}(t), t_i \neq t_j\} \rightarrow \Phi_\alpha(t)$. We chose $\Phi_\alpha(t)$ to be the α-percentile of this set of divergence measures. The parameter α must be tuned to adjust the sensitivity of the detection algorithm: it defines the maximum distribution deviation that can be accounted to normal statistical fluctuations, therefore an acceptance region for the AD test. Similarly, we define the *external dispersion* $\Gamma(t)$ as a synthetic indicator extracted from the set of divergences between the current distribution $X(t)$ and those in the reference set. Formally, $\{L(t_i, t), \quad t_i \in \mathcal{I}(t)\} \rightarrow \Gamma(t)$. We chose $\Gamma(t)$ as the mean.

The detection scheme is based on the comparison between the internal and external metrics. If $\Gamma(t) \leq \Phi_\alpha(t)$ then the observation $X(t)$ is marked as normal. In this case, the boundaries of the observation window are updated by one time bin shift. Conversely, the condition $\Gamma(t) > \Phi_\alpha(t)$ triggers an alarm, and $X(t)$ is marked as abnormal. The corresponding time bin t is then included in the set of anomalous time bins $\mathcal{M}(t)$, and is excluded from all future reference sets. In this case only the upper bound of the observation window is shifted, i.e. $a(t+1) = a(t)$ and $b(t+1) = b(t)+1$. Such update rule is meant to prevent the reference set from shrinking in case of persistent anomalies. In fact, only the time bins in $\mathcal{W}(t) \setminus \mathcal{M}(t)$ are considered for the reference set.

A possible distance metric between two distributions is the *Kullback-Leibler* (KL) divergence. Let p and q be two discrete probability distributions defined over a common discrete probability space

Ω. The KL divergence is defined as:

$$D(p||q) = \mathrm{E}\left[\log\left(\frac{p(\omega)}{q(\omega)}\right)\right] = \sum_{\omega\in\Omega} p(\omega)\log\left(\frac{p(\omega)}{q(\omega)}\right) \tag{2.4}$$

where the expectation is taken on $p(\omega)$, and following continuity arguments, $0\log\frac{0}{q} = 0$ and $p\log\frac{p}{0} = \infty$. The KL divergence provides a non-negative measure of the statistical divergence between p and q. It is zero $\leftrightarrow p = q$, and for each $\omega \in \Omega$ it weights the discrepancies between p and q by $p(\omega)$. The KL divergence has several optimality proprieties that make it ideal for representing the difference between distributions. However, it can not be actually considered as a distance metric, since it is not symmetric and does not satisfy the triangular inequality. In particular, the lack of symmetry can be inconvenient in certain scenarios, particularly in the presence of events that take very low probability values in only one of the two tested distributions. Therefore, we adopted a more elaborated divergence metric, symmetric by construction:

$$L(p,q) = \frac{1}{2}\left(\frac{D(p||q)}{H_p} + \frac{D(q||p)}{H_q}\right) \tag{2.5}$$

where $D(\cdot||\cdot)$ is defined according to eq. (2.4), and H_p and H_q are the entropy of p and q respectively.

The feature distributions p, q in eq. (2.5) are unknown, hence they must be empirically obtained from the data samples. Some issues may arise in the estimation of the discrete probability distributions. Indeed, when the traffic distribution is computed for example from per IP counters, an obvious problem is the cardinality of the probability space Ω. A simple solution in this case is to consider per sub-network counters. Instead, when the considered traffic feature is the distribution of the throughput or the RTT across the users, then the empirical distributions found in real datasets are often heavy-tailed and span over ranges of a few orders of magnitude. In many cases, the sample size $N(t)$ is smaller than the range of spanned values. The standard approach in this case is to apply binning, i.e. to quantize the spanning range of the variable into a reduced number of bins, and to take the frequency of samples in each bin as the estimate of the distribution. The choice of the binning is critical because it affects the accuracy of the estimate, and ultimately the sensitivity of the detector. When this is the case, we adopt a non-uniform lin-log binning where the lower range is binned linearly and the upper one logarithmically, and the edges are automatically adapted so as to obtain a fixed number of bins. In some other cases we use our domain knowledge in defining meaningful bin edges. For example, in the case of the video download rate, we define bin edges which correspond to changes in the QoE, according to Fig. 2.23.

The design of the algorithm considers the identification of a set of distributions, which is used as the normality reference for the detection step. The identification of a suitable reference assumes a paramount relevance in the context of CDNs' traffic AD, due to the highly dynamic way CDNs host and serve the contents. Most of the AD work considers training once-and-for-ever and tests the current sample against the most recent ones. In the context of CDN AD, a reference based only on the most recent samples would not be able to take into account the steep variation in the total traffic counters in the morning and in the late evening, resulting in a series of false alarms. From the exploration of the real traffic traces we found that the traffic served by the analyzed CDNs (Akamai and Google CDN) share some common *structural characteristics* which must be considered for the choice of the observation window and reference set. For example, the traffic is non-stationary due to time-of-day variations, with steep variations occurring at certain specific hours like peak-utilization time, and with very strong 24-hours seasonality. We remark that such variations do not only apply to the flow counts and active server IPs, but also to the distribution of many other features such as volume, minimum RTT to the servers, download throughput, etc.

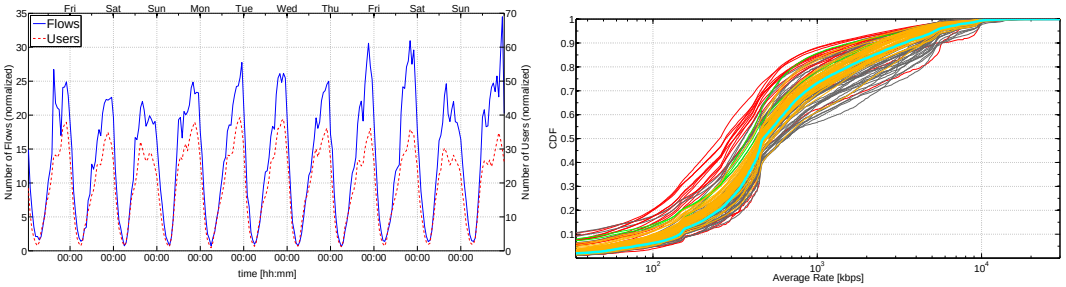
The heuristic used for the construction of the reference set follows a progressive refinement approach, where the mentioned structural characteristics are used at each step for reducing the set of candidate references in the observation window $\mathcal{W}(t)$. At each step, the set of candidate references is incrementally reduced by filtering the elements according to three different criteria. Given a new sample at time t of size $N(t)$, in the first step the algorithm picks the subset $\mathcal{I}_0(t)$ of past time bins with samples of similar size, formally $\mathcal{I}_0(t) = \{j|N(t) - s \leq N(j) < N(t) + s\}$. Such size-based criterion avoids comparing distributions with very different statistical significance, as the sample size can vary across two orders of magnitude during the 24 hours (see for example Fig. 2.25(a)). In a second refinement step, the subset of elements in $\mathcal{I}_0(t)$ with the smallest divergence from current observation are picked. In this way, samples related to different time of day and/or type of day (working day vs. weekends/festivities) are filtered out. The residual set $\mathcal{I}_1(t)$ might still contain residual heterogeneous samples. To eliminate these samples, in the third step we resort to an heuristic in which we apply a graph-based clustering procedure to identify the dominant subset with the lowest inter-samples divergence: samples are mapped to nodes, with edges weighted proportionally to the KL divergence among them. The algorithm divides the nodes in two clusters so as to minimize the intra-cluster weights, and finally the larger cluster is picked as the final reference set $\mathcal{I}(t)$.

The overall procedure is designed to minimize the inter-samples divergence within the reference set, so as to preserve good sensitivity of the detection process. We stress the fact that past observations (samples) which were previously marked as “anomalous” by the detector are excluded from the reference identification procedure; in other words, only samples marked as “normal” are taken as candidates. This introduces a feedback loop, as the output of the detector for past samples impacts the identification of the reference set, and therefore influences the future decisions.

Our experience shows that the proposed heuristic copes well with the time variability of both the distribution shape and the sample size. It does so by embedding the intrinsic pseudo-cyclical structure of the real traffic process into the reference set, resulting in a minimum set of past observations with the lowest divergence with respect to the current sample. In a nutshell, it leverages pseudo-seasonality to compensate for non-stationarity. As an example, Fig. 2.25 shows the typical output of the reference identification algorithm. In this specific example, we consider the distribution of the average download rate across the users watching YouTube videos during 11 consecutive days. Fig. 2.25(a) explains the ideas behind the first step of the reference set identification procedure, where distributions are selected based on the number of samples – flows in this case – used to derive them.

Fig. 2.25(b) depicts the output of the reference set identification algorithm. The cyan CDF represents the sample under test. The gray CDFs correspond to those samples in the observation window which are discarded by the identification procedure. The red CDFs are the samples in the observation window which are discarded for being previously marked as anomalous. Finally, the orange CDFs are those selected as reference. Note that out of all the possible candidate distributions, the algorithm selects the ones with lowest divergence to the current one, i.e., the orange CDFs. We remark that the proposed scheme is robust to irregularities in the pseudo-cycles – as introduced for example by non-weekends festivities, or solar/legal time shifts – since it does not rely on any external label information (e.g. calendar day or absolute time).

CDN cache selection policies may also have a strong impact on the service quality as experienced by the end users. This is not only a main issue for the end-users, but also for the ISP providing the Internet access to the contents, as customers will in most cases directly blame the ISP for the bad QoE, even if the origin of the problems is located outside its boundaries.



(a) Number of flows and users watching YouTube videos

(b) Output of the reference set identification algorithm

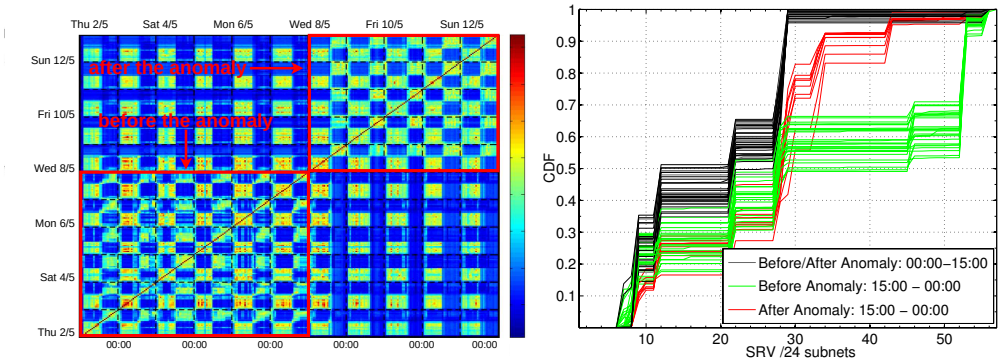
Figure 2.25: (a) Total number of flows related to download average rate, and number of users generating the traffic. (b) Output of the reference set identification algorithm

As an application example of the presented AD algorithm, we report now a real case in which an unexpected cache selection and load balancing policy employed by Google results in an important drop on the average download throughput for the end-users watching YouTube videos. Indeed, conversations with the ISP confirmed that the effect was indeed negatively perceived by the customers, which triggered a complete Root Cause Analysis (RCA) procedure to identify the origins of the problem. As the issue was caused by an unexpected caches selection done by Google, the ISP internal RCA did not identify any problems inside its boundaries. This standard procedure followed by operators should always be complemented with a verification of the status of the services being accessed by the users, which in many cases are the root of the problems.

The dataset corresponds to one month of HTTP video streaming flows collected at the fixed-line network of a major European ISP, from April the 15th till May the 14th, 2013. The monitored link aggregates about 30.000 residential customers accessing to the Internet either using ADSL or Fiber-To-The-Home (FTTH) technologies. Flows are captured using Tstat. Using Tstat filtering and classification modules, we only keep those flows carrying YouTube videos. These flows are finally imported and analyzed with DBStream.

As reported by the ISP operations team, the anomaly occurs on Wednesday the 8th of May. Fig. 2.26(a) shows the TSP of the video volume served by the different IPs in the dataset, aggregated in /24 subnetworks, and using a time-scale of 1 hour. Recall that in a TSP plot, each point $\{i, j\}$ represents the degree of similarity between the distributions at hours t_i and t_j. The blue palette represents low similarity values, while reddish colors correspond to high similarity values. The TSP is symmetric around the 45° diagonal, thus the plot can be read either by column or by row. For a generic value of the ordinate at t_j, the points on the left (right) of the diagonal represent the degree of similarity between the past (future) distributions w.r.t. the reference distribution at t_j. Note the regular "tile-wise" texture within a period of 24 hours, due to a clear daily periodicity behavior in the selected servers. Specifically, there are two subnet sets periodically re-used in the first and second half of the day. The TSP clearly reveals that a different subnet set is used during the second half of the day from the 8th of May on, revealing a different cache selection policy. This change is also visible in the CDFs of the per subnet volume depicted in Fig. 2.26(b). Indeed, we can see that the same set of subnets is used between 00:00 and 15:00 before and after the anomaly, whereas the set used between 15:00 and 00:00 changes after the 8th, when the anomaly occurs.

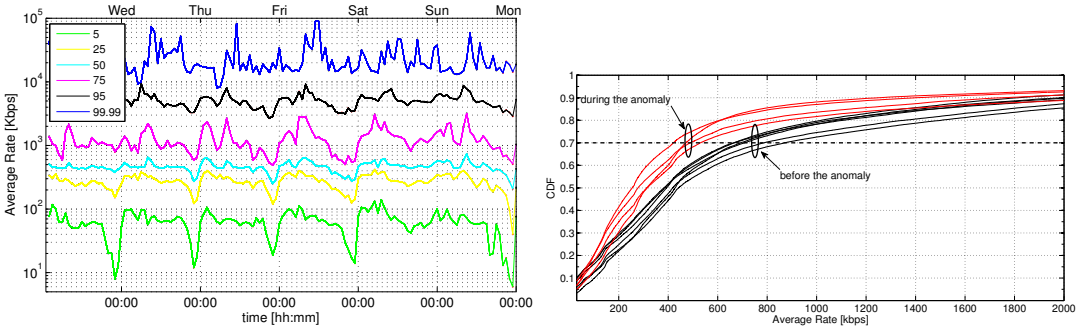
Despite this detected change in the cache selection policy employed by Google, such a modifica-



(a) TSP of volume per CDN /24 subnet

(b) Volume per CDN /24 subnet

Figure 2.26: Traffic volume distributions per CDN /24 subnets. There is a clear shift on the selected caches serving YouTube before and after the reported anomaly on Wednesday May, 8th, specifically in the afternoon, between 15:00 and 00:00

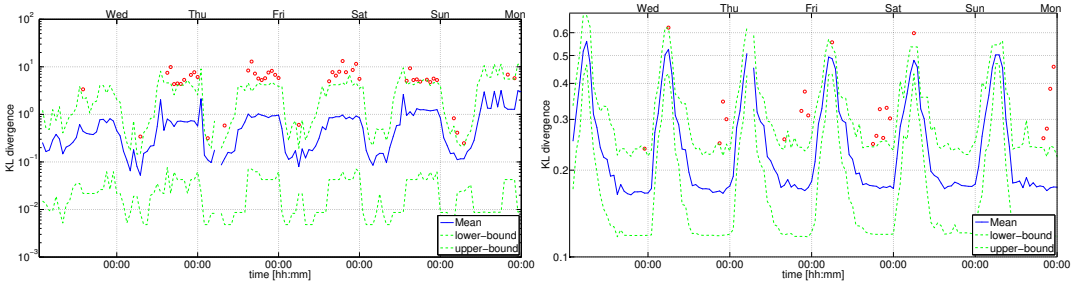


(a) Temporal evolution of several percentiles of the average video flows download rate

(b) CDFs before and during the anomaly

Figure 2.27: Distribution of the video flows average download rate across the users: (a) trend over time for several percentiles, (b) CDFs at peak hours (21:00-23:00), before and during the reported anomaly

tion does not justify by itself the QoE degradation reported by the ISP. To further investigate this issue, we analyze the distributions of the average video flows download rate. Figure 2.27(a) depicts the temporal trend of several percentiles of the average video flows download rate per user, starting one day before the anomaly occurs and covering five consecutive days after it. The lowest percentiles (i.e., 5% and 25%) show a constant drop on the average download flow rate during peak hours (between 21:00 and 23:00), even before the anomaly actually occurs. However, starting on Wednesday, even the 50% and 75% percentiles present an important drop at peak hours, explaining the flagged QoE degradations. Figure 2.27(b) analyzes the distribution of the average video flows download rate, in the hours before and during the anomaly. Interestingly, the only distributions exhibiting a marked change before and during the anomaly are those corresponding to the peak hours (21:00-23:00), which are those reported in figure 2.27(b). Indeed, if we focus for example on the 70% percentile, we observe a drastic reduction on the video flows download rate, going from about 780 kbps to 470 kbps. Even if this reduction might not look significant a priori, figure 2.23 shows that it is sufficient to drop the perceived quality below the level of acceptance.



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

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

Figure 2.28: Detection of anomalies in YouTube traffic. Alarms and acceptance region for the distribution of (a) volume and (b) video flows average download rate. The red markers correspond to the flagged anomalies

Indeed, Fig. 2.23(a) shows that the overall QoE drops from a MOS score close to 4 at 780 kbps to a MOS score below 3 at 470 kbps. A MOS score of 4 corresponds to good QoE, whereas a MOS score below 3 already represents poor quality. Fig. 2.23(b) additionally shows how the acceptance rate (i.e., the proportion of customers accepting to use the YouTube service at the corresponding downlink rate value) drops from about 90% in normal conditions to nearly 60% during the anomaly, providing more evidence on the impacts of such downlink rate drop on the users. To conclude the analysis, we report in Fig. 2.28 the output of the proposed AD system. Fig. 2.28(a) considers the per /24 subnet served volume as the monitored feature. It shows how $\Phi_\alpha(t)$ (with $\alpha = 95$-th percentile) adapts over time to follow the natural traffic daily changes. The red markers indicate when the condition $\Gamma(t) < \Phi_\alpha(t)$ is violated, triggering an anomaly. From Wednesday the 8th of May onward the algorithm systematically rises alarms from 15:00 to 00:00, which correspond to the discussed change in the caching policy. Fig. 2.28(b) reports the same information for the average video flows download rate. In this case, the AD system detects some anomalies only between peak hours (21:00-23:00) from the 8th onward, coherently with the observations drawn from Fig. 2.27. Interestingly, it can be noticed that even during peak hours, the anomalies are not detected on Saturday the 11th, whereas they are back on Sunday. This behavior is easily explained by the lower traffic served during the peak hours on Saturday, as shown in Fig. 2.25(a). Indeed, the percentiles depicted in Fig. 2.27(a) do not reveal a clear deviation on Saturday average download rates. Comparing the changes on the volume distribution against those on the video flows download rate distribution, we observe that the cache selection policy used by Google resulted in a QoE degradation only during the peak hours on the high load days. This suggests that the servers of the selected caches were not correctly dimensioned to handle traffic load peaks.

2.7.4 Entropy-based Diagnosis of Device-Specific Anomalies

We present now an approach which aids operators in rapidly diagnosing anomalies as the previously analyzed. This approach currently targets cellular network scenarios, because of their major popularity and the difficulty in diagnosing problems in these networks. Indeed, cellular network operators have witnessed an amazing increase of heterogeneous mobile devices (smartphones, tablets, M2M devices such as telemeters, etc.) during the last decade. The applications supported by these devices introduce new traffic patterns which are potentially harmful for the network. For

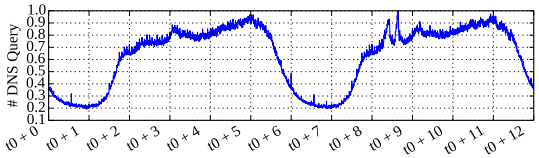


Figure 2.29: DNS requests count

example, applications that provide continuous online presence (e.g., WhatsApp, Facebook, Skype) might generate a big burden on the signaling plane, impacting network performance. Specific devices and applications might also cause undesirable overloading events due to synchronized communication patterns, typical for M2M applications. In this evolving scenario, detecting and rapidly diagnosing device-specific traffic misbehaviors becomes crucial for cellular network operators.

The proposed approach is articulated in two steps: **(i) Detection:** the *trigger* consists of detecting an abrupt change in the time series of specific traffic features revealing unexpected and potentially harmful behaviors. We call these time series *symptomatic signals*. From our operational experience, application-specific anomalies are particularly visible in the DNS traffic. Indeed, abrupt changes in the DNS requests count can be considered as a symptom of such anomalies. Therefore, we use the DNS requests count as the main symptomatic signal of the approach. The abrupt change detection is performed by a standard auto-adaptive algorithm, based on the mean and the variance of the DNS requests count.

(ii) Diagnosis: to find out the root causes of the detected anomalies, we define a set of features related to the class of problems we target, based on expert know how. In particular, we consider the following set of features, associated to each DNS request-response transaction: anonymized Mobile Device Identifier (ID), contacted DNS server IP, Radio Access Technology (RAT), Access Point Name (APN), Type Allocation Code (TAC), DNS requested Full Qualified Domain Name (FQDN), device manufacturer (obtained from the TAC code through public GSM Association databases), and error code of the DNS response (DNS rcode).

The first step of the diagnosis consists of identifying which of these features present a significant change in their probability distribution, simultaneously to the trigger. To this aim, we use the entropy as a means to condense the complete distribution of a feature into a single value for a given time bin. We refer to the time series of the entropy of these features as the *diagnostic signals*. The entropy of a random variable X is $H(X) = -\sum_{i=1}^{n} p(x_i)log(p(x_i))$, where $x_1, \ldots, x_n$ is the range of values for X, and $p(x_i)$ is the probability that X takes the value x_i. The entropy is normalized to a scaling factor $log(n_0)$, where n_0 is the number of distinct x_i values in a given time bin. Entropy-based approaches have been proposed for traffic analysis in the past, but in a fixed-line network context, and using only transport-layer features such as IPs and ports. Our analysis specifically targets cellular networks, using a much richer set of features, from network to device specific ones.

To detect changes in the diagnostic signals, we apply the same auto-adaptive algorithm used in the detection step. In the practice, different anomalies cause significant changes only in a subset of the considered diagnostic signals. This subset allows us to build-up a signature for the detected anomaly. Finally, by further drilling down into the features that correlate the most to the trigger, the approach permits to narrow down the causes for a certain anomaly: e.g., anomalies linked to a specific device type, a specific service failure, and so on.

As an application example, we present a case study based on the detection and diagnosis of a large scale anomaly occurred in a real cellular network. Fig. 2.29 shows the time series of the total DNS requests count observed in the network for two consecutive days. Two significant and anomalous

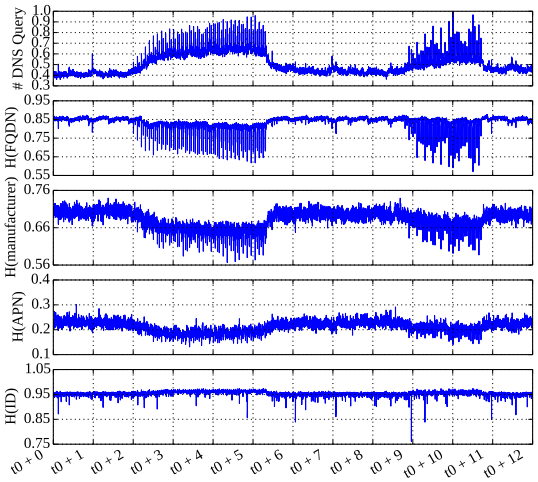


Figure 2.30: Entropy of selected features

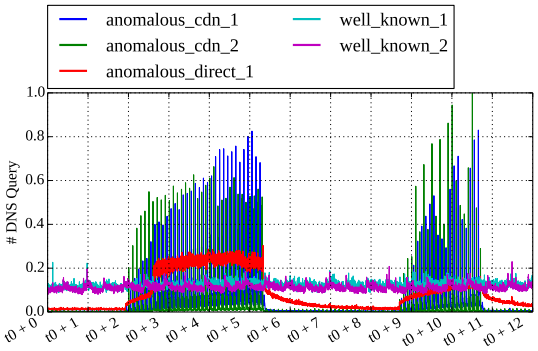


Figure 2.31: DNS requests per FQDN class

spikes are observed on the second day, which are easily spotted by the abrupt-change detection algorithm.

Fig. 2.30 provides a closer look into the anomaly, comparing the time series of the total DNS requests count and the entropy of 4 selected features: FQDN, manufacturer, APN, and ID. These features are extracted for each DNS request-response transaction (to preserve user privacy, any user related data are removed on-the-fly). The other diagnostic signals are omitted for brevity, as they show a behavior similar to the reported next. We notice that some of the observed diagnostic signals are correlated in a minor way to the anomaly. This is the case for ID, TAC, RAT, and DNS rcode, therefore we can exclude the cases in which the anomaly is caused by few users, a specific RAT, etc.. On the contrary, dimensions such as FQDN and manufacturer present a very high correlation with the spikes in the DNS count, suggesting that the issue might be due to specific devices (manufacturer) querying for certain services (FQDN). Features such as APN and server IP present a partial correlation with the anomaly, thus they need to be further cross-checked.

The next step of the diagnosis is to drill down each of the dimensions that are highly correlated with the anomaly. This can be achieved, e.g., by comparing the heavy hitters before and during the anomaly. Fig. 2.31 reports the specific case for the FQDN. The plot shows the time-series of the most requested FQDNs during the anomaly. We observe that, while some of the top FQDNs associated to well-known services present a stable behavior (`well_known_1/2`), the FQDNs `anomalous_cdn_1/2`

and `anomalous_direct_1` show a significant increase. The first two refer to content of a specific popular OTT service delivered via a major Content Delivery Network (CDN), whereas the third one points directly to the specific OTT service, showing that the problem is actually related to this service.

The mapping of the TAC codes to the manufacturer of the devices requesting the FQDNs related to the anomaly also reveal a specific smartphone type involved in the anomaly. In particular, the specific anomalous service runs on all these devices, but not on the other smartphone types. W.r.t. the dimensions presenting partial correlation, we found that all the different APNs are affected by the anomaly but in a different manner, suggesting that different APNs are configured for different customers. Indeed, different APNs are normally linked to different default DNS servers.

As a main conclusion, the proposed approach is helpful in highly reducing the time spent by the network operator in the diagnosis of unexpected traffic behaviors. In particular, this service outage resulted in an abrupt increase in the number of connection attempts from a large number of devices, and its fast diagnosis was paramount to understand the nature of such an anomaly.

2.7.5 Best and Worst Comparison

This section designs and evaluates a data analysis technique to help large scale network administrators to pinpoint the root cause behind QoS/QoE glitches. We employ Tstat to collect measurements from a PoP (Point of Presence) of an Italian ISP. For each observed TCP connection, Tstat generates a log, i.e., a line reporting a large variety of statistics (client IP address, server IP address, data-rate, retransmitted packets, DNS server etc.). Logs are then collected into files which constitute the input for our analysis.

The rationale behind this methodology is to analyze the traffic measurements, classify clients based on their perceived QoS and discriminates those experiencing good QoS (named Best) from those unable to get a satisfactory QoS (named Worst). Thus we compare the traffic exchanged by these two sets of clients through different tests in order to highlight differences and correlations that may hide the cause of such poor performance.

2.7.5.1 Methodology

We can split the methodology, as depicted in Fig. 2.32, in two phases: 1) identify the connections for the Best and the Worst clients, i.e., connections generated by client achieving a good QoS (best) and the connections of clients experiencing bad QoS performance (worst) and 2) make a comparison between the two connection sets.

First of all, however, we choose a metric of interest, i.e., a primary metric. It determines the feature according to which Best and Worst connection sets are defined. The throughput and the server response time are typical choices. Then, we clean the measurement data: using a multi-stage approach composed by three filters: i) the *Raw filter* selects the metric(s) of interests to define the Best and Worst sets of connections; ii) *Service filter* selects the connections belonging to the service we are interested in. iii) *Pruning filter* removes non significant measurements for the selected primary metric, e.g., we remove flows carrying small amount of data. Then, we obtain the empirical cumulative distribution for the considered primary metric and we employ it to identify Best and Worst connection sets. E.g., as depicted in Fig. 2.33, we employ the throughput CDF to define the Best set of connections as the top 90%, and the Worst set as the bottom 10%.

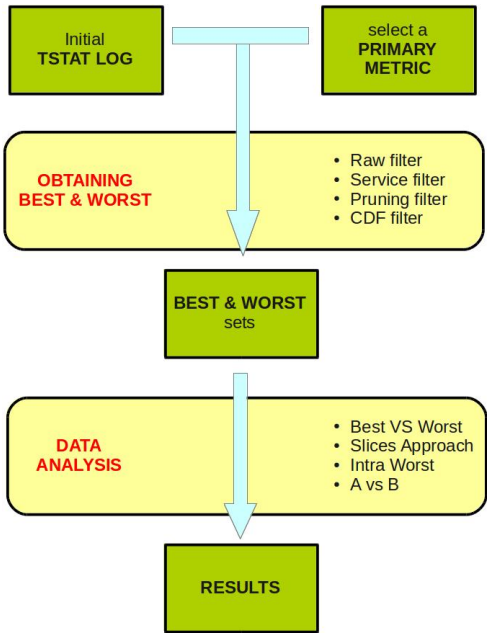


Figure 2.32: Methodology workflow

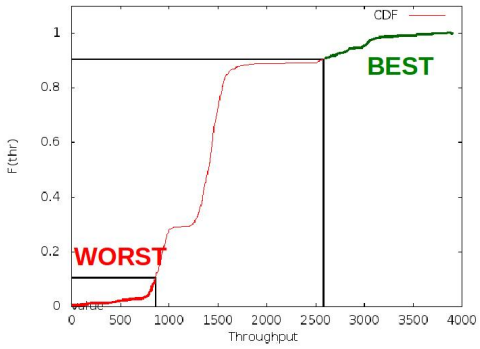


Figure 2.33: Example of throughput distribution

The next step consists of four different automatic tests to analyze measurements from different perspectives.

1. The first test is called *Best vs Worst*. The idea is to compare Best and Worst connections metric by metric (i.e., client IP address, retransmissions, etc.). This test looks for differences among the two sets of connections. Intuitively, a difference over a given metric may indicate a correlation with the cause behind the performance difference between Best and Worst connection sets. In detail, we perform the comparison by building the empirical probability density function for all the metrics that have been filtered for both Best and Worst sets. Then, we compare the two resulting PDFs through a well known similarity measure called Cosine Similarity (which varies between 0 and 1). Intuitively, if the similarity value is below a threshold T_s, the metric might be correlated to the poor performance of the primary metric. Vice-versa, if the PDFs of the considered metric is similar for both Best and Worst sets, we can exclude a correlation with the primary metric. Fig. 2.34 show an example in which Best and Worst

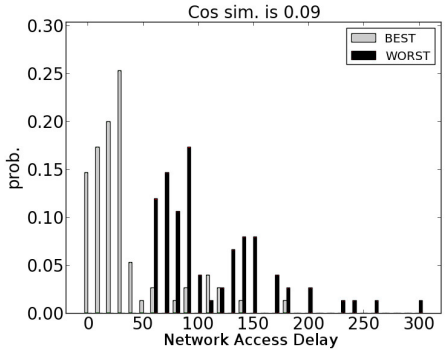


Figure 2.34: PDF comparison for network access delay

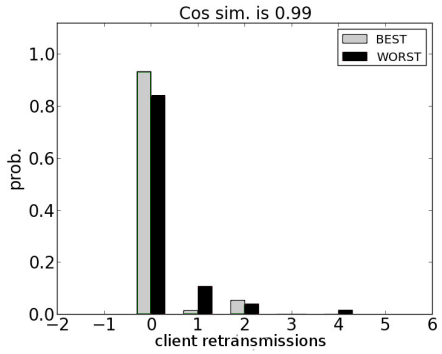


Figure 2.35: PDF comparison for client retransmissions

Signature Database	
Rule	Signature
RTT diff + TTL diff + SERVER diff	Farer servers
RTT diff + TTL sim + SERVER sim	Congestion at servers
CLIENT diff + NA diff	Network Access Delay issue
Legend - “sim” : similar PDFs , “diff” : different PDFs	

Figure 2.36: Example of signatures

PDFs are very different (notice the Cosine Similarity is really low). Instead, Fig. 2.35 reports the an example in which the two distributions are very similar (Cosine Similarity close to 1).

Alongside the comparison task, we populate a database with “problem identification signatures”, i.e., we associate recognizable rules to known problems, e.g., we associate signature “bad Round Trip Time (RTT) + bad Time To Live (TTL) + different server IP addresses” to the problem “far away servers”. Further examples are given in Fig. 2.36. This test has the potential of spotting the problem that affects the connections in the Worst set and at correlating metrics among them. However, if fails at detecting anomalies that involve a small number of connections.

2. The second test is called *Slice Approach*. Starting from Best and Worst connection sets, we group them according to a certain value of a metric, e.g., a server IP address or a specific TTL value. The resulting subsets of connection logs are then compared. This test allows us to

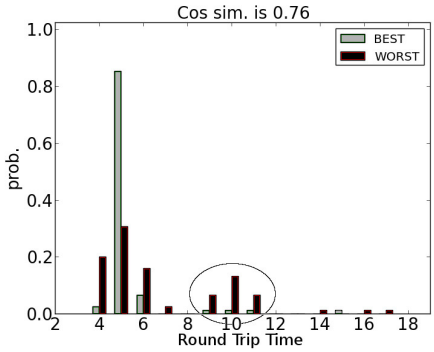


Figure 2.37: PDF comparison for Round Trip Time. The small anomaly affecting connections in the Worst set is not highlighted by the Cosine Similarity

Servers contacted by Worst clients				
Server IP	*Size*	*Avg. RTT*	*Avg. TTL*	*Avg. NA*
173.194.1.3	23	13	51	87
78.23.100.2	26	16	51	88
4.231.1.5	25	**28**	**48**	91
~~13.15.5.2~~	~~10~~	~~31~~	~~51~~	~~87~~

Figure 2.38: *Intra Worst* comparison among server IP addresses

detect small anomalies that could not be pinpointed through the Cosine Similarity approach, e.g., Fig. 2.37 shows how a small number of large Round Trip Time values is not enough to obtain a low Cosine Similarity. However, this approach has the limit of introducing unbalances in the datasets to compare: we can indeed perform the comparison only when a value is common to both Best and Worst sets and only if the two subsets have similar sizes.

3. The third test is called *Intra Worst* and performs further analysis within the Worst connection set only. We group by a particular metric (server IP addresses, client IP addresses, TTL, etc.) and obtain different subsets of connection logs. A minimum number of observations is needed to enable fair comparison. Every subset is collapsed into a single record that contains aggregate statistics (for example the average) computed for all the metrics involved in the analysis (except the one we group by on). We thus look for the outliers, values very far from the standard and, i.e., representing particularly bad behaviors. For example, we compare server IP addresses to identify those which are affected by congestion or which are far away from the clients. Fig. 2.38 shows an example in which servers contacted by worst client are compared among them. One subset is not eligible to comparison due to its smaller size. One server seems to be farther.

4. The last test is called *Metric Comparison*. First, two metrics, A and B, are selected. Then, we form couples a,b, where $a \in$ A and $b \in$ B, and create the correspondent subsets of connection logs. For example, we decide to use client IP subnet as A and server IP address as B. We aim at finding bad behaviors that take place for a particular client subnet (or server) independently on the contacted server (or originating client subnet). Also in this case we detect anomalies by looking for outliers.

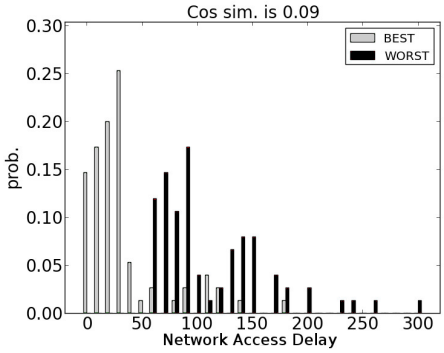


Figure 2.39: *Best vs Worst* test on Access Network Delay metric

Client subnets in the Worst set			
Client IP	*Avg. TTL*	*Avg. RTT*	*Avg. NA*
12.30.0.0	**49**	**10.89**	161.04
12.132.0.0	52	5.45	85.36
12.133.0.0	52	4.74	112.08
12.127.0.0	51.71	6.07	88.52

Figure 2.40: Intra Worst dataset for client subnet comparison

2.7.5.2 Preliminary Results

We evaluate our methodology on a collection of Tstat logs we collected at a vantage point located within the network of country-wide European ISP. Our dataset is 1-day long, in particular the measurements it contains were collected on the 2nd of April 2014. For our analysis we focused on TCP connections carrying HTTP traffic only. The dataset is composed of about 38 million connection logs referring to 9837 distinct clients. For the results we report in this document, we select the throughput as the primary metric and focus only on the connections directed to `googlevideo.com`, the service responsible for carrying YouTube video content. We start our analysis by running the *Best vs Worst* test. Results reported in Fig. 2.39 shows that the Access Network Delay metric is very different for connections in Best and Worst sets. This suggests that the Access Network Delay is strongly correlated to the download throughput.

We also observe that a set of connections is affected by small TTLs and large RTTs, but the size of this set is too small to influence the Cosine Similarity. We thus run the *Slice approach* test which confirms that the Access Network delay still seems to be the most critical metric for performance. Some values, like a client subnet or a specific TTL value, were observed in the Worst set only thus it was not possible to perform the comparison. We thus run the *Intra Worst* test to compare the aggregate statistics of different client subnets. As depicted in Fig. 2.40, it emerges that a client subnet found in the Worst set shows smaller TTL values and larger RTT values, suggesting the presence of a routing anomaly. We can confirm this by grouping on TTL. Indeed, we notice that all the connections with the lowest TTL value came from that particular subnet.

Finally, thanks to the *Metric Comparison* test, we can confirm this observation, as regardless the contacted server we notice that the aforementioned subnet shows worse TTL and RTT.

Thanks to these four tests we were able to detect the anomalies affecting the Worst clients. The biggest factor worsening performance for some clients is a large delay in the Access Network. This

affects all the clients belonging to the Worst set. Then, we detected a smaller issue involving only the clients belonging to a particular client subnet. This client subnet seems experience bad performance due to a misconfiguration in its routing: clients follow longer paths and take more time to reach servers.

2.8 Verification and Certification of Service Level Agreements

The principal algorithms that are used by the Reasoner for the verification and certification of service level agreements (SLA) are:

1. The UDP speed correction algorithm

2. The irregularity detection algorithm

3. The anomaly detection algorithms, by correlating the active and passive data

We will explain here the first two algorithms, since the third one is still one process and there is no concrete algorithm yet. The UDP speed correction algorithm it is used almost every time there is identified an packet loss >0.1% on the UDP transfer. The algorithm is explained on Fig. 2.41.

As it can be seen, on every SLA verification that is done between server and client, after all the different test, RTT and TCP throughput, the UDP test is carried out. At the beginning the probe forces the transmission at the maximum speed, usually nearly 1000Mbps. After the algorithm gets the data of the first test and calculates the correct bandwidth Bc. After which it tests a bandwidth Bc* that has a difference 1/1000*Bc from Bc. If the test reports that the packet loss is < 0.1% than the reason accepts Bc* as the correct bandwidth if not it will repeat the test until it finds the correct bandwidth. For more dettail see [87]

The irregularity detection algorithm it is used to find on the active data irregularities, usually represented as unexpected drop of the bandwidth. To detect the measurements that are affected by this kind of "anomalies" the algorithm applies two rules, Fig. 2.42 and Fig. 2.43. If the rules are true than the data is processed by the anomaly detection algorithm (which is still in progress, waiting for lab results).

In Fig. 2.42, representes the firs rule, that can be explained as follows: Being "Mi" the report of the bandwidth every second, for an measurement we have "n" samples "Mi". We calculate a threshold as 70% (this value can differ in dependence of previous results of the bandwidth)of the mean of all samples. We assume that the data is affected by anomaly if 30% of the samples are below the threshold.

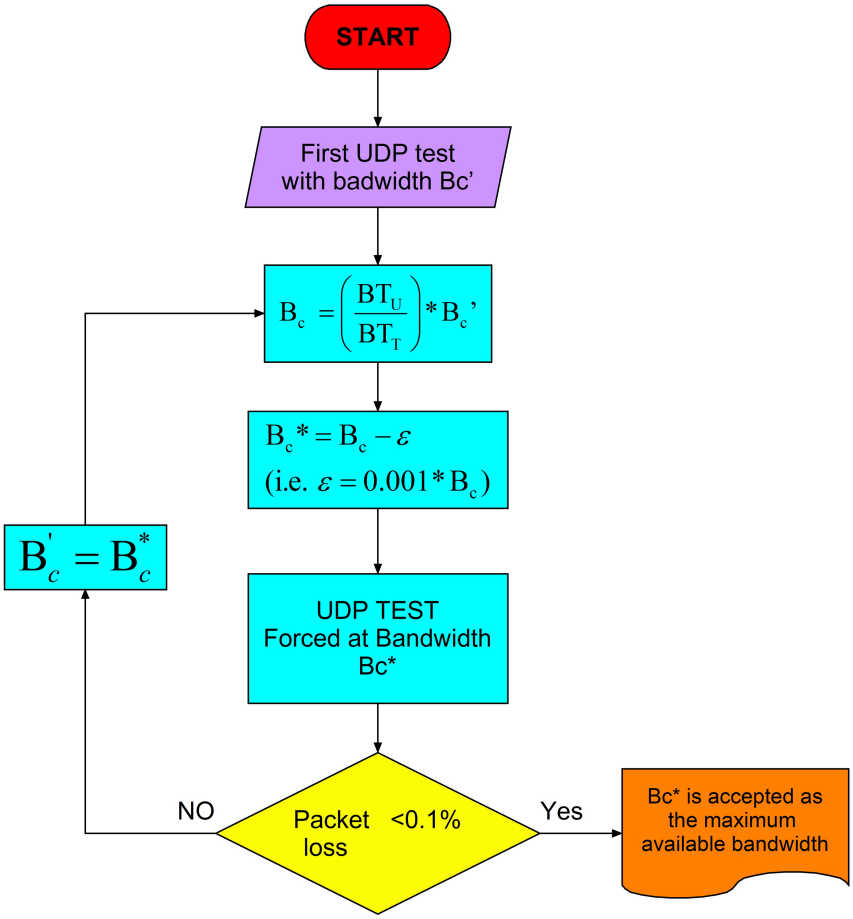


Figure 2.41: UDP speed correction algorithm

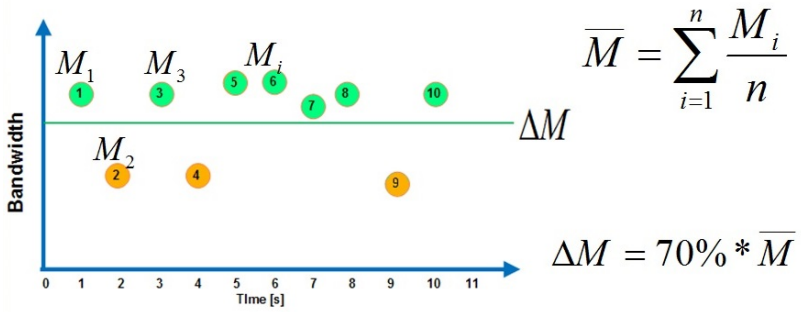
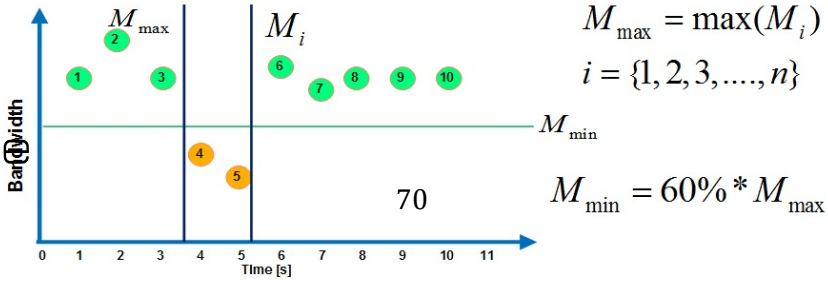


Figure 2.42: UDP speed correction algorithm



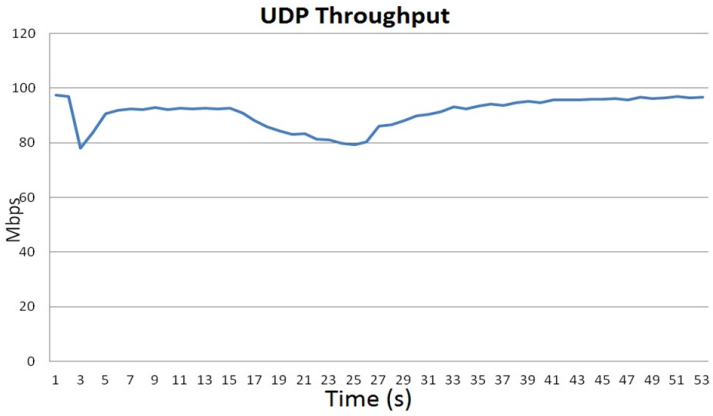


Figure 2.45: Bandwidth on client 1, using UDP

In Fig. 2.43, it is presented the second rule of the anomaly detection. The second rule is more straight through. The rule says that the measurement has to be tested on a sliding window of 20% of the total window. First there a threshold Mmin set as 60% of Mmax (the value may differ depending on previous values), where Mmax is the sample with the maximum value among all the samples. That the rule states that the data is affected by anomaly if 20% of the consecutive samples is below the Mmin threshold.

2.8.1 Bandwidth Overbooking

There might be some case when the ISP might use bandwidth overbooking. The ISP tends to book more bandwidth that it has at his disposal. In this cases some services might degrade from time to time, depending on the bandwidth usage of the other clients. The client in this case can measure a minimum bandwidth, and in some case also a maximum bandwidth. To test this particular case we considered the scenario in Fig. 2.44

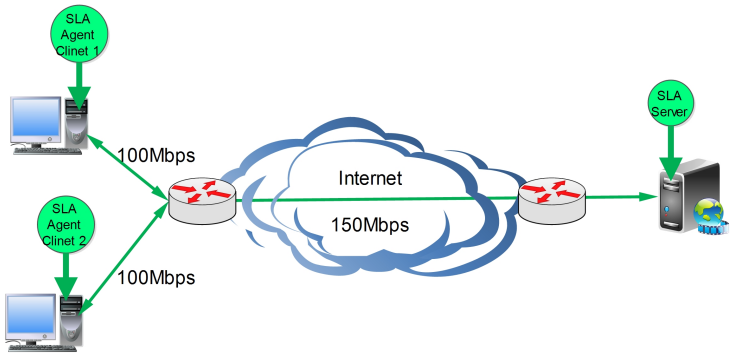


Figure 2.44: Scenario experimented on overbooking

In the case of Fig. 2.44, we have two client with a tested connection that has gave us a result of 100Mbps each. Conversely the total available bandwidth of the shared channel has been set at 150Mbps. So the singular tests on each user are correct, but in the case both users are active the results differs as shown in Fig. 2.45 and Fig. 2.46 By making the sum of the result in Fig. 2.45 and

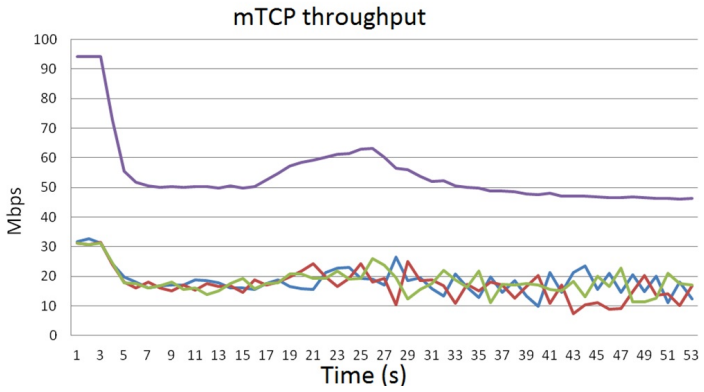


Figure 2.46: Bandwidth on client 2, using multisession TCP

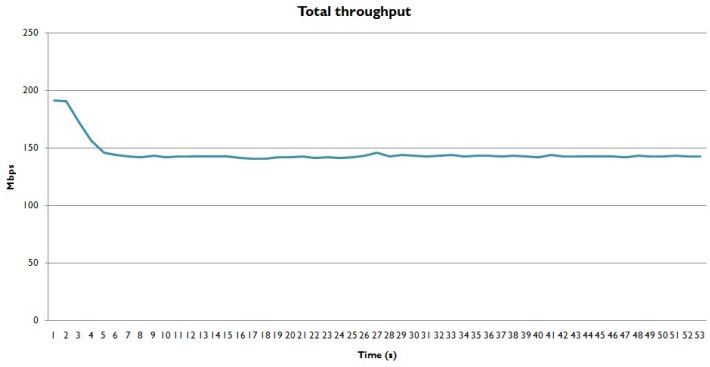


Figure 2.47: Total throughput on both clients

Fig. 2.46, we can obtain the total available bandwidth, of 150Mbps as shown in Fig. 2.47.

To detect bandwidth overbooking the algorithm in Fig. 2.48 can be used. Although this algorithm depends very much on the number of the clients behind at the access network. To get a more precise result all the client should be included in the test.

To detect overbooking, we just make singular test on each client, and then the second step is to make parallel test on all users at the same time. If the bandwidth for every user on the second test differs to much from the first test than we have overbooking and the total bandwidth of the Internet provider is calculated as the sum of the bandwidth of every user. Other cases of anomaly were published at mPlane D4.2 [73] and [101].

2.9 Network Proximity Service Based On Neighborhood Models

The knowledge of network proximity is critical to achieve Quality-of-Service (QoS) guarantees for end users. For example, in Peer-to-Peer (P2P) applications and Content Distribution Networks (CDNs), it is desired to direct user requests to a peer or a server that is nearby and thus with a connection of small latencies. In this section, network latency refers only to round-trip time (RTT).

To exploit the proximity information in large distributed systems, a practical challenge is the efficient acquisition because active probing of network latency for all paths in a large network is infea-

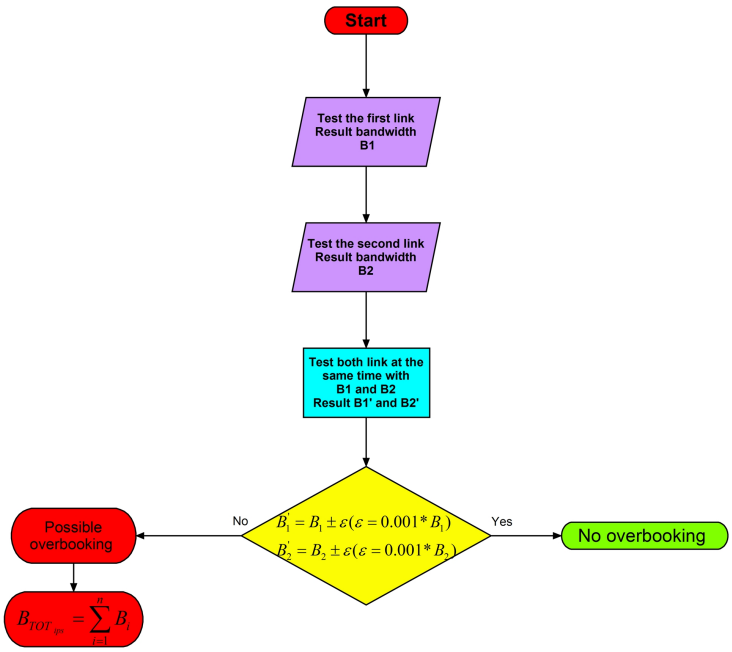
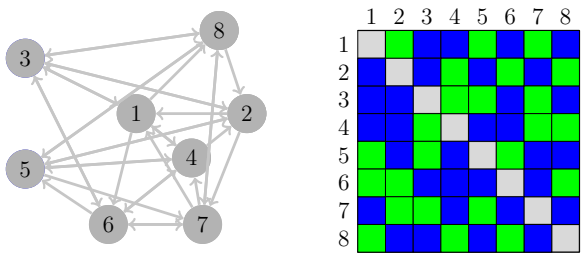


Figure 2.48: Algorithm for detecting overbooking

sible due to the quadratic complexity. This issues has been well studied, resulted in a rich line of research on network proximity inference based on latency measurements on a few paths [22, 66, 75]. In particular, a recent progress is that the inference of network latencies can be cast as a matrix completion problem whereby a partially observed matrix is to be completed [26, 27, 61, 62]. Here, the matrix contains latencies between network nodes with some of them known and the others unknown, shown in Fig. 2.49(a). We then observed a similarity between network inference and the problem of *recommender systems* which studies the prediction of preferences of users to items [56], shown in Fig. 2.49(b). If we consider network latency as a preference measure between nodes, then peer and server selection is just a recommendation task. This observation enables us to leverage the rapid advances in machine learning and investigate the applicability of various solutions to recommender systems for network inference. Our previous studies have shown that a class of matrix factorization techniques are suitable and achieved good results that are known to be acceptable for recommendation tasks [26, 27].

Alternatively, neighborhood models are also widely used in recommender systems which exploit the similarities between users and between items [89]. For example, two users are considered similar if they rate a set of items similarly. Meanwhile, two items are considered similar if they are given similar ratings by a set of users. Thus, two kinds of recommendations can be made to a user: liked items by similar users and items that are similar to the liked items by that user. In this section, we develop a lightweight network proximity service based on neighborhood models whereby, if two nodes have similar proximity to a number of common nodes, the two nodes are likely to be close to each other. Different from previous work, our approach infers proximity without recovering network latencies. A simple proximity measure is computed and can be exploited for ranking and rating network paths which is useful in e.g. P2P and CDN applications for peer and server selection. The approach is highly scalable and involves only the latency measurement by `ping` from each node to a small number of pre-selected landmark nodes which can be servers at Google, Yahoo and Facebook, etc. Simulations on existing datasets and experiments with a deployment on a real



(a) A matrix completion view of latency inference. In the matrix, the blue entries contain measured path latencies, represented by directed edges in the graph, and the green entries are missing

	item1	item2	item3	item4	item5
user1	5	3	4	1	?
user2	5	3	4	1	5
user3	5	?	4	1	5
user4	1	3	2	5	1
user5	4	?	4	4	4

(b) Recommender system with preferences in the scale of $\{1, 5\}$

Figure 2.49: Connection between network inference and recommender systems.-

network, namely PlanetLab, showed that our approach provided accurate proximity services that are comparable to state-of-the-art latency inference approaches, while being much simpler.

2.9.1 Network Proximity Service

2.9.1.1 Measuring Network Proximity

Instead of measuring latencies between network nodes, we require each node to probe the latencies to a number of landmark nodes and the latency measurements are put in a feature vector which is attached to each node, illustrated in Fig. 2.50. Let $v_i = [v_{i1}, \ldots, v_{ik}]^T$ be the feature vector of node i, where v_{ij} is the latency between node i and landmark j and k is the number of landmarks. The landmark nodes can be any IP addresses on the Internet such as content servers in Google, Yahoo and Facebook or news and university web servers which stay alive stably and respond to the `ping` measurement. Note that anycast IP addresses such as Google public DNS servers at $8.8.8.8$ and $8.8.4.4$ need to be avoided.

A similarity/dissimilarity measure between the feature vectors of two nodes can then be computed using e.g. the correlation, the L2 or L1 norm and the cosine of the included angle, etc., among which the L1 norm is chosen due to its robustness to noisy and outlier measurements [45], given by

$$p_{ij} = \frac{1}{k} \sum_{l=1}^{k} |v_{il} - v_{jl}|. \tag{2.6}$$

p_{ij} is a proximity measure, although it does not inform us about the actual latency between i and j.

Intuitively, when p_{ij} is small, i.e. i and j have similar latencies to the landmarks, chances are that i and j are close to each other. This is exactly the idea in neighborhood models for recommender systems.

Thus, the proximity measure carries information that can be exploited to rank network nodes. For example, if $p_{ij} < p_{il}$, then we can guess that i is closer to j than to l, i.e. the latency between i and j is smaller than that between i and l. Obviously, proximity inference based on p_{ij} is less accurate than based on measured latencies. However, a proximity service based on active probing would require $O(n^2)$ measurements for a network of n nodes which does not scale well. In contrast, our new service reduces the measurement overhead from $O(n^2)$ to $O(kn)$, with k likely to be independent of n. In addition, comparing to state-of-the-art latency prediction approaches, our proximity service is much simpler and requires no computation for learning a prediction model such as Euclidean embedding for Vivaldi [22] and matrix factorization for DMFSGD [62].

Note that it is possible that a node has a poor connection to all landmarks, i.e. $\min(v_i)$ is larger than a threshold. We consider such cases as that the node has a poor connection to the entire Internet, due probably to a poor Internet access link, and turn the proximity measure between that node and any other node in the network to a large constant. This prevents nodes with poor Internet connections from being labeled as close to each other.

2.9.1.2 Landmark Selection

It is easy to see that the proximity measure depends on both the locations and the numbers of the landmark nodes. In practice, we can check the suitability of the landmarks by computing statistics such as variance of the latencies. There are two considerations.

- For node i, v_i is more informative about its location if some latencies in v_i are small and some are large, i.e. i is close to some landmarks and far away from some others. Thus, the lack of variance in v_i is generally a good indicator of the poor choice of the landmarks for node i. If many nodes have small variance in their feature vectors, a likely reason is that many landmarks are in the same region and close to each other.
- For each landmark, we can also construct a feature vector containing latencies from each node to the landmark. Let $v_i^L = [v_{1i}, \ldots, v_{ni}]^T$ be the feature vector for landmark i. Similarly, the lack of variance in v_i^L is also a good indicator that landmark i is not suitable to serve as a landmark for the nodes in the network, which happens when the landmark has an anycast IP address[4].

[4] Another possible reason is that most nodes in the network are in the same region and close to each other. This is a

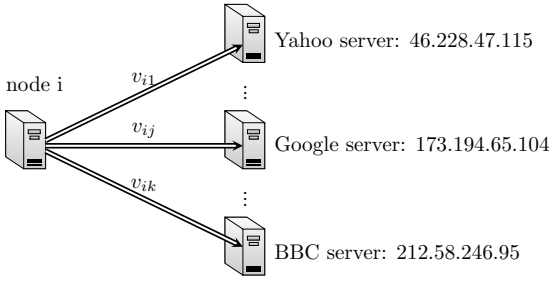


Figure 2.50: Network proximity service

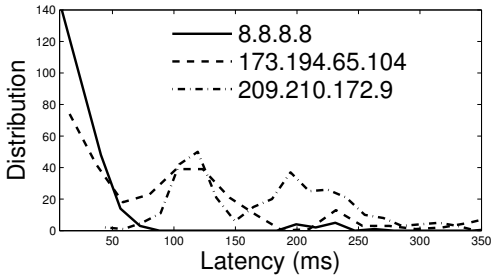


Figure 2.51: Distributions of RTT to an anycast Google public DNS server at 8.8.8.8, a unicast Google Web server at 173.194.65.104 and a unicast DNS server at 209.210.172.9

Essentially, we can construct a matrix containing latencies between nodes and landmarks, denoted by $V = [v_1, \ldots, v_n]$, and it is desired that the variance in both the rows and the columns in V are large. If we have a set of landmarks from which we choose a few, we can design a selection algorithm that maximizes the variances in V. Empirically, we found that if the landmarks are well distributed all over the world, the variances in V are generally large enough and a good network proximity service can be achieved using a small number of randomly selected landmarks, shown in Sec. 2.9.2.

We performed a test on PlanetLab on June 2$^{\text{nd}}$, 2014 that from 592 live nodes we pinged an anycast Google public DNS server at 8.8.8.8, a unicast Google Web server at 173.194.65.104 and a unicast DNS server at 209.210.172.9 and received latencies to all three servers from 310 nodes, shown in Fig. 2.51. We can see that both the mean and the variance of the latencies to 8.8.8.8 are much smaller, which is expected as 8.8.8.8 corresponds to 28 servers at different locations. Note that among all the latencies to 8.8.8.8, only 12 are larger than 100ms and the latencies from all 11 Chinese nodes are larger than 180ms[5]. This suggests that even an anycast IP may provide useful proximity information about nodes in particular areas. For example, a node is unlikely to be in China if its latency to 8.8.8.8 is smaller than 100ms. By choosing the right landmarks, we may design a location classifier that pinpoints a network node to a small region.

2.9.1.3 RSD-Based Approach

It is worth to notice that proximity can also be estimated using other metrics/techniques than delay measurements. In this section, we explain how the *routing state distance* (RSD) [42] recently introduced can be used. Further work should reveal the relevance or not of this approach.

2.9.1.3.1 Routing State Distance Recently, Gürsun et al. [42] have proposed a metric, called *routing state distance* (RSD), to quantify the diversity of BGP routes. In particular, RSD considers all the collected BGP routes to each prefix pair to compute and compare the routing states of the two prefixes.

More formally, let us consider a universe X of n nodes (i.e., $|X| = n$). Gürsun et al. define $N(x_1, x_2)$ as the next-hop on the path from x_1 to x_3.[6] By following recursively nodes on $N(\cdot, x_2)$, one finally reaches x_2. Generally, N is seen as a $N \times N$ matrix where $N(x', x)$ can be interpreted as a matrix

trivial situation which we rule out in this section.

[5]The only other node with a latency to 8.8.8.8 greater than 100ms is lim-planetlab-2.univ-reunion.fr. The mean latency is 213ms and the variance is 4ms. The node is in the University of La Réunion located in an island in the Indian Ocean.

[6]Note that $N(x, x) = x$, i.e., any node x is the next-hop the node x itself.

element that stores the next hop from x' to x. Finally, it is worth to notice that $N(x,:)$ (respectively, $N(:,x)$) denotes the $x-th$ row (respectively, column) of N.

Based on that, the RSD between two nodes x_1 and x_2 is defined as follows:

$$RSD(x_1,x_2) = |\{x_i | N(x_i,x_1) \neq N(x_i,x_2)\}| \tag{2.7}$$

In other words, $RSD(x_1,x_2)$ is the number of positions where the columns $N(:,x_2)$ and $N(:,x_1)$ differ. RSD is thus an integer that takes a value in $0,1,\ldots,n$.

2.9.1.3.2 Ranking Probes Based on RSD For our problem, we need to apply RSD to BGP data. As suggested by Gürsun et al. when considering BGP data, one might apply implementation changes in RSD. Now, in the matrix N, a column corresponds to prefixes and a row corresponds to ASes. Thus, $N(a,p)$ is defined as the next hop from AS a to prefix p.

Gürsun et al. define the routing state $RS(p)$ of a prefix p as the directed graph obtained by merging all AS-paths in BGP routes for p. Then, they define the routing state distance between two prefixes p_1 and p_2 as $|RS(p_1) \oplus RS(p_2)|$, where $\oplus$ denotes the XOR between the two graphs (i.e., their union minus their intersection).

The RSD-based approach works as follows:

1. For every probe, its RSD distance to the point of interest is calculated.
2. A ranking of probes is produced, which is a list of probes in increasing order of RSD distances to the PoI.

2.9.2 Simulations on Existing Datasets

We performed simulations on the following datasets:

- **Meridian** contains static RTTs between 2500 DNS servers obtained from the Meridian project [107].
- **P2PSim** contains static RTTs between 1740 DNS servers obtained from the P2PSim project [41].
- **Harvard** contains dynamic RTTs between 226 PlanetLab nodes collected in 4 hours [58].

In these datasets, as we only have RTTs between nodes in the networks, we randomly select a number of nodes as landmarks and the feature vector of each node consists of RTTs to those selected nodes. For the Harvard dataset, we extracted the static RTTs by computing the mean RTT between each pair of nodes and used them in the first two subsections for evaluating the ranking and rating accuracy. The dynamic RTTs in the Harvard dataset are used to evaluate the stability of the proximity measure only in the last subsection.

2.9.2.1 Ranking of Network Proximity

We first evaluate the ranking of network proximity using the Spearman's rank correlation coefficient which is defined as the Pearson correlation coefficient between the ranked variables [12]. For

Table 2.6: Impact of k on ranking accuracy

ρ	mean	std
k=10	0.784	0.017
k=20	0.802	0.010
k=30	0.809	0.008
k=60	0.819	0.003

Meridian

ρ	mean	std
k=10	0.786	0.038
k=20	0.813	0.026
k=30	0.820	0.016
k=60	0.828	0.011

P2PSim

ρ	mean	std
k=10	0.942	0.020
k=20	0.948	0.008
k=30	0.952	0.006
k=60	0.952	0.003

Harvard

Table 2.7: Comparison of ranking accuracy

ρ	mean	std
Neighborhood	0.811	0.007
DMFSGD	0.823	0.001
Vivaldi	0.807	0.002

Meridian

ρ	mean	std
Neighborhood	0.819	0.020
DMFSGD	0.889	0.002
Vivaldi	0.834	0.002

P2PSim

ρ	mean	std
Neighborhood	0.952	0.006
DMFSGD	0.910	0.003
Vivaldi	0.868	0.002

Harvard

two sequences X and Y, the raw data X_i and Y_i are converted to the ranks x_i and y_i in the sequence, and the Spearman's rank correlation coefficient is computed as

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 (y_i - \bar{y})^2}}. \tag{2.8}$$

ρ is between 1 and -1, and the larger it is, the more the ranks of X and Y are positively correlated. Thus, we evaluate the ranking accuracy by calculating the Spearman's rank correlation coefficient between the proximity measures by our service and the true latencies between network nodes. In other words, the two sequences X and Y are our proximity measures and the true latencies, and we wish a high ρ value so that their rankings match each other.

As the landmarks are randomly selected from all nodes in the network, we are interested in the impacts of the number of landmarks on the accuracy of proximity inference. We tested $k = 10$, 20, 30 and 60 respectively, with 10 runs of random landmark selection for each k, and the mean rank correlation coefficients and the standard deviations for each dataset are shown in Table 2.6. It can be seen that the rank correlation improves with the increase of the landmark number k and that the ranking results are stable and not sensitive to the random selection of the landmarks in the network. Overall, ranking of network proximity is more accurate on the Harvard dataset than on the other two, due probably to its small size of the network, i.e. 226 nodes.

We then compare our proximity service with two popular latency prediction approaches, namely *Vivaldi* [22] and *DMFSGD* [26,61,62]. The former predicts RTTs based on Euclidean embedding and the latter does so based on matrix factorization. Both employ the same architecture that each node probes and exchanges messages with k randomly selected neighboring nodes in the network. In contrast, our service based on neighborhood models only probes k landmarks, with no requirement of message exchanges between nodes. To make the comparison fair, we set $k = 32$ so that all methods have the same measurement overhead. Note that $k = 32$ is the default setting in Vivaldi and DMFSGD. We ran the simulations for 10 times with random landmark and neighbor selection for our service, DMFSGD and Vivaldi respectively. The rank correlation for DMFSGD and Vivaldi is calculated by comparing the ranks of the predicted RTTs and of the true RTTs using eq. 2.8. The mean rank correlation coefficient and its standard deviation for each method is shown in Table 2.7. It can be seen that our service achieved comparable results with DMFSGD and Vivaldi.

Table 2.8: Comparison of rating accuracy

RMSE	mean	std
Neighborhood	0.943	0.014
DMFSGD	0.860	0.003
Vivaldi	0.945	0.003
Meridian		

RMSE	mean	std
Neighborhood	0.922	0.058
DMFSGD	0.667	0.004
Vivaldi	0.879	0.003
P2PSim		

RMSE	mean	std
Neighborhood	0.570	0.028
DMFSGD	0.601	0.009
Vivaldi	0.704	0.007
Harvard		

2.9.2.2 Rating of Network Proximity

We then evaluate the rating of network proximity that turns a proximity measure into an ordinal number in the range of $\{1, 5\}$. Ordinal rating is a loose version of ranking that labels a measure as rank 1 if it is among the top 20 percent smallest, as rank 2 if between top 20 and top 40 percent, and so on. While less informative, the advantage of rating over ranking is that rating measures are more stable over time. Thus, we compare the ratings of the proximity measures returned by our service with the ratings of the true latencies, evaluated by using the same criterion, *Root Mean Square Error* (RMSE), as in [26], given by

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}}. \tag{2.9}$$

The smaller the RMSE, the better. We also compare our service with DMFSGD and Vivaldi. DMFSGD can directly predict ratings of RTTs, as described in [26,27]. For Vivaldi, we turn the predicted RTTs into ratings.

As above, we ran the simulations for 10 times with random landmark and neighbor selection for our service, DMFSGD and Vivaldi respectively, and calculated the mean RMSE and its standard deviation for each method, shown in Table 2.8. It can be seen that on Meridian and P2PSim, DMFSGD is the best, while on Harvard, our service based on neighborhood models is the best. Overall, on the rating performance, our service achieved results at least comparable to Vivaldi.

Table 2.9 shows the confusion matrices by our service. In these matrices, each column represents the predicted ratings, while each row represents the actual ratings. Thus, the diagonal entries represent the percentage of the correct prediction, and the off-diagonal entries represent the percentage of "confusions" or mis-ratings. For example, the entry at $(2, 2)$ represents the percentage of the rating-2 paths which are correctly predicted as rating-2, and the entry at $(2, 3)$ represents the percentage of the rating-2 paths which are wrongly predicted as rating-3, i.e. the confusions from rating-2 to rating-3. It can be seen that while there are mis-ratings, most of them have a small error of $|x_{ij} - \hat{x}_{ij}| = 1$, marked as shaded entries in the confusion matrices. This means that the mis-ratings are under control. For example, a rating-5 path may be wrongly predicted as 4, but seldom as 3, 2 or 1, since the entries at $(5, 3)$, $(5, 2)$ and $(5, 1)$ in all confusion matrices are small.

2.9.2.3 Stability over Latency Dynamics

We further evaluate the impact of the latency dynamics on the stability of our service using the Harvard dataset which contains $2,492,546$ dynamic RTTs with timestamps collected in 4 hours from PlanetLab [58]. In the dataset, about 94.0% of the paths between pairs of nodes are measured between 40 and 60 times.

In the experiment, we assume that each node probes the landmarks once and the first RTT measure-

Table 2.9: Confusion matrices

	1	2	3	4	5
1	**81%**	14%	4%	1%	0%
2	13%	**58%**	20%	7%	2%
3	3%	21%	**38%**	26%	12%
4	1%	5%	27%	**39%**	27%
5	1%	2%	12%	27%	**58%**

Meridian

	1	2	3	4	5
1	**54%**	26%	10%	10%	0%
2	40%	**37%**	16%	6%	1%
3	6%	34%	**49%**	10%	1%
4	0%	3%	24%	**54%**	18%
5	0%	0%	0%	20%	**79%**

P2PSim

	1	2	3	4	5
1	**77%**	21%	2%	0%	0%
2	23%	**57%**	19%	1%	0%
3	0%	21%	**64%**	13%	2%
4	0%	3%	15%	**70%**	15%
5	0%	0%	0%	17%	**83%**

Harvard

ment from each node to each landmark is put in the feature vector of the node. Thus, the proximity measures between nodes are computed using the latency information acquired in the beginning of the simulation which are not updated over time. We then evaluate how the ranking and rating accuracy is affected by the dynamics of the true RTTs in the dataset. To this end, we ran the simulations with the RTTs between nodes updated using the timestamps in the dataset and computed the Spearman's rank correlation coefficient ρ and the RMSE over time (in every three minutes) between the proximity measures and the true, updated RTTs (turned into rating when computing the RMSE), shown as the curve of ρ *for Neighborhood* and of *RMSE for Neighborhood* respectively in Fig. 2.52. In addition, we also calculated the ranking correlation ρ over time between the RTTs in the beginning of the simulation (the first measurement between each pair of nodes) and the true, updated RTTs. This measure reflects the dynamics of the RTT measurements in the dataset, shown as the ρ *for dynamic RTT* curve in Fig. 2.52. It can be seen that the RTTs in the dataset are fairly stable, with the rank correlation over time never smaller than 0.95. In such cases, our service is able to provide accurate proximity inference, i.e. a rank correlation around 0.9 and a RMSE around 0.7, without updating the feature vector of each node for 4 hours. Note that if we do update the latencies in the feature vector of each node by a running mean, i.e. the mean of 10 most recent measurements, we only improve the accuracy slightly by less than 5%, at the cost of more probes to the landmarks.

2.9.3 Deployment on PlanetLab

We deployed our network proximity service on PlanetLab to test the performance when using landmarks other than those in the network.

2.9.3.1 Experimental Setup

We first identified the IPs that can be used as landmarks. The first landmark set, so called the *DNS* set, contains the DNS servers in the P2PSim dataset in which we found 792 out of 1740 alive and reachable. The second landmark set, so called the *WEB* set, contains the web servers at Google,

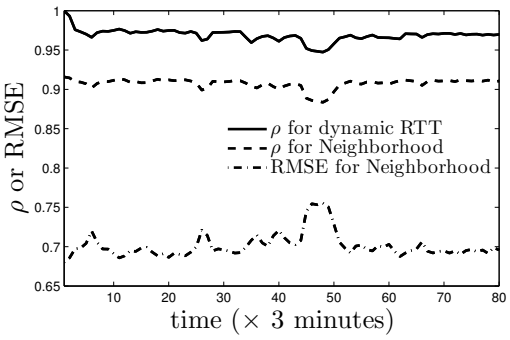


Figure 2.52: Impact of latency dynamics on proximity inference. The curve of ρ for dynamic RTT represents the rank correlation over time between the RTTs in the beginning of the simulation and the updated RTTs. The curve of ρ for Neighborhood represents the rank correlation between the non-updated proximity measures and the true, updated RTTs, and that of RMSE for Neighborhood represents the RMSE between the ratings of the non-updated proximity measures and of the true, updated RTTs

Yahoo and Facebook. To extract the IPs, we pinged from each PlanetLab node to www.google.com, www.yahoo.com and www.facebook.com. As these service providers direct user requests to nearby servers, the `ping` returned us the IPs of 147 Google, 11 Yahoo, 69 Facebook web servers. Unlike the DNS set the IPs in which are largely distinct from each other, the WEB set contains similar IPs such as 74.125.237.17 and 74.125.237.18. Thus, we clustered the IPs in the WEB set by their first 24 bits, resulted in 58 Google, 11 Yahoo and 31 Facebook IP clusters. In the experiments on the DNS set, the k landmarks are randomly selected from all 792 DNS servers, whereas on the WEB set, we first select randomly k IP clusters and each node then selects randomly an IP from each selected IP cluster.

The deployment of our service is simple and involves only the RTT measurement by `ping` from PlanetLab nodes to landmark nodes. For the purpose of evaluation, we also collected RTTs between PlanetLab nodes. During our experiment period between June 8th and June 12th, 2014, we were able to reach between 306 and 327 PlanetLab nodes. We confirm that the RTTs on PlanetLab are fairly stable, as shown in Sec. 2.9.2.3.

2.9.3.2 Comparison with DMFSGD and Vivaldi

As above, we compared our service based on neighborhood models with DMFSGD and Vivaldi, with $k = 32$ for all methods. For our service, we tested random landmark selection from the DNS and WEB set as well as from the PlanetLab nodes, called *Neighborhood DNS*, *Neighborhood WEB* and *Neighborhood* respectively. For each method, 10 runs of random landmark/neighbor selection were carried out using measurements collected at some time in our experiment period. We calculated the means and the standard deviations of the ranking correlation and the RMSE by comparing the results of each method with the true RTTs, shown in Table 2.10. Note that the experiments were repeated using measurements at different times and the results were found consistent due to the stability of the RTTs on PlanetLab. It can be seen that, while worse than DMFSGD and Vivaldi, our service still achieved decent accuracies on ranking and rating of network proximity. It is worth noting that the accurate RTT prediction by DMFSGD and Vivaldi comes at the cost of 10 to 20 message exchanges between each pair of neighboring nodes. In contrast, our service is the most lightweight,

Table 2.10: Comparison of ranking and rating accuracy on PlanetLab

ρ	mean	std
Neighborhood DNS	0.894	0.027
Neighborhood WEB	0.886	0.029
Neighborhood	0.903	0.021
DMFSGD	0.936	0.013
Vivaldi	0.939	0.014

RMSE	mean	std
Neighborhood DNS	0.709	0.034
Neighborhood WEB	0.722	0.038
Neighborhood	0.701	0.044
DMFSGD	0.622	0.020
Vivaldi	0.641	0.022

with each node probing a few pre-selected landmarks only once in the beginning of the service and with no computation required.

Note that the small standard deviations in Table 2.10 show the insensitivity of our service to random landmark selection when deployed on real networks such as PlanetLab. Thus, our experiments in this and previous section suggest that empirically, random landmark selection is sufficient as long as the landmark nodes are well distributed all over the world. In such situations, each latency measurement from a node to a landmark provides useful information about the location of the node.

2.9.4 Conclusion

This section presents a lightweight network proximity service that labels nodes with similar proximity to landmark nodes as being close to each other. The service allows the ranking and rating of network proximity which can be exploited for peer and server selection in P2P and CDN applications. Comparing to state-of-the-art latency prediction approaches such as DMFSGD and Vivaldi, the biggest advantage of our service is its simplicity. The only overhead in our service is a small number of `ping` measurements, and there is neither computation nor message exchange between network nodes required. Extensive simulations and real deployments show that our service can achieve accurate and stable proximity inference with a few randomly selected landmarks. Unlike many other landmark-based systems such as GNP and IDES, the landmarks in our service can be any Internet servers over which we have no experimental control.

2.10 Topology

2.10.1 MPLS Tunnel Diversity

One of the cornerstones of the Internet is the way data is forwarded through routing paths. Up to now, most of the IP flows have been treated the same way whatever their Quality of Service needs, their destinations, or their origins. This absence of privileges and flow distinction is called *best effort routing* or *Internet neutrality*. One of the tools allowing operators for easily differentiating classes of service and, so, performing *Traffic Engineering* (TE) mechanisms is *Multiprotocol Label Switching* (MPLS [86]). Historically, MPLS has been designed to reduce the time required to make forwarding decision thanks to the insertion of *labels* before the IP header. Nowadays, it is commonly believed that MPLS is mainly used for providing additional virtual private networks (VPN) services [74] and TE capabilities [95, 105, 108]. Few studies have focused on MPLS, studying essentially its deployment level [25, 92, 93] or its impact on topology discovery tools [34]. However, to

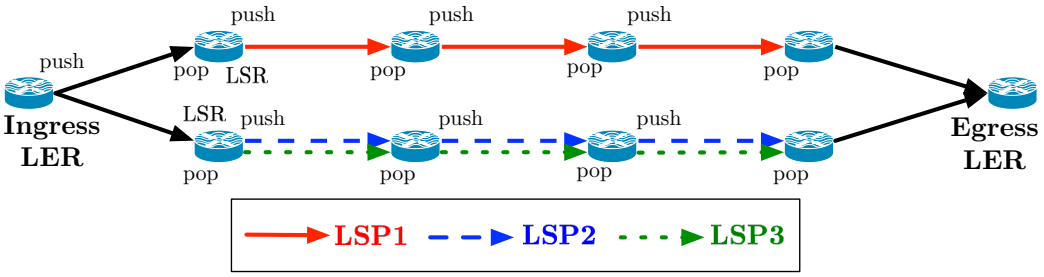


Figure 2.53: General overview of an MPLS tunnel

the best of our knowledge, none of them has focused on its actual usage in today's Internet.

Based on an extensive measurement campaign, we notice that, in a large proportion of the cases, a pair <ingress; egress> of an MPLS tunnel is made of multiple MPLS branches, both in terms of labels and IP level paths. However, without using additional probing, such as Paris Traceroute [7] in MDA mode [8], it is not straightforward to distinguish equal-cost multiple (ECMP) load balanced paths from multiple TE classes usage.

On one hand, ECMP load-balanced paths within MPLS tunnels correspond to the standard usage of the *Label Distribution Protocol* (LDP) [6] on top of IGP in order to enable inter-domain routing stability and extensibility (and preserving ECMP features of the underlying IGP, if any). On the other hand, allowing TE for MPLS refers to the use of *Resource Reservation Protocol TE* (RSVP-TE) [9] or VPN to enable service differentiation through the use of multiple *forwarding equivalent classes* (FEC - i.e., a set of packets a single router forwards to the same next hop, via the same interface with the same treatment).

In order to make this distinction possible at a lowest overhead possible for the network, we propose a passive method, the *Label Pattern Recognition* (LPR) algorithm, to differentiate standard IP equal cost paths (that LDP allows) from tunnels built with RSVP for actual TE purpose. From the analysis we performed on our dataset, we conclude that the usage of LDP is clearly the rule, while the use of RSVP-TE or VPN remains mainly the exception for transit traffic. To refine our study, we also apply LPR on a per AS basis to identify distinct TE behavior across the set of ASes we collected.

2.10.1.1 MPLS Background

The *Multiprotocol Label Switching* (MPLS) [86] was originally introduced to speed up the forwarding process. In practice, this was done with one or more 32 bits *label stack entries* (LSE) inserted between the frame header (Data-link layer) and the IP packet (Network layer). MPLS routers, i.e., *Label Switching Routers* (LSRs), exchange labelled packets over *Label Switched Paths* (LSPs). Each packet contains a stack of LSE, each being made of a 20 bits label, a Time-To-Live field and a type-of-service field.

The first MPLS router (*Ingress Label Edge Router* – Ingress LER – router, at the tunnel entry) adds the label stack, while the last MPLS router (*Egress Label Edge Router* – Egress LER – router, at the tunnel exit) removes the label stack. In most cases, the label stack may be removed by the penultimate MPLS router (*penultimate hop popping*, PHP). In that case, the tunnel exit is one hop before the Egress. Fig. 2.53 illustrates the main concepts associated to MPLS tunnels.

In an MPLS network, packets are forwarded on the basis of their label based on an exact match lookup instead of a longest prefix match lookup on the IP destination. At each MPLS hop, the label of the incoming packet is replaced by the corresponding outgoing label found in the switching table.

MPLS has two main usages: (i) a basic encapsulation technique allowing to transparently transmit packets through an MPLS domain using best effort IP routes computed by an IGP, and (ii) a TE tool allowing to better control routing and resources used by some flows. Here, we consider only point-to-point usage since, with `traceroute`, we measure only point-to-point routes (see Sec. 2.10.1.2.1 for details about our measurements).

The basic encapsulation method is used in two common scenarios. First, in the BGP transit scenario, a transit network using BGP as an inter-domain routing protocol and an IGP (e.g., IS-IS or OSPF) as an intra-domain routing protocol may use MPLS tunnels between its border routers to transparently carry packets between them. This way intermediate routers do not need to know about external destinations, only the incoming border routers need to know the outgoing one (by the BGP decision process, it is the BGP next hop) and the corresponding LSP. Another similar usage is for basic BGP MPLS VPN (Virtual Private Networks [85]). Again LSPs are constructed between the provider equipment (PE) of the VPN, and packets pertaining to a VPN may transparently cross the MPLS domain.

For this basic encapsulation method, labels are allocated through the *Label Distribution Protocol* (LDP) [6]. A router announces to its MPLS neighbors the association between a prefix in its routing table and a label it has chosen. Therefore, labels are allocated from downstream and, for a given prefix, a router advertises the same label to all its neighbors. Depending on the implementation, LDP may advertise a label for all prefixes in its IGP routing table (default case for Cisco routers) or only for loopback addresses (default case for Juniper routers). For transit traffic, LSPs are constructed by LDP towards loopback addresses of the exit router. The IP route followed by the LSP is the best effort IP route(s) computed by the IGP. If load balancing is not used, there is only one route between two endpoints (for instance, only LSP1 in Fig. 2.53 between the Ingress and the Egress LERs). On the other hand, if load balancing is used there may be several routes (usually with equal cost: ECMP Equal Cost Multipath – for instance LSP1 and LSP2 on Fig. 2.53 between the Ingress and the Egress LERs). Note that LDP builds an LSP-tree towards the destination prefix. Note also that, while the prefix used to build this tree may be very specific (i.e., a single IP loopback address), the *Forwarding Equivalence Class* (FEC – i.e., a set of packets a single router forwards to the same next hop, via the same interface with the same treatment) may be very large, i.e., all traffic exiting a Tier1 AS from the same border router.

Another quite different usage of MPLS is TE, where the goal is to tune the routes used by some flows either to give them some requested quality of service, or to optimize the network usage. In this case, it is expected that different flows entering the MPLS domain at the same Ingress LER and leaving at the same Egress LER may use different routes (for instance, LSP1 and LSP2 on Fig. 2.53). In this case, an IGP adapted for TE [52, 60] is in charge of computing routes satisfying the TE constraints, while the *Resource Reservation Protocol TE* (RSVP-TE) [9] is the signaling protocol in charge of reserving resources and allocating labels along the route. Note that it is expected that several LSPs can be build between the same pair of MPLS routers. Their label sequences are completely different while their IP path may or may not be distinct (for instance, LSP2 and LSP3).

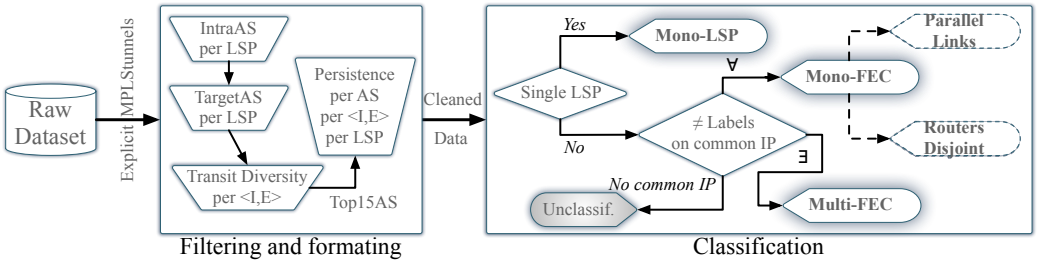


Figure 2.54: Overview of the LPR algorithm

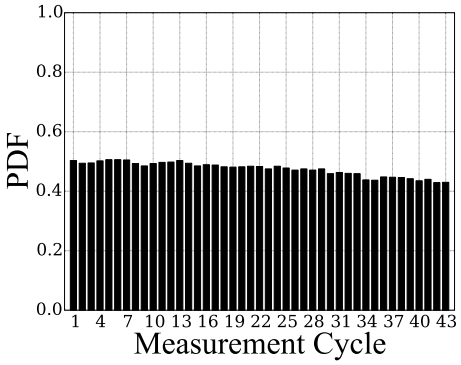


Figure 2.55: Proportion of traceroute, per measurement cycle, traversing at least one MPLS tunnel

2.10.1.2 Label Pattern Recognition Algorithm

Our main objective is to reveal the use of actual TE practice. More formally, for each <Ingress LER, Egress LER> pair, we aim at distinguishing the use of multi-FEC from standard IP load balancing with ECMP. Instead of running additional extensive active probing such as using Paris Traceroute with the MDA algorithm [8], we target a passive classification method. Eventually and in the same fashion, with MPLS labels, we are able to transparently perform some alias resolution and so distinguish router disjoint sub-paths from parallel links.

To this end, we develop the *Label Pattern Recognition* (LPR) algorithm, that analyses the MPLS labels on IP addresses that are common to several LSPs (or that are likely to belong to the same router). Our algorithm classifies each couple <Ingress LER, Egress LER> into three main classes according to the recognition of the standard behaviors of RSVP-TE versus LDP in terms of label distribution. All the processing of LPR are done off-line. The whole process is illustrated at Fig. 2.54. Sec. 2.10.1.2.1 explains how we performed measurement in order to build our raw dataset. Next, LPR works in two stages: first, it filters the dataset (Sec. 2.10.1.2.2) before performing the classification itself (Sec. 2.10.1.2.3).

2.10.1.2.1 Data Collection Methodology In order to collect data for evaluating transit tunnels diversity, we deploy scamper [63] on the PlanetLab network. We configure scamper so that it works in Paris Traceroute [7] mode with ICMP-echo packets. As targets for our measurements campaign, we randomly pick up 10^6 IP addresses from the Archipelago [20] destination list. We evenly divided

ASN	3356	7018	2914	6453	6461	286	6830	15412	1273	7843	2828	4436	10026	292	7473
#MPLS IPs	647	294	252	242	180	117	111	120	110	103	92	82	78	201	65
#IPs	5,039	7,700	1,712	1240	1,006	377	2150	311	504	386	1488	425	466	10,424	472
Ratio MPLS	0.128	0.038	0.147	0.195	0.179	0.310	0.052	0.386	0.218	0.267	0.062	0.193	0.167	0.019	0.138
#<I,E>	9,577	1,032	1,393	922	634	260	59	118	258	272	503	141	241	107	111
#LSPs	132,607	31,569	22,425	17,475	18,059	3,018	290	2,342	4,434	7,485	5,237	722	9,113	4,559	965

Table 2.11: Statistics about the Top 15 ASes after some filtering

this list between 200 PlanetLab monitors, scattered in North America, Europe, Asia, and Oceania.

We performed several measurements cycles. For each cycle (one per week), we kept the same destinations list and the same PlanetLab monitors[7]. Measurements cycles started on August 23rd, 2013 and stopped on June 13th, 2014. Each measurement cycle was run on Friday, noon (Belgium time). This leads to a dataset made of 43 snapshots from our set of 200 PlanetLab vantage points. In particular, it measures explicit MPLS tunnels, the ones that show IP tagged with MPLS labels in their return answer [25]. Moreover, we perform an IP to AS mapping using the BGP routing data available from the Route Views project (`http://www.routeviews.org/`).

We first measure the proportion of `traceroute` that traverses at least one MPLS tunnel in each of the 43 measurement cycles. We observe that this proportion is roughly stable and really important, i.e. 45% of the paths (see Fig. 2.55), although we notice a slight decrease over time of the path proportion encountering at least one MPLS tunnel. On a per unique IP grain, we measure that 7% of IP addresses can be directly tagged as MPLS interfaces. Compared to previous works [25, 93], MPLS seems now more present than in the past. In this study, we focus on IP paths with MPLS labels because we need to interpret their similarity to classify their use into multi- or mono-FEC.

2.10.1.2.2 Filters In our study, we do not consider inter-domain MPLS tunnels as its usage seems really negligible. This means that IP addresses involved in a tunnel must belong to the same AS. This first filtering step is done by the `IntraAS` filter (see Fig. 2.54). It removes 0.5% of LSP in the raw dataset.

If the destination of the traceroute belongs to the same AS than the MPLS tunnel, we are in a case where the tunnel is not used for transit traffic (and so is unlikely to be used for TE purpose). We therefore remove, from our dataset, all MPLS tunnels that do not fall within this transit traffic scenario. This is the objective of the `TargetAS`. It removes ≈10% of LSP in the raw dataset.

To have a better view of the routing diversity, we next want to keep only <Ingress LER, Egress LER> pairs that are used to reach at least two destinations belonging to different ASes (`Transit Diversity` filter). The idea here is to capture multi-FEC scenario based on IP destination prefixes that, by definition of IP routing, represent the more usual practice of TE. It removes up to ≈15% of LSP in the raw dataset. Note that, even with that filter, we may underestimate the transit tunnel diversity.

After those three filters, our dataset contains a set of MPLS tunnels identified by a <Ingress LER, Egress LER> pair, both LERs being connected through one or several LSPs. In order to limit the study to relevant cases, we select the ASes (and tunnels belonging to those ASes) having the largest set of IP addresses flagged as MPLS (in the `traceroute` output). We limit those ASes to the Top 15, representing almost 75% of the LSPs in the cleaned dataset. Table 2.11 presents various statistics related to this Top 15 ASes after the first three filters. It shows, for each AS, the number IP addresses

[7] It is worth to notice that, in case a given PlanetLab monitor is down for a cycle, a new PlanetLab monitor is selected to replace it.

Algorithm 4 LPR algorithm

```
/* Data cleaning and formatting */
for l in allLSP () do
    cLSP ← Apply intra-AS filter
    cLSP ← Apply target-AS filter
for c in coupleIE (cLSP) do
    ccoupleIE ← Apply AS-diversity filter
top20AS ← AS_top20 (ccoupleIE)
for as in top20AS do
    for cc in as_ccoupleIE do
        for l in as_ccoupleIE_LSP do
            ctop20AS ← Apply persitent filter
/* Main Classification Step */
for as in ctop20AS do
    for cc in as_ccoupleIE do
        if #Diff_LSP(cc) = 1 then /*→ mono LSP: mono FEC, no LB (class 1)*/
            ⇒ move cc in pattern class 1 for as and continue
        else /* → multi LSP */
            m, n, k, p ← 0
            Lip ← Common_IP_set(cc)  // no class 4 if extended
            for ip in Lip do
                p ← p + #Diff_Labels(ip)
                if #Diff_Labels(ip) > 1 then
                    n ← n + #Diff_Labels(ip)
                    if #Diff_Labels(ip) > #Diff_Labels(∀pred(ip)) then
                        k ← 1
                else if #Diff_Labels(ip) = 1 or #Diff_Labels(∀pred(ip)) = 1 then
                    m ← m + 1
            if n > 0 then /* → multi FEC (class 2) */
                r = 0
                if #Diff_IP_LSP(cc) = 1 then
                    r ← 1 + n/p
                else
                    r ← n/p
                if r = 1 then /* → multi IP LSP, no LB */
                    ⇒ move cc in pattern class 2 for as
                else if r = 2 then /* → mono IP LSP, no LB */
                    ⇒ move cc in pattern class 2.1 for as
                else if r < 1 & k = 0 then /* → multi IP LSP and LB */
                    ⇒ move cc in pattern class 2.2 for as
                else if r < 1 & k = 1 then /* → inconsistent multi IP LSP */
                    ⇒ move cc in pattern class 2.3 for as
                else /* → inconsistent mono IP LSP */
                    ⇒ move cc in pattern class 2.4 for as
            else if m = |Lip| then /* → mono FEC and LB (class 3) */
                if #Diff_Label_LSP(cc) > 1 then /* → LB multi-routers */
                    ⇒ move cc in pattern class 3 for as
                else /* → LB multi-links */
                    ⇒ move cc in pattern class 3.1 for as
            else /* → not enough info to conclude... (class 4) */
                ⇒ move cc in pattern class 4 for as
```

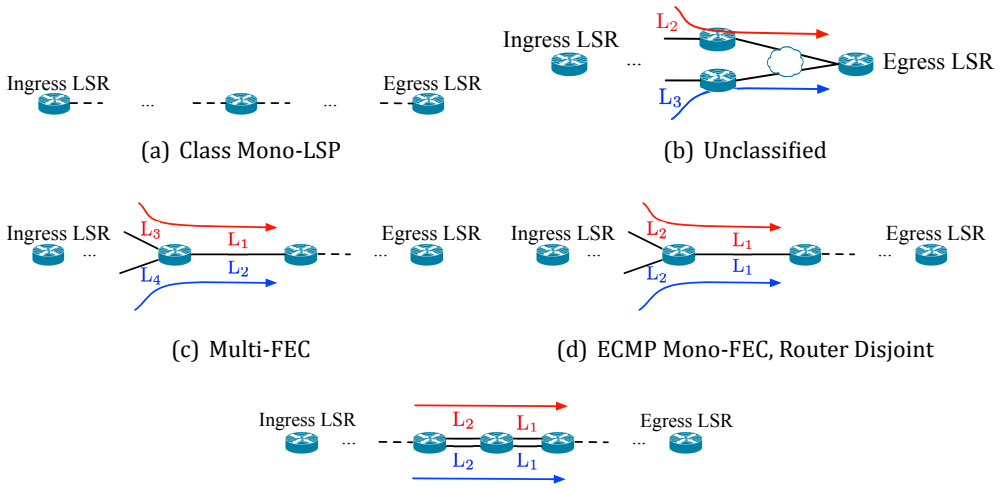


Figure 2.56: Typical MPLS Label Based Patterns

tagged as MPLS ("#MPLS IPs"), the number of IP addresses collected ("#IPs"), as well as the ratio between both. It also provides the number of <Ingress LER, Egress LER> pairs and LSPs. Note that our top 15 ranking has been computed on several snapshots while Table 2 reports values extracted from a single snapshot.

Finally, we verify the persistence in time of the LSPs to remove noise due to routing changes. We keep LSPs encountered in measurement cycle X only if they are also seen in measurement cycle $X+1$ or $X+2$. Otherwise, they are removed from our set (`Persistence` filter). This filter removes about 20% of LSPs in the raw dataset. Note that we keep track of dynamics as it can represent in itself a TE usage rather than routing changes. In practice, if 100% of LSP disappear for a given AS, we reinject the whole set of its LSP to perform a static classification on them. Moreover, since this filter may bias our analysis, we perform it with and without this filter to understand its implication on the classification.

The resulting set of MPLS tunnels can then be statically classified. Those subsets of LSPs are robust, i.e., they are persistent in time, possibly diverse in targets, and focus on transit traffic within a single ISP. At this stage, the dataset is now structured, i.e., we can sort each remaining LSP within its <Ingress LER, Egress LER> pair and its AS. In the following, for each LSP falling within the same AS and the same couple of border edge routers <Ingress LER, Egress LER>, we compare their content both in terms of IP addresses and labels.

2.10.1.2.3 Classification The first class, not illustrated in Fig. 2.56, is called *Mono-LSP*. That is, for a given <Ingress LER, Egress LER> pair, there exists only a single LSP (same IP addresses and same labels) for different destination ASes (thanks to the filtering process, we know that remaining pairs concern at least two ASes). This means we do not observe transit tunnel diversity, the same LSP being always used whatever the destination. As a consequence, for this subset of LSPs, there is neither ECMP load balancing (by definition of the class) nor several FECs used to reach different ASes with different constraints.

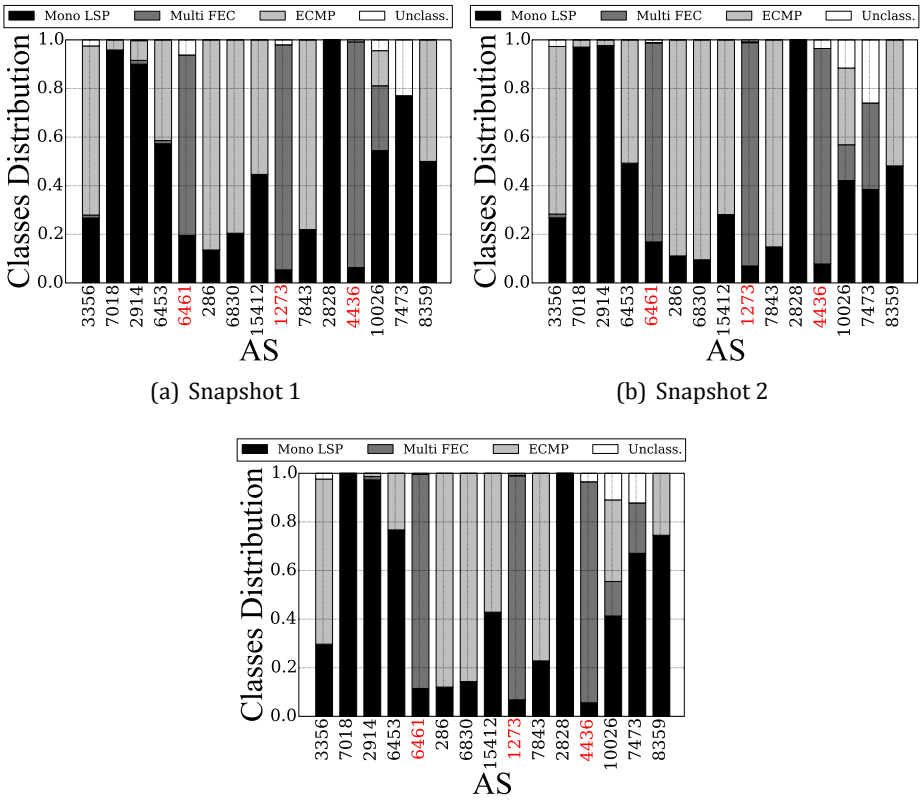
The second class, illustrated in Fig. 2.56(c), is called *Multi-FEC*. That is, for a given <Ingress LER,

Egress LER> pair, there exists at least one IP convergence point, i.e., a common router where at least two LSPs converge, where we observe multiple LSPs using different labels. Indeed, on Fig. 2.56(c), the first LSP (in red) considers labels (L_3, L_1), while the second (in blue) considers labels (L_4, L_2). Labels L_1 and L_2 being used on the same IP address, thus on the same router, this case suggests the use of multiple FEC for that edge routers pair. On the contrary to the use of standard LDP where, by default, labels have a router scope (that is, a given router proposes the same label for a given destination to all its upstream routers – destinations being here loopback of BGP edges routers for scalability) and so do not distinguish FEC for a given edge router. Here each LSP corresponds to a distinct FEC since the labels are distinct. As there are multiple LSPs (that differ in term of label or IP) between the Ingress LER and the Egress LER, this use of multi FEC suggests TE practice.

The last interesting class, shown in Fig. 2.56(d) and Fig. 2.56(e), depicts several distinct LSPs that differ on at least one IP address for a given <Ingress LER, Egress LER> pair. More specifically, this class implies that for all LSPs traversing a common router, labels are identical (on contrary to the previous class where there exists at least one difference). The subset of LSPs belonging to a given <Ingress LER, Egress LER> pair may thus only differ in terms of IP addresses on common router (while being distinct in terms of both IP addresses and labels on non common routers). Therefore, this class refers to the simple use of a single FEC between the Ingress and the Egress LER. Various LSPs being present at the IP level, it typically corresponds to the use of ECMP load balancing. This class that we call *ECMP Mono-FEC* (or *Mono-FEC* on Fig. 2.54 for readability reasons) may be divided into two subclasses: if labels are the same along all the LSPs, we can use this information as a simple alias resolution technique to state that IP addresses sharing those labels belong to the same router. Then, when it is the case, we conclude that the <Ingress LER, Egress LER> pair only uses parallel links to perform ECMP load balancing (Fig. 2.56(e)), while we can state that there is at least one router disjoint LSP otherwise (Fig. 2.56(e)). This distinction is of the highest interest: we do not require the use of an active resolution probing to show that a large share of ECMP load balanced paths relies actually only on parallel links.

It is worth to notice that the Egress LER generally does not exhibit labels due to the use of PHP (i.e., the label stack is removed by the penultimate MPLS router). In such a case, LSPs of a given <Ingress LER, Egress LER> pair may never intersect each other on a common IP address providing labels (the Egress LER is a convergence point by definition while other intermediate LER are not necessarily traversed by multiple LSP). If we are unable to classify such a pair of edges routers because there does not exist common IP in the set of LSPs, we arbitrarily tag it as *unclassified*. To avoid such a limitation (no more unclassified pairs) and also improve our classification in general, we can rely on a simple passive alias resolution mechanism: in a standard situation, on a `traceroute` path, all previous IP level interfaces of a given common IP address should belong to the same router. Indeed, if we assume that a router answers to `traceroute` probes using the incoming interface of the probe and there is no layer-two device connected to this interface (as illustrated in Fig. 2.56(b)), then such IP addresses are aliases of the same router (since to enter a router through the same IP, it is necessary to use the same link on the same upstream router). In the case of mono-FEC, we should thus observe the same label on previous IP addresses while we should observe distinct ones in case of multi-FEC case (as usual). This simple mechanism reenforces the analysis by augmenting the set of common routers as illustrated in Fig. 2.56 at the price of provoking some false positives (we evaluate both approaches in the next section).

Eventually, we also divide the multi-FEC class into three subclasses (although this is not shown on Fig. 2.54 and that we do not provide any results due to space limitations): *without ECMP*, *with ECMP*, and *inconsistent*. The first two cases refer to the possibility of having both ECMP and multi-FEC (LDP and RSVP-TE can be both enabled), while the last case reveals cases where the label distribution



(a) Snapshot 1

(b) Snapshot 2

(c) Snapshot 3

Figure 2.57: MPLS tunnels classification for three measurement snapshots with respect to the top 15 ASes

does not map to a usage we expect (no conservation of labels diversity between subsequent IP hops). On the contrary to other classes, the multi-FEC one may track several combinations of MPLS behaviors because the LPR algorithm is asymmetric, i.e., an <Ingress LER, Egress LER> pair falls in this class if $\exists$ one common router with distinct labels while other classes are more restrictive. They require a $\forall$ check on the whole common router set. Algorithm 4 presents a formal version of LPR.

2.10.1.3 Tunnel Diversity and TE Analysis

In this section, we use the LPR output (i.e., the tunnel classification) to understand how and whether IP network operators perform TE with MPLS. Fig. 2.57 provides an overview of most popular MPLS usages revealed with the LPR algorithm on three snapshots dated the beginning, the middle, and the end of our temporal dataset. We represent here only the results for the top 15 ASes (see filters, Sec. 2.10.1.2.2).

First, we notice that only three ISPs (ASN 6461, 1273, and 4436, i.e., the ones tagged in red in the xticks labels of the figure) exhibit an highly dynamic LSP profile: all their LSPs change from one snapshot to the next. Note that we correlate this analysis with specific measurements and we observe that such changes are actually on the order of five minutes and seem to be associated to the brand of underlying routers (e.g., with Juniper routing devices). Moreover, apart from this dynamic

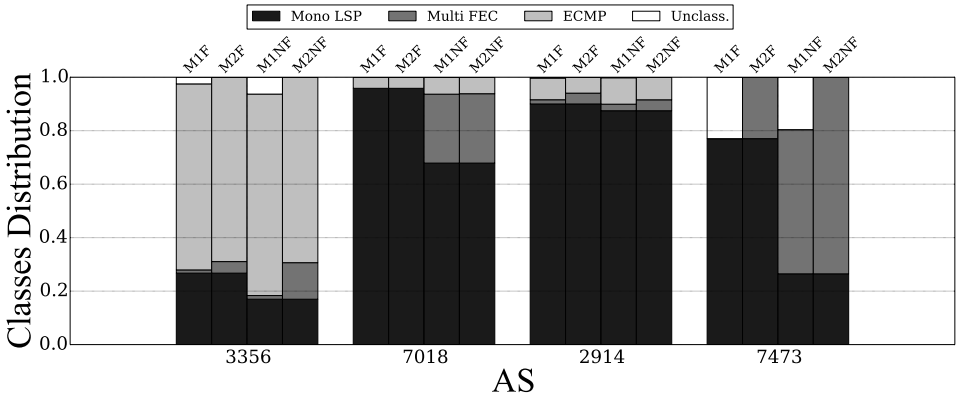


Figure 2.58: Impact of the processing mode and filtering on the classification (4 worst cases)

and fast evolution, one may observe that most <Ingress LER, Egress LER> pairs of such ASes fall into the multi-FEC category. Definitively, those three ASes perform some kind of dynamic TE with RSVP-TE.

Second, except on those four ASes, we do not observe a large use of multi-FEC for most <Ingress LER, Egress LER> pairs. AS10026 seems to be the only exception in non-highly dynamic ASes. The deployment and the use of multi-FEC TE seem limited to specific operators while most of them (ASN 3356, 6453, 286, 6830, 15412, and 8359 in particular) seem to use ECMP load balancing at a large scale. Only AS2828 exhibits mono-LSP for all <Ingress LER, Egress LER> pairs we collect. We also observe that this high ECMP deployment actually mainly relies on parallel links (detailed results are not given here due to space limitations). For most ASes, more than half of load balanced paths between MPLS edge pairs are actually only made of parallels links (as shown on Fig. 2.56(e)).

Third, we can observe that results are quite similar in time (at least on the three snapshot we selected) meaning that we do not observe significant MPLS changes in use cases across our temporal dataset. The only changes we collect concern too few data to be really conclusive (on AS7473). In summary, TE usage seems limited to specific operators and it does not evolve quickly over time.

To understand the potential bias and so the error margin resulting from our filtering process and due to the way we compute the common router set, we select the four ASes where classification results differ the most (the remaining eleven ASes are almost completely insensitive to our classification calibration). Fig. 2.58 shows the four possible combinations of our two features: with our without the temporal filtering ("F" and "NF" respectively on Fig. 2.58), and with or without the common router extension discussed in Sec. 2.56 ("M1" and "M2" respectively on Fig. 2.58). We can observe that most changes seem to concern the prevalence of the multi-FEC class that is more represented than before (in Fig. 2.57, we perform the temporal filtering and do not extend the common router set). This side effect is natural by design since, as stated before, dynamics is correlated to the multi-FEC usage and extending the common router set increases the quantity of comparable labels and so cases where multi-FEC can be visible (recall that one proof of existence is sufficient for this class). Therefore, this figure allows for providing an interval that describe the lower bound and the upper bound of multi-FEC prevalence. In all situations except for AS7473 (and 7018 in a much lower extent), such intervals are really reasonable meaning that our classification is fortunately not too sensitive to its calibration. On AS7473 (the penultimate smallest network and the only one that evolves in our top 15), one can deduce that the extension of the common router set allows for classifying all unclassified edge pairs into the multi-FEC class and that the temporal filtering should be carefully used to classify ASes enabling partial dynamic TE.

2.10.2 Middleboxes Taxonomy

Nowadays, the standard and well-known description of the TCP/IP architecture (i.e., the end-to-end principle) is not anymore applicable in a wide range of network situations. Indeed, enterprise networks, WiFi hotspots, and cellular networks usually see the presence of *middleboxes* being part of the network architecture in addition to traditional network hardware [91].

Recent papers have shed the light on the deployment of those middleboxes. For instance, Sherry et al. [91] obtained configurations from 57 enterprise networks and revealed that they can contain as many middleboxes as routers. Wang et al. [106] surveyed 107 cellular networks and found that 82 of them used NATs. D'Acunto et al. [23] analyzed P2P applications and found that 88% of the participants in the studied P2P network where behind NATs. Further, it has been shown that middleboxes have a negative impact on the TCP protocol (and its extensions) evolution [46, 47].

There is a wide range of middleboxes, going from "simple" NAT to complex system that can potentially modify the whole packet header. There is thus a wide variety of middleboxes, as the wildlife and plant life are diversified. However, on the contrary to the biology, there is no rigorous behavioral taxonomy of the middleboxes, classifying them according to their effects on packets, on traffic, or on the network quality experienced by users. Furthermore, as middleboxes increasingly impact the Internet, protocol designers have to cope with a middlebox-full Internet. Each mechanism has to be certified as middlebox-proof [35, 46]. For those researchers, a summary of the potential middlebox network interferences would be a valuable asset.

This is exactly what we want to tackle here. In this section, we propose a path-impairment oriented middlebox policy taxonomy, that aims at categorizing the initial purpose of a middlebox policy as well as its potential unexpected complications. We choose to classify middlebox *policies* (i.e., "Definite methods of action to guide and determine present and future decisions" [5]) rather than middleboxes themselves because the latter often combine multiple policies. Our taxonomy is based on three main policies: *Action* (i.e., the fate of a packet crossing a middlebox implementing this policy), *Function* (i.e., the policy purpose), and *Complication* (i.e., the potential path connectivity deterioration).

Based on a `tracebox` [24] (a `traceroute` extension that is able to reveal the presence of middleboxes along a path) measurement campaign on IPv4 and IPv6 networks, we confront our taxonomy to the reality on the ground. We establish a large-scale dual-stack survey of middleboxes that implement rewrite, drop, and proxy policies that may harm the performance of regular traffic and affect protocol deployability. Then, we discuss our results in light of our taxonomy.

2.10.2.1 Taxonomy

In this section, we propose a path-impairment oriented middlebox policy taxonomy, that aims at categorizing the initial purpose of a middlebox policy as well as its potential unexpected complications. We chose to classify middlebox policies (i.e., "A definite method of action to guide and determine present and future decisions" [5]) rather than middleboxes themselves because the latter often combine multiple policies. Our taxonomy focuses on packet mangling middleboxes aspects, the initial purpose of such an action and the network interferences that may involuntarily result.

We describe each policy implemented in a single (not multi-hop) middlebox in three ways, each including several taxa (i.e., categories); (i) *Action*, the fate of a packet crossing a middlebox that implements this policy; (ii) *Function*, what the policy is intending to achieve, its purpose; (iii)

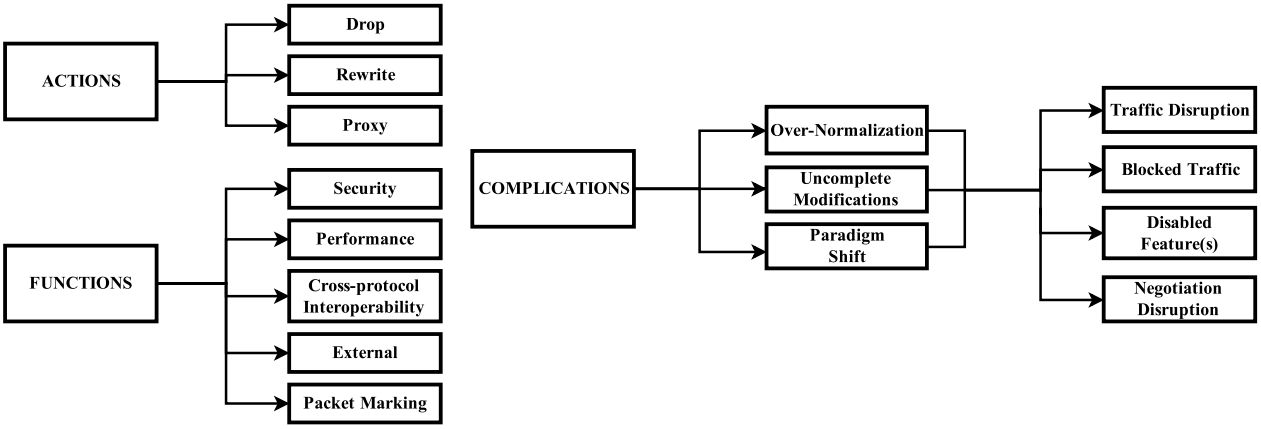


Figure 2.59: A path-impairment oriented middlebox policy taxonomy

Complication, the possible resulting path connectivity deterioration. Fig. 2.59 illustrates our path-impairment oriented middleboxes policy taxonomy. Each middlebox policy has to fall in *at least* one taxon for each of the three angles in order to be consistent with the taxonomy. Furthermore, each category essentially describes path characteristics, deliberately avoiding a larger focus.

A middlebox policy can be classified differently if we consider the policy in itself or the functional entity it is part of. For instance, the purpose of a tunnel endpoint policy is to provide cross-protocol interoperability but, if we consider the whole tunnel, it may be security (e.g, VPNs), traffic engineering (e.g., MPLS), or another. As we aim at characterizing middlebox-related network interferences rather than establishing an exhaustive middlebox taxonomy [17], we focus on single-hop modifications and ignore composition aspects. Moreover, we intentionally leave aside the middlebox state-related features (e.g., statefull, stateless), robustness to failure, and its implicit or explicit nature.

While examining the possible complications involved by middlebox policies, we deliberately narrow our horizon to performance worsening and feature's inability of use, omitting the multiple and various reported security flaws created by middlebox policy implementations [51, 82, 83].

Fig. 2.59 displays the taxonomy and shows its three facets: *Actions*, *Functions*, and *Complications*. These families are described in the subsequent sections.

2.10.2.1.1 Actions The *Action* category describes the actual action of a middlebox on a matched packet, defined by middlebox policies. We consider three basic kinds of middleboxes policy action: *Drop*, *Rewrite*, and *Proxy*. This aspect is decisive because middlebox policies that applying different actions will more likely cause different types of network dysfunctions.

Drop policies are common features whose purposes vary from security to performance optimization concerns. Depending on how both ends react to this type of failure, the outcome may also vary from minor traffic disruptions, such as bandwidth reduction or the inability to use specific TCP options, to the inability to establish a TCP connection. An example of an ordinary middlebox policy that may apply dropping policies are TCP sequence number sanity checks for dropping out-of-window packets and invalid ACKs [46, 47].

Rewrite policies are also common among middleboxes. Their initial purposes are covering the entire Sec. 2.10.2.1.2. As they break the TCP/IP end-to-end principles [59], they may cause various

problems to protocol end-to-end functions which assume unmodified layers over the third. An aging but still relevant example is the TCP Initial Sequence Number (ISN) re-randomization policy which aimed at mitigating ISN prediction attacks [24], but which may fail to ensure consistencies with TCP absolute sequence related numbers other than the TCP sequence number and the TCP acknowledgment number (e.g., Selective Acknowledgement) and cause traffic disruption [46, 70].

Proxy policies middleboxes are relay agents between clients and servers of an application. They vary from the rewritten policies middleboxes by not simply rewriting the forwarded packets but rather by receiving client data from a connection, then establishing a second connection to send data to the server and vice versa.

2.10.2.1.2 Functions The *Function* category describes the purpose of a middlebox policy (e.g., what it is intending to achieve). As we already noticed earlier in this section, a policy can be classified differently if we consider the policy in itself or the middlebox set it is part of (e.g., a tunnel endpoint with respect to the whole tunnel). We consider five kinds of Functions middleboxes: *Security*, *Performance*, *Cross-Protocol Interoperability*, *External*, and *Packet Marking*.

Security-motivated middlebox policies are implemented in Security-oriented middleboxes, dedicated hardware deployed in enterprise, and home network for improving network security. They may serve purposes like providing an authentication mechanism (e.g., ALGs), defending against attacks such as DDoS, attacks on IP (e.g., IP spoofing, fragmentation attacks) or attacks on TCP (e.g., sequence number related attacks, TCP reset attacks), providing access control, normalizing the TCP features to prevent the use of features considered unknown and/or unsafe or separating similar networks into different security zones.

It has been shown that such a type of middleboxes is becoming more and more popular [91]. This increasing popularity raises concerns regarding overly restrictive middlebox policies which may either block benign traffic matched as unsafe, causing blackholes, or apply modifications to transport and network header fields, precluding the use of protocol features or hampering their proper functioning, indirectly reducing networks performance and/or functionalities (e.g., forbidding ECN or MultiPathTCP). Security-oriented middleboxes involves IP and application firewalls, IDS/IPSes, certain Application-Level Gateways (ALGs) and NATs.

Performance-enhancing middlebox policies middleboxes goal is the improvement of the network performance regarding a link along the path, a connection between both path's ends or the middlebox itself. It may aim at improving pure effective bandwidth performance through transport layer engineering as well as solving box-relative performance issues.

This category of middlebox policies may apply semantically wrong legacy modifications which may cause traffic disruption and protocol inconsistencies. They might as well introduce TCP errors and unforeseen changes in TCP adaptive behavior, leading to various network interferences. TCP performance enhancing proxies, caches, and packet schedulers are examples of middleboxes implementing *Performance* policies.

Cross-protocol interoperability middlebox policies are performing protocol translation. They aim either at connecting dissimilar networks or at translating protocols over the network layer. The main problem with this kind of policies is to be consistent with every protocol versions and features. Indeed, the input protocol language may evolve, and middleboxes may then apply outdated translations to it or may fail to ensure completeness of its modifications. This can lead either to the inability to use certain features or to network performance degradation. Example of cross-protocol interoperability middleboxes policies are NAT 6to4, NAT-Protocol Translation (NAT-PT [102]), Stateless

IP/ICMP Translation (SIIT [76]), tunnels endpoints and transcoders.

Packet marking middleboxes classify packets according to policies and select or mark them for differentiated services (via IP DSCP's field). Their action is limited to network layer modifications. They are packet classifiers, or ECN-capable routers. Their policies are not likely to degrade network performance.

External middlebox policies have purposes that are external to the paths to which they belong (e.g., IPv4 address exhaustion) or not related to the Internet at all (e.g., economic purposes). External middlebox policies are heterogeneous and have to be analyzed independently.

2.10.2.1.3 Complications We describe here the potential *Complications* caused by middleboxes policies by examining them under two points of view: (i), their technical causes, which are directly related to their initial purposes and, (ii), the associated actions (respectively Sec. 2.10.2.1.2 and Sec. 2.10.2.1.1), and their unfortunate consequences, i.e., causes and consequences.

The *Causes* of the network interferences created by middleboxes aims at classifying the technical problems origin. It regroups manufacturers/policy designers fundamental errors or deliberated choices that, from a path-impairment perspective, lead to network interferences.

Over-normalization refers to a middlebox policy that limits protocol features and options, as a blacklist or whitelist filter, to a restricted subset of the protocol. The problem of this type of middlebox behavior constraining the design of new extensions have been addressed [47]. It may limit protocol performance as well by preventing the usage of the entire protocols capabilities, or simply by taking drop decisions. The Cisco ASA firewall family includes a TCP normalizer feature [36] that limits TCP to a subset of the protocol considered safer either by removing and rewriting bytes related to unwanted features or by dropping packets containing those features.

Incomplete modifications refers to middlebox policies that fail to ensure completeness of their modification(s). This type of network inconveniences is caused by middleboxes modifying a specific protocol field and not modifying semantically related fields, allowing translated/modified data alongside untranslated/unmodified data. They may fail to identify all related fields for legacy reasons or simply neglecting it for performance concerns (e.g., refusing to parse TCP options). Examples of such middlebox policies are TCP ISN shuffling boxes and NATs regarding respectively TCP SACK and FTP.

A *Paradigm shift* (2-way to n-way) happens when both ends running a protocol are assuming 2-way peering relationships. Middleboxes, by applying modifications *in the middle of the path*, are breaking the TCP/IP end-to-end principles, and thus, are causing both ends to undergo a paradigm shift *de facto* to n-way peering relationships [59]. As many mechanisms are not designed to handle this new paradigm, errors may occurs. When both ends are trying to share state related data or to negotiate capabilities this phenomenon may, in certain scenarios, put both ends in inconsistent states or, combined with an unfortunate load balancing, distort protocol negotiations [46]. This paradigm shift also raises security concerns. We know that NATs are limiting the use of IPSEC [4], but it also invalidate most transport layer security solutions. Moreover, the trust models and key distribution models have to be definitively rethought as n-way relationships [17].

The *Consequences* are the network complications final outcome, what both ends actually experience. We focus exclusively on path performance related issues, leaving aside security and processing performance considerations.

Traffic disruption policies produce unwanted consequences such as interferences with control data rendering it useless, bandwidth reductions or others path performance impairments.

Middlebox policies may cause *Blocked traffic* either explicitly (sending TCP RST packet) or implicitly (dropping packets). It may not be the final outcome of a connection. If a specific option/feature is blocked by a middlebox, the client could be configured to retry establishing the connection without the undesirable options/features, but if it is not, no connection at all is possible.

Middlebox policies may aims at preventing the use of features considered unknown and/or unsafe by modifying, nopping them or by preventing them from being negotiated. If it is achieve symmetrically, the consequences are limited to the inability to use the restricted features; It is a *Feature-disabling* policy. If the modifications are done asymmetrically and the negotiation is not resilient enough, the policy may fail to disable the feature and lead to inconsistent protocol states [46]. Policies resulting in the latter consequences are categorized as *Negotiation disruption* policies.

2.10.2.2 Middleboxes Detection

Algorithm 5 Proxy detection algorithm

```
1:  function DETECTPROXY(DESTINATION)
2:  tcp:
3:      probeTCP ← defineProbe(dst=destination, transport=TCP)
4:      resultTCP ← doTraceroute(probeTCP)
5:      lenTCP ← number of hops of resultTCP
6:      if lenTCP < 1 OR resultTCP is a timeout then
7:          return
8:  udp:
9:      probeUDP ← defineProbe(dst=destination, transport=UDP)
10:     resultUDP ← doTraceroute(probeUDP)
11:     lenUDP ← number of hops of resultUDP
12: result:
13:     if lenUDP > lenTCP then
14:         There is a proxy on the path to destination
15:         return
```

Algorithms 5 and 6 are respectively the proxy detection algorithm and the statefull box detection algorithm. The proxy detection algorithm performs a first `traceroute` round over TCP, a second over UDP and compares the number of hops. The statefull box detection algorithm is more complex and relies on multiple TCP `traceroute`-like probing with decreasing Initial Sequence Number values. It assumes that a statefull box will perform in-window sanity tests and expects specifically wrong packets to be dropped.

2.10.3 IGP Weight Inference

2.10.3.1 Inference Overview

Routing simulations, as well as theoretical Internet graph understanding [43], need ground truth data to provide realistic and significant results. However, lots of such studies lack of recent and reliable routing topologies. In worst cases, concerned authors only consider small toy topologies while, in best ones, they use data sources coming with strong limitations. For instance, for more

Algorithm 6 Statefull box detection algorithm

```
1:  function DETECTSTATEFULL(DESTINATION)
2:      seqNum ← random(0, 2^32 − 1)
3:  round1:
4:      probe1 ← defineProbe(dst=destination, transport=TCP,sequence_num=seqNum)
5:      result1 ← doTraceroute(probe1)
6:      server_TTL ← number of hops of result1
7:  round2:
8:      probe2 ← defineProbe(dst=destination, transport=TCP,sequence_num=seqNum - 10)
9:      result2 ← doTraceroute(probe2, MAX_TTL=server_TTL - 1)
10: round3:
11:     probe3 ← defineProbe(dst=destination, transport=TCP,sequence_num=seqNum - 20)
12:     result3 ← doTraceroute(probe3)
13:     len3 ← number of hops of result3
14:     if len3 = server_TTL then
15:         There are no statefull middlebox between you and the destination.
16:         return
17: round4:
18:     result4 ← doTraceroute(probe2)
19:     len4 ← number of hops of result4
20:     if len4 ≠ server_TTL then
21:         return
22: round5:
23:     result5 ← doTraceroute(probe3)
24:     len5 ← number of hops of result5
25:     if len5 ≠ server_TTL then
26:         return
27:     else
28:         There is a statefull middlebox between you and the destination (probably the len3-th hop).
29:         return
```

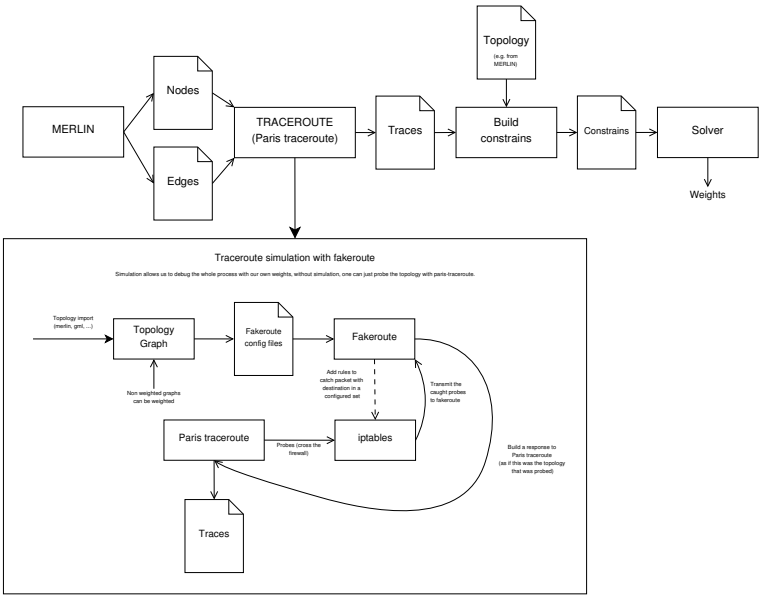


Figure 2.60: IGP weight inference whole system

than ten years now, routing authors make use of weighted ISP topologies [65] collected with Rocketfuel [94]. Based on traceroute data collected from several vantage points, Mahajan et al. build a linear constraint-based model to approximate IGP link weights. However, it is known that Rocketfuel suffers from several limitations (see, for instance, Teixeira et al. [99] and Coyle et al. [21]). Since the seminal paper by Mahajan et al., little efforts have been made in improving the quality of weighted ISP maps. Even the very recent Internet Topology Zoo does not contain any link weight information [55].

We basically rely on the MERLIN platform [67] for collecting topology information of the targeted AS. MERLIN is based on IGMP probing. It basically sends multicast management requests that are able to retrieve, within a single probe, all multicast interfaces and links of a targeted router. It is worth to notice that purely unicast information can also be revealed [71]. IGMP probing can natively discover multicast topologies at the router level with a low probing cost, avoiding so the use of any alias resolution techniques [53].

The MERLIN platform is centralized, i.e., each vantage point is commanded by a central server. The central server is located in the University of Napoli. For this data collection, we use four MERLIN vantage points located in New-Zealand, USA, France, and Italy.

Unfortunately, some routers do not reply to IGMP probes sent by MERLIN, leading to an anonymous behavior that is similar to the one observed with traceroute. This phenomenon is called *IGMP filtering* [68] and is due to routers refusing to respond to IGMP probes (i.e., *local filtering*) and routers refusing to forward IGMP messages (i.e., *transit forwarding*). As a consequence, topologies collected with MERLIN might be fragmented.

The relevance of this phenomenon depends on the AS under investigation. Although there exist techniques for glueing together isolated components [68]. If filtering becomes an issue, we replace MERLIN by exploreNET [100], a new active probing tool that aims at revealing subnetworks of targeted ASes.

Once the data has been collected, we launch a large-scale Paris Traceroute [7] campaign from mPlane

probes targeting one (or more) destination(s) for each subnet /24 included in each collected AS. Those destination are selected among the addresses reported by MERLIN within largest connected components.

We use the scamper Paris Traceroute implementation [63]. In addition, we enable the MDA algorithm [8] (this corresponds to the `tracelb` option in scamper) to trace all per-flow load-balanced paths between each PlanetLab node and destination.

Note that the collected traces are successively manipulated such that the addresses belonging to the same router of the largest component are substituted with a unique identifier.

This Paris Traceroute campaign may discover additional links between routers, but also interfaces belonging to already reported routers and not yet reported ones. This is due to the fact that IGMP probing mainly focus on multicast portion of the network. All these newly elicited links and routers are added to the largest component of the AS of interest. The largest component of each AS and the set of traceroute traces collected from the Planetlab nodes represent the input for the IGP weights process.

2.10.3.2 Constrained System

At this point, we have MERLIN topologies with injected traffic. Basically, the classical weight inference method relies on inequality constraints based on sums of IGP weights [65]. This forms a constrained system that is solvable using Linear Programming and a given objective function (we decide to minimize the sum of weights in the graph, for example). It is also possible to encode additional constraints such as the weight symmetry or the fact that a direct edge between nodes is always the best.

One of the key point of using the MDA algorithm is that it provides a directed acyclic graph (DAG) vision of the network instead of the classical tree provided by standard traceroute. With a DAG, equality weight constraints are easier to retrieve.

2.10.3.3 Simulator

We evaluate our capacity to infer IGP weight through simulations. Fig. 2.60 illustrates our home-made simulator. The core of the simulator is *fakeroute* [14], a tool that simulates network topologies by intercepting `traceroute` probes towards a set of addresses. One of the interesting aspect of fakeroute is that topologies are made of text files describing the successive links, allowing for the presence of load balancers.

Our simulator works as follows: as input, it takes a static topology (it might be previously collected MERLIN topology, synthetic topology, ...) and transforms it into a topology graph that will be used to feed fakeroute. Fakeroute builds the successive links and closely works with `iptables` in order to intercept probes sent and simulate routers replies. A set of vantage points is then selected and each of them launches Paris Traceroute instances with the MDA algorithm.

Constraints are then built based on the static topology and traces collected by the vantage points during the simulation. Finally, the constraints are used to feed the solver which, in turns, output weight for the initial topology.

2.11 Accurate and Lightweight Anycast Enumeration and Geolocation

Geolocation of hosts according to their IP addresses is widely performed, both commercially (e.g., MaxMind [69]) and for research purposes. However there is a class of IP addresses on which today's geolocation methods fail: anycast addresses. IP-layer anycast [79] allows a group of *replicas*, or distributed servers that offer the same service, to share a single unicast IP address. Traffic directed to that address gets routed to just one replica, as determined by BGP routing. Geolocation services incorrectly assume that each unicast IP address corresponds to a host or set of hosts at a single location, and are unable to flag instances where this is not the case, not to mention being able to respond to a geolocation query with a set of hosts and locations.

One explanation for this deficit in anycast IP geolocation is that current services are oriented towards geolocation of client machines, not servers. Content distributors might geolocate clients in order to respect contractual constraints that limit them to sending certain content to certain countries; banks might geolocate clients for security purposes and as part of their due diligence concerning knowledge of their customers; advertisers might geolocate clients in order to profile populations and deliver targeted messages. However, more and more services are being delivered through anycast. DNS root and .ORG top level domain services [2, 44] have been anycast pioneers, using the technique to optimize availability and response time. The AS112 project [1], which reduces the load on DNS root servers caused by reverse DNS lookups for private network and link-local addresses, uses anycast. Multicast groups implement anycast mechanisms for rendez-vous point discovery [54]. To connect IPv6 networks across an IPv4 infrastructure, 6-to-4 relay routers [48] advertise an anycast IP prefix. Sinkholes [39] use anycast to detect, contain, and analyze worm activity. And now we are seeing commercial services such as content delivery networks (CDNs) increasingly being offered through anycast. For example, the EdgeCast CDN [30], which supports such major services as Pinterest and Twitter, is entirely anycast. With this rise of anycast, there is a concomitant need to understand anycast services. Internet service providers have a large commercial stake in managing over-the-top content flows, for example, and want to know where these flows are coming from and why. Businesses that rely upon services that are delivered via anycast need adequate troubleshooting tools. The locations from which services are provided are of interest to scientists ranging from security researchers to economists and social scientists, and of course network researchers.

Prior work on identifying, enumerating, and geolocating anycast services [33, 64] has focused entirely upon DNS. The techniques used are specific to DNS and are not generalizable to other services, as they rely upon DNS CHAOS [33] or EDNS [15]. The contribution of this work is that it provides a general technique for determining whether services are being offered via anycast on any unicast address. Further, it shows how to enumerate the replicas that are behind an anycast address and how to geolocate those replicas.

The geolocation methodology described in this section also differs significantly from prior IP address geolocation work [31,32,40,81,90]. Like [40], it is based upon speed-of-light constraints. But the technique that consists in using multilateration to infer the location of a single host is designed for single-host unicast and fails with anycast. We put an original twist on the technique, to adapt it to the situation where there may be multiple hosts. A simple image conveys the difference of our approach: that of discs that cover areas on a map. Single-host unicast multilateration employs discs centered around multiple vantage points, and locates the target host somewhere in the area formed by the intersection of the discs. In our technique, we recognize anycast in cases where such discs do

not overlap, and we position each host at a large population center within a disc. Our methodology consists in (i) framing the enumeration task as a maximum independent set optimization problem, that we optimally solve, (ii) framing the geolocation task as a classification problem, where we exploit side information such as population density, and (iii) solving both problems iteratively, with output of the geolocation task fed back to the enumeration task, which provides sizeable benefits.

The remainder of this section is organized as follows. We first state our objectives in Sec. 2.11.1, and then explain both our overall workflow as well as the enumeration and geolocation techniques in Sec. 2.11.2. Validation and calibration of these techniques over a dataset gathered in PlanetLab is the object of Sec. 2.11.3. We then run our algorithm on a fully fledged measurement campaign from multiple measurement infrastructures in Sec. 2.11.4, comparing our results to state of the art techniques.

2.11.1 Problem Statement

The problem that we are trying to solve is: given a target unicast IP address, t, determine if there is an anycast service being offered via this address, enumerate the replicas that are offering this service, and geolocate those replicas. The means at our disposal are a relatively low number (in the hundreds) of measurement agents situated at m different locations around the world.

We assume that we can launch latency measurements from the vantage points and that we have accurate knowledge of their positions, expressed as latitude and longitude (lat, lon). Such infrastructures are realistic to obtain, as evidenced by PlanetLab and RIPE Atlas, among others. The geolocation of each vantage point in such services is typically reported by the person who hosts the measurement agent, and these reports can be verified through unicast geolocation services [31, 32, 40, 81, 90], to check for initial accuracy and to catch cases when an agent is moved to another location.

We use latency measurements to identify discs (circles centered around the vantage points) where anycast replicas lay. While constraint-based geolocation is not a new technique as far as single-host unicast geolocation is concerned, we face a very different problem, as we need to identify and geolocate an unknown number of replicas as opposed to locating a single host. The identification problem is an *optimization problem*, in which an optimal solution consists in identifying all of the replicas. The geolocation problem is a *classification problem*, in which we try to select, from a set of discrete locations within each disc, the most likely position of the anycast replica in that disc.

Our design goals are:

- *completeness*: to fully enumerate anycast replicas
- *accuracy*: to geolocate replicas with city-level precision
- *reliability*: to avoid false positive by design
- *flexibility*: to avoid relying upon or exploiting service- or protocol-specific information
- *low overhead*: to use the smallest possible number of vantage points

Completeness and *accuracy* are obvious goals, necessary to achieve sound results. Sec. 2.11.4 compares the results of our technique to the most recent state of the art [15, 33].

Reliability by design (i.e., absence of false positive detection) is a desirable property that contributes to flexibility of use. For instance, an accurate and lightweight anycast identification technique could be applied to the detection of IP prefix hijacking: it could monitor IP addresses within an address prefix that are expected to be single-host unicast addresses, raising an alarm if anycast behavior is detected. Avoiding false alarms by design would be a necessary condition for such a service.

Flexibility allows the technique to adjust to recent trends in anycast deployment, such as its increasing use for CDNs, and to continue working in the face of new and unprecedented trends. As anycast deployment has until very recently been limited to DNS, it is understandable that that most available techniques are bound to this specific protocol [10,33,88] or exploit new developments in DNS [15]. This is in contrast to our protocol- and service- agnostic technique.

Low overhead makes it easier to design and operate continuously running services. Most of the initial studies on DNS anycast characterization [10,88] relied on a comparable number of vantage points to ours (e.g., about 200 PlanetLab nodes), however more recent studies have employed from 20k vantage points [11] up to 200k recursive DNS resolvers [15] plus 60k Netalyzr datapoints [33]. Our results show this increased overhead to be unjustified, as we achieve similar completeness and accuracy to the most recent state of the art [15,33] with less than 1/1000th of the vantage points.

2.11.2 Methodology

We illustrate our workflow with the help of Fig. 2.61. From a high level viewpoint, starting from a *dataset of latency measurements* $\mathcal{D}$, that we gather from controlled vantage points (Sec. 2.11.2.1), we illustrate the anycast detection condition (Sec. 2.11.2.2). Our *enumeration* approach consists in identifying areas on the globe (the set $\mathcal{E}$) for which we are confident they contain at least one anycast instance (Sec. 2.11.2.3). For each such area, we then perform a *geolocation* step by identifying the most probable city hosting the instance in a set $\mathcal{G}$ (Sec. 2.11.2.4). To enhance our coverage, we adopt an *iterative* approach, by feeding the solution to the geolocalization $\mathcal{G}(k)$ at step k back to modify the selection of vantage points in the input dataset $\mathcal{D}(k+1)$ at step $k+1$ (Sec. 2.11.2.5). We now describe each step in more details.

2.11.2.1 Latency measurement

For any given target t, we conduct latency measurements from PlanetLab [80] and from the RIPE infrastructure [84]. For each vantage point p, the measurement infrastructure yields a delay measurement $\delta(p,t)$ representing the round trip propagation delay required for a packet to travel from p to the closest instance of t and back to p.

As an individual latency measurement $\delta_i(p,t)$ can be affected by queuing delay, we estimate $\delta(p,t) = \min_i \delta_i(p,t)/2$ by halving the minimum value of successive round trip time (RTT) measurements (10 in this work). Since network operators generally keep their links to a low load, it is highly probable that at least one of those probes traverses the network without facing congestion in router queues. Although forward and backward paths are not necessarily symmetric, by halving the RTT we make the worst case assumption of maximal distance from the vantage point.

Despite our measurement campaign (Sec. 2.11.4) includes multiple kind of RTTs measurement (e.g., ICMP, DNS), without loss of generality we focus our attention on ICMP measurements (i.e., the most general) for the remainder of this section.

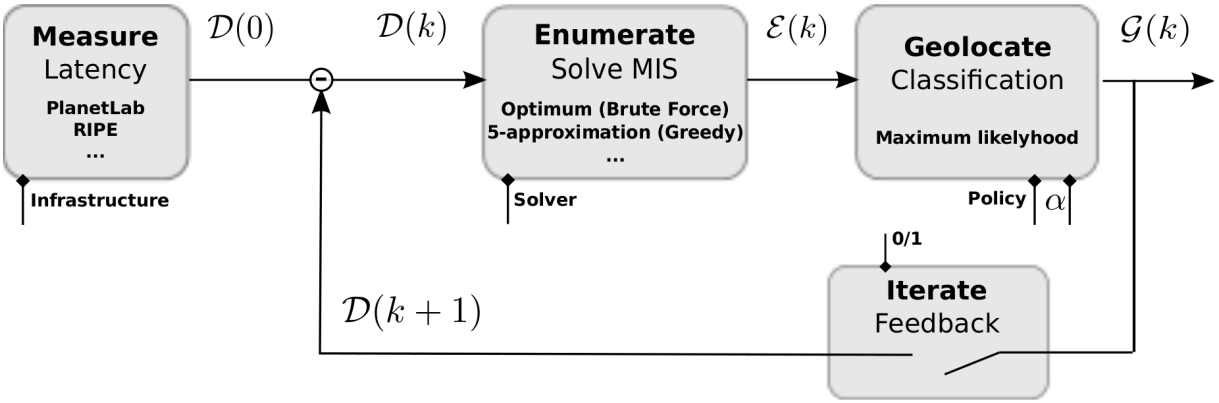


Figure 2.61: Methodology workflow

2.11.2.2 Anycast Detection

Prior to enumerating anycast instances, we must detect whether there are indeed anycast replicas behind a given unicast IP address. We do so by detecting speed-of-light violations in our dataset by comparing latency measurements δ to the expected propagation time due to speed-of-light considerations.

First, we map individual latency measurements $\delta(p,t)$ to geographic areas. Let $c_f = c_0/r_f$ be the speed of light in an optical fiber, with $c_0 \approx 3 \cdot 10^6 m/s$ the speed of light in a vacuum, and $r_f \approx 1.52$ the optical fiber index of refraction. A speed c_f is clearly optimistic, as it ignores time spent by equipment to process and forward packets. Given latency measurement $\delta(p,t)$, it follows that $d^+(p,t) = c_f\delta(p,t)$ is an upper bound on the actual distance $d(p,t)$ that the packet has possibly traveled. As we know the vantage point's latitude and longitude $(\text{lat}_p, \text{lon}_p)$, we identify the area in which we are confident the target t seen by p is located: as shown in Fig. 2.62, this is a a disc of radius $d^+(p,t)$ centered at p, that we denote $\mathcal{D}_p$. We further denote the set of all discs for all vantage points in our dataset as $\mathcal{D}$ (we drop the step k notation unless strictly necessary).

Next, we consider pairs of latency measurements $\delta(p,t)$ and $\delta(q,t)$ for the same target. Specifically, given two vantage points p, q we compute their geodesic distance $d^g(p,q)$ according to Vincenty's formulæ. Since packets cannot travel faster than light, if

$$d^g(p,q) > d^+(p,t) + d^+(q,t) \geq d(p,t) + d(q,t) \tag{2.10}$$

then our measurements indicate that p and q are in contact with two separate anycast replicas.

Some remarks are in order. First, (2.10) compares distances with homogeneous dimensions, that are however gathered with different techniques. Note that considering the geodesic distance $d^g(p,q)$ between vantage points yields a *conservative lower bound* to the expected propagation time between p and q, as a packet will not travel along a geodesic path but will follow a path shaped by physical and economic constraints (i.e., the geography of fiber deployment, optoelectronic conversion, BGP routing, etc.). Conversely, $d^+(p,t) = c_f\delta(p,t)$ *optimistically upper bounds* the distance that a packet may have traveled during $\delta(p,t)$.

As the the inequality is violated only when the conservative lower bound exceeds the optimistic upper bound, it follows (2.10) is conservative in detecting anycast instances, and by definition avoids

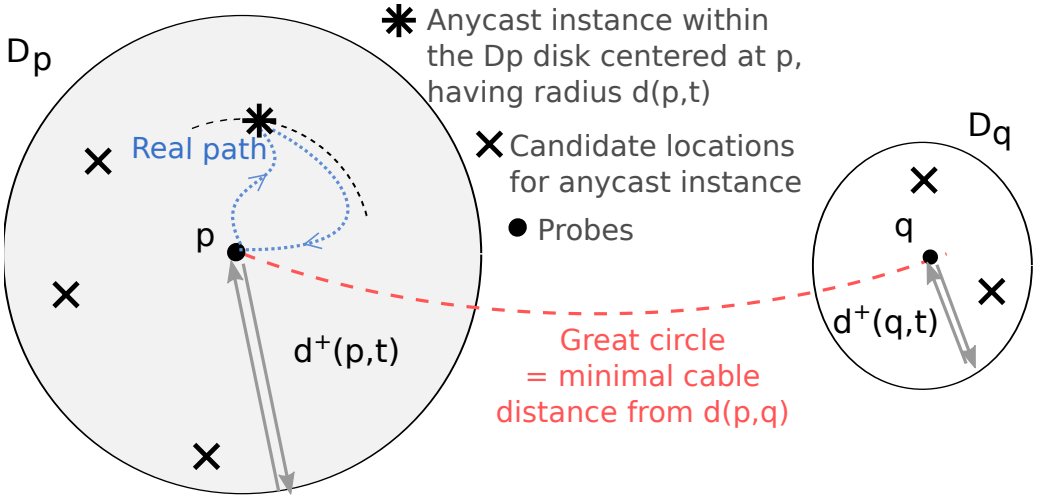


Figure 2.62: Synoptic of anycast instance detection via latency measurements

raising false positive anycast instances (i.e., flagging as anycast a single-host unicast target).

2.11.2.3 Anycast Enumeration

From a geometric viewpoint, a sufficient condition for (2.10) to be violated is that the two discs $\mathcal{D}_p$ and $\mathcal{D}_q$ do not overlap. Any time this condition is encountered, we can be certain that there are anycast replicas corresponding to the target unicast IP address t. While this observation is not per se particularly novel [64], we are the first to leverage a full set of distributed measurements in the study of anycast deployment and its geographical properties. Notice that this is extremely important since, while false negatives are possible on a single inequality (i.e., flagging as single-host unicast an anycast target), the chances of a false negative drop with the use of multiple pairs of vantage points.

It is possible that multiple anycast instances may be located within a given disc. Although the aim of anycast is to offer services from distinct locations, the locations may be distinct from an IP routing point of view but not distant geographically from each other. Therefore, our technique can only provide a lower bound of the number of anycast instances that correspond to our observations.

To achieve our joint goals of enumeration and geolocation, we model the problem as a *Maximum Independent Set (MIS)* problem. Our aim is to find a maximum number of vantage points (and corresponding discs) for which we are confident they contact distinct anycast instances (an instance being included in the disc). To do so, we select a maximum subset of discs $\mathcal{E} \subset \mathcal{D}$ such that:

$$\forall \mathcal{D}_p, \mathcal{D}_q \in \mathcal{E}, \qquad \mathcal{D}_p \cap \mathcal{D}_q = \emptyset \tag{2.11}$$

The enumeration problem is thus solved by the subset $\mathcal{E}$, whose cardinality $|\mathcal{E}|$ corresponds to the minimum number of instances that avoid latency violations, and which represents thus a plausible explanation to our observations. Notice that $|\mathcal{E}|$ is a lower bound on the number of anycast instances, since due to the conservative definition of (2.10) we might have removed discs that overlap due to noisy measurements. Additionally, a coarse location of anycast replicas is represented by each disc of $\mathcal{E}$, that constitutes the starting point for the finer grained geolocation of Sec. 2.11.2.4.

Algorithm 1 Greedy 5-approximation to MIS for anycast instances enumeration

Require: A set of disc $\mathcal{D}$
Ensure: A set of disc $\mathcal{E}$ such as $\forall p, q \in \mathcal{E}, \mathcal{D}_p \cap \mathcal{D}_q = \emptyset$
  Initialization: sort discs in $\mathcal{D}$ by increasing radius size
  Initialization: $\mathcal{E} \leftarrow \emptyset$
  for all disc $\mathcal{D}_d$ of $\mathcal{D}$ **do**
    for all disc $\mathcal{D}_e$ of $\mathcal{E}$ **do**
      if $\mathcal{D}_d \cap \mathcal{D}_e = \emptyset$ **then**
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{D}_e\}$
      end if
    end for
  end for

Although the MIS problem is NP-hard, it can be solved in finite time for small number of vantage points with a brute force approach. This allows us to compare the solution of known greedy approximate solutions: while a simple greedy strategy has poor performance in general ($(n-1)-$approximation) the situation improves by simply sorting discs in increasing radius size (5-approximation) as shown in Algorithm 1. We point out that, even though more refined solutions do exist [50] that achieve $(1-\frac{2}{k}) - OPT$ performance, they are however computationally very costly $n^{O(k^4)}$ – and as we will show later in Sec. 2.11.3, the greedy solution often performs well in practice.

2.11.2.4 Anycast Geolocation

Our aim being to provide geographic locations at city granularity, we need to refine the preliminary location that is output by the enumeration algorithm. We opt for city granularity for two reasons. First, note that a 1 ms difference in latency measurement corresponds to a 100 km disc in geodesic distance terms. It follows that great trust should be put in latency measurements to achieve finer-grained geolocation. Second, notice that ISPs and system administrators often use machine names that map to the city they are serving, which allows us to assess the correctness of our geolocation technique.

As opposed to classical approaches that operate in the geodesic (or Euclidean) space by constructing density maps of likely positions (see references in [32]), or assessing target location to be the center of mass of multiple vantage points [15], we transform the geolocation task into a classification problem as in [31]. Specifically, since our output is a geolocation at city level granularity, we shift the focus from identifying a geographical locus $(lat, lon) \in \mathcal{D}_p \subset \mathbb{R}^2$ to identifying which city $\mathcal{C} \in \mathbf{N}$ contained in the disk $(lat_C, lon_C) \in \mathcal{D}_p$ is most likely hosting the anycast instance.

This focus shift greatly simplifies the problem in two ways: first, it significantly reduces the space cardinality, and, second, it allows us to further leverage additional information with respect to delay or distance measurements, namely the city population. Our reasoning extends previous work [31], which argues that IPs are likely to be located where humans are located: in other words, due to

the distribution of population density, large cities represent the likely geolocation of single-host unicast IP addresses. We further argue that, since anycast replicas are specifically engineered to reduce service latency, they ultimately have to be located close to where users live: hence the bias toward large cities is again likely to hold for server side anycast IPs as well.

Our geolocation step outputs IATA airport codes as shorthand for cities. For each of the discs that is output by the enumeration phase, some of the over 7,000 airport codes available worldwide may be contained in the disc. Aside from the trivial case where a single airport is contained in the disc, in the general case multiple airports $\{A_i\} \in \mathcal{D}_p$ are contained in any given disc (represented as a cross in Fig. 2.62). The output of the geolocation phase can thus be expressed with disc-airport pairs $\mathcal{G} = \{(\mathcal{D}_i, A_i)\}$ according to the notation of Fig. 2.61.

To guide our selection of the most likely location of a site, we employ two metrics, namely: (i) the population of the main city c_i that the airport A_i serves, for the reasons described above, and (ii) the distance between the airport and the disc border $d(p,t) - d(p,A_i)$, using geographic proximity as a proxy for topological proximity in the routing space.

For a given disc $\mathcal{D}_p$ we compute the likelihood of each airport $\{A_i\} \in \mathcal{D}_p$ for all airports in the disc, as:

$$p_i = \alpha \frac{c_i}{\sum_j c_j} + (1-\alpha)\frac{d(p,t) - d(p,A_i)}{\sum_j d(p,t) - d(p,A_j)} \tag{2.12}$$

where $p_i \in [0,1]$ follows from the normalization over all airports $\{A_i\} \in \mathcal{D}_p$ of the c_i (population of the main city served by airport A_i) and of the $d(p,t) - d(p,A_i)$ (the distance of the airport i from the disc border) contributions. A parameter $\alpha \in [0,1]$ tunes the relative importance of population vs. distance in the decision, in between the distance only ($\alpha = 0$) vs. city only ($\alpha = 1$) extremes.

Based on the p_i values, we devise two maximum likelihood policies that return either (i) a single $A_i = \text{argmax}_i p_i$ or (ii) all locations (A_i, p_i) annotated with their respective likelihoods. These policies involve a trade off, as returning all locations increases the average error (since in case $\text{argmax}_i p_i$ is correct, it pays the price of incorrect answers for $1 - p_i$), whereas returning a single location possibly involves a bigger risk.

2.11.2.5 Iteration

Recall that the enumeration step lower bounds the number of instances, due to the possibility of overlapping discs. Now, consider that the geolocation decision in effect transforms a disc $\mathcal{D}_p$, irrespective of its original radius, into a disc $\mathcal{D}'_p$ centered around the selected airport with arbitrarily small radius.

Hence, we argue that, provided the geolocation technique is accurate, it would be beneficial to transform the original set of discs $\mathcal{D}$ by (i) remapping $\mathcal{D}_p$ to $\mathcal{D}'_p$ and (ii) excluding from $\mathcal{D}$ those discs that contain any of the geolocated cities $\mathcal{D}'_p$.

Consider for the sake of the example the situation depicted in Fig. 2.63, where three discs centered around vantage points a, b and c overlap. Let the solution to the enumeration problem at step k select vantage point b (hence the smallest disc $\mathcal{D}_b$), and let furthermore the solution to the geolocation problem select city c_1. By coalescing $\mathcal{D}_b$ around c_1 (with arbitrarily small radius), it follows that at step $k+1$ disk $\mathcal{D}_a$ no longer overlaps with any other disc, meaning that it would be possible to discover another anycast instance (i.e., c_3 in the example) that was previously precluded (whereas disk $\mathcal{D}_c$ still overlap and discovery of c_0 is still precluded).

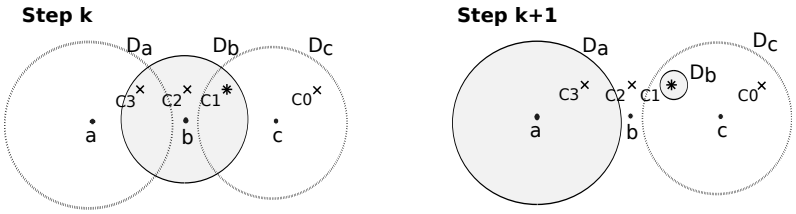


Figure 2.63: Synoptic of iterative workflow

Denoting with $\mathcal{A}(k)$ the subset of airports geolocated up to step k, and with $\mathcal{G}(k)$ the geolocation at step k (considering a single airport selected per disk for the sake of simplicity):

$$\mathcal{G}(k) = \{(\mathcal{D}_i, A_i) \in \mathcal{E}(k) \times \mathcal{A}(k))\} \tag{2.13}$$

we have that the dataset $\mathcal{D}(k+1)$ as input to the numeration problem at step $k+1$ would be:

$$\mathcal{D}(k+1) = \mathcal{D}(k) \backslash \{\mathcal{D}_i : \exists (\mathcal{D}_i, A_i) \in \cup_{i=1}^{k} \mathcal{G}(i)\} \tag{2.14}$$

This workflow can be iterated until no further disc can be added that does not overlap. At each iteration, the set of geolocalized cities grows, so that the set of discs that no longer overlap diminishes, which keep the running time reasonably bounded. Note that iterative operations can be employed irrespectively of the underlying solver (i.e., brute force, greedy, etc.).

2.11.3 Validation

We validate our methodology against publicly available ground truth. For the sake of simplicity, this subsection considers just 200 PlanetLab vantage points, and defers to Sec. 2.11.4 a more comprehensive study with multiple measurement infrastructures. We first explain our ground truth dataset (Sec. 2.11.3.1), then perform a calibration of the enumeration (Sec. 2.11.3.2), and the geolocation (Sec. 2.11.3.3) tasks.

2.11.3.1 Ground truth

Our technique is not bound to a particular service. However, we are limited in our validation to the availability of reliable ground truth: we thus focus on DNS root servers, 12 of 13 of which use anycast. Indeed, while in general it is challenging to enumerate the sites of an anycast group and associate each site with its geographic location, we are able to build a reliable ground truth for enumeration and geolocation of root servers F, I, K, and L that are operated by ISC, Netnod, RIPE NCC, and ICANN respectively.

Operators of the root servers maintain an official website [2] with maps annotated with the number and geographic distribution of deployed sites around the world. Additionally we use existing techniques to reliably disambiguate between different instances of the same DNS root server, that exploit queries of the DNS CHAOS class. We stress that CHAOS measurements are only required to assess the enumeration recall and the geolocation accuracy of our proposed methodology (that only relies on distributed delay measurements and is thus not bound to the specific DNS use-case), but are otherwise not used for enumeration and geolocation.

Table 2.12: Recall of anycast enumeration algorithms with 200 PlanetLab nodes. Gain [%] with respect to the greedy baseline is reported.

	Root server							
Algorithm	**F**		**I**		**K**		**L**	
Greedy	17		13		9		20	
BruteForce	18	+6%	13	-	9	-	20	-
iGreedy	18	+6%	15	+15%	10	+11%	22	+10%
iBruteForce	21	+23%	15	+15%	10	+11%	22	+10%
Dataset CHAOS UB	22		23		11		33	
Published GT	55		46		17		128	

To build a reliable ground truth, we issue distributed IPv4 DNS queries of class CHAOS, type TXT, and name hostname.bind to DNS root servers. Despite the fact that CHAOS replies do not follow a standard format, some operators use IATA airport codes (e.g., AMS, PRG in root severs F and L respectively) and IXPs short names (e.g., AMS-IX, BIX, MIX in root server K) to name their infrastructure servers. In other few cases (e.g., root server I), operators use arbitrary codes, but make publicly available a list that maps site codes to locations. In sporadic cases, multiple CHAOS names are located in the same city: as we are interested in locating geographically distinct anycast replicas, as opposite to enumerating the number of physical or virtual servers operating on a site, we coalesce all replicas located in the same site.

2.11.3.2 Anycast Enumeration

We first benchmark the performance of the anycast enumeration technique. We point out that our aim in this section is not to show the absolute performance (that will be the object of Sec. 2.11.4), but rather to *relatively compare* the performance of the different solvers (i.e., greedy vs. brute force), and to additionally gauge the impact of the iterative workflow.[8]

Under this perspective, Table 2.12 reports the number of anycast replicas found by solving the maximum independent set problem $|\mathcal{E}|$ for different solvers. Solvers are sorted top to bottom in increasing performance, and the percentage gain with respect to the greedy baseline is also tabulated. For completeness, the table reports the dataset upper bound (UB) and the published ground truth (GT), which represent the number of distinct CHAOS names we were able to observe in our dataset, and the number of distinct servers that are publicly reported on the web sites of root servers, respectively. From the table, we gather that the greedy solver achieves in practice performance that is, in most of the cases, as good as that of the brute force solution (I, K, L) or anyway comparable (F). More interestingly, the iterative workflow produces benefits that are sizeable and consistent across datasets and solvers.

Given the strikingly similar recall performance, and considering that the running time of iGreedy (*hundreds of milliseconds*) is orders of magnitude smaller with respect to that of the brute force approach (*hundreds to thousands of seconds*), it seems reasonable to limitedly consider the iterative greedy (iGreedy) solution in what follows. Notice indeed that while we were able to run the brute force solution on the PlanetLab dataset, its cost is prohibitive for larger datasets considered in Sec. 2.11.4. It also follows that, while refined solutions do exist [50], they are not appealing due

[8]The performance of the iterative workflow also depends on the geolocation policy: we fix for the time being the *argmax* policy with $\alpha = 0.5$ and justify this choice in Sec. 2.11.3.3.

to the good recall and short running time of the greedy solver.

Before evaluating the accuracy of our anycast geolocation technique, we proceed with a careful manual validation of the iGreedy enumeration of servers around the world against the published ground truth, and encounter only three spurious cases: one case related to a manifestly wrong location of a PlanetLab node, as well as two cases where airports are located few kilometers away from the discs border (which, as servers are unlikely physically hosted in the airports, is intrinsically tied to imprecision due to the naming convention).

2.11.3.3 Anycast Geolocation

We then precisely assess the impact of the geolocation parameters, i.e., the distance vs. population weighting factor α and the selection policy argmax vs. proportional, by measuring (i) the percentage of correct classification (i.e., geolocation) and (ii) the mean geolocation error in kilometers. For any given disk $\mathcal{D}_p$, let us denote with $A^\star$ the airport code given by the ground truth, and further denote with A_i the different airports that are located in $\mathcal{D}_p$. In case no airport falls in $\mathcal{D}_p$, then we remove the disk, which allows to include further disks at the next iteration. Considering the argmax policy, in case $A^\star = A_i$ (with i such that $\text{argmax}_i\, p_i$), the classification is accounted as correct and the error for this instance is $Err_p = 0\,\text{Km}$. In case $A^\star \neq A_i$, then the classification is erroneous, and off by a distance $Err_p = d(A_i, A^\star)$. In the proportional policy instead, the classification is accounted as correct only for p_i (i.e., proportionally to the percentage of time the correct instance would be selected). The geolocation error for this instance is then computed over all airports inside the disc, and weighted according to the respective likelihood of each airport $Err_p = \sum_j d(A_j, A^\star)p_j$.

Fig. 2.64 shows the percentage of correct geolocation (left y-axis) and the mean geolocation error (right y-axis) in kilometers for the different policies (argmax vs. proportional) and weighting factor (α) under study. Again, in this subsection we are more interested in the *relative comparison* and the calibration of the technique, rather than absolute performance: under this light, it appears that argmax is preferable to proportional policy. While this does not hold in general, in the iGreedy solution disks having small size are more represented: it follows that distance-based criterion is already fairly accurate, and can be additionally improved by properly taking into account knowledge about city population. Hence, in the remainder of this work we consider an argmax policy that equally weighs distance and population ($\alpha = 0.5$), which leads to jointly high geolocation correctness and low error.

2.11.4 Measurement campaign

We now report results of our measurement campaign, that include multiple datasets gathered at different times, and using different measurement infrastructure (Sec. 2.11.4.1). Specifically, our main aim is to critically compare our results to the state of the art (Sec. 2.11.4.2) and to further assess the robustness of our methodology to vantage point location (Sec. 2.11.4.3).

2.11.4.1 Datasets

We collect datasets from multiple measurement infrastructures (PlanetLab, RIPE) and protocols (ICMP `ping`, DNS CHAOS application layer measurements). In this section, we consider ICMP mea-

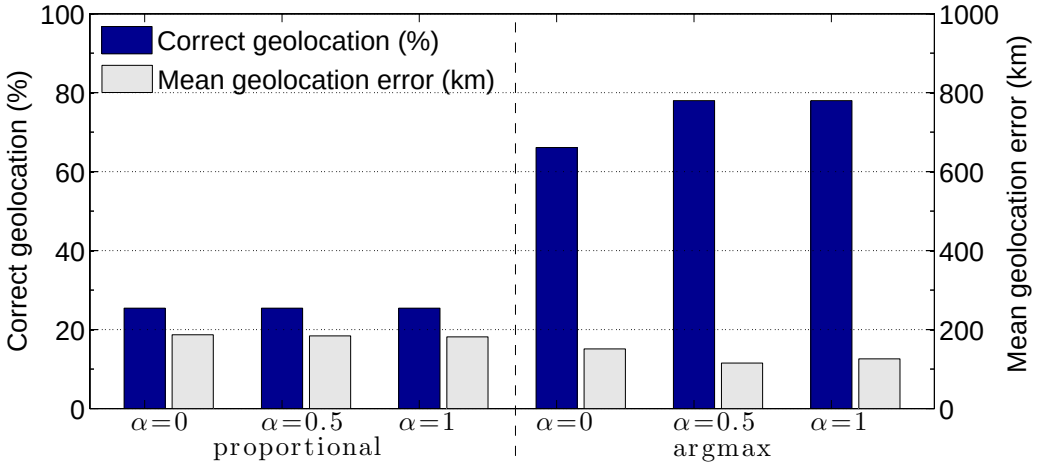


Figure 2.64: Geolocation performance with 200 PlanetLab nodes: Percentage of correct geolocation and mean geolocation error for different policies (argmax vs. proportional) and weighting factor (α).

surements over both PlanetLab and RIPE, and build our ground truth as early explained in Sec. 2.11.3.1.

In the case of PlanetLab [80], we perform measurements from about 200 nodes located in 26 countries. Node geolocation is provided by PlanetLab, and the PlanetLab Europe nodes position is validated with unicast geolocation techniques.

In the case of RIPE [84], we perform measurements from over 6000 nodes located in 350 ASes and 122 countries. Geolocation of RIPE nodes is provided by the users hosting a node. If this information is missing, RIPE finds the location using MaxMind [69] over the IP address of the node.

Since the full RIPE infrastructure comprises 30 times more nodes than PlanetLab, we also consider a subset having approximately the same size, where we select nodes in a stratified sampling as a function of the distance (to ensure geographic vantage points diversity). Unless otherwise stated, results reported in the following are gathered with this dataset, that we denote RIPE200.

2.11.4.2 Comparison Against State of the Art

Eye candy. For the sake of illustration, Fig. 2.65 reports an example of results for root server L (iGreedy, argmax, $\alpha = 0.5$). The map reports vantage points (black dots) and the results of iGreedy as shaded disks that contain either correct ✓ or erroneous $\times$ geolocation markers (and in the last case the location of the ground truth P as well). The map additionally reports instances that are missed M, either because they are not observed in our measurement, or because they are observed in disks that overlap (represented as circles with no shading).

Several interesting observations are worth making. First, note that despite the very large discs around some vantage points, the MIS formulation factors them out so that no false positives are raised. Second, notice an example of vantage points that our iterative workflow allows to include (i.e., discs of the Bruxelles and Paris vantage points intersect). Third, population bias yields to misclassification for the point located in Porto, Portugal: this vantage point exhibits a relatively large latency (6 ms) to hit a target also located in Porto, so that the disk is large enough to include Madrid (population of 3.3M) which is an order of magnitude more populated than Porto (population of

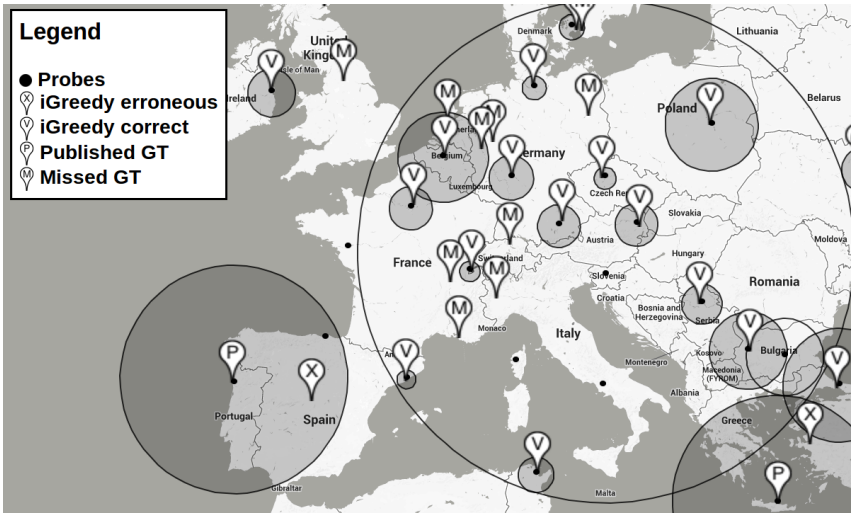


Figure 2.65: Enumeration and geolocation of sites for root server L with iGreedy

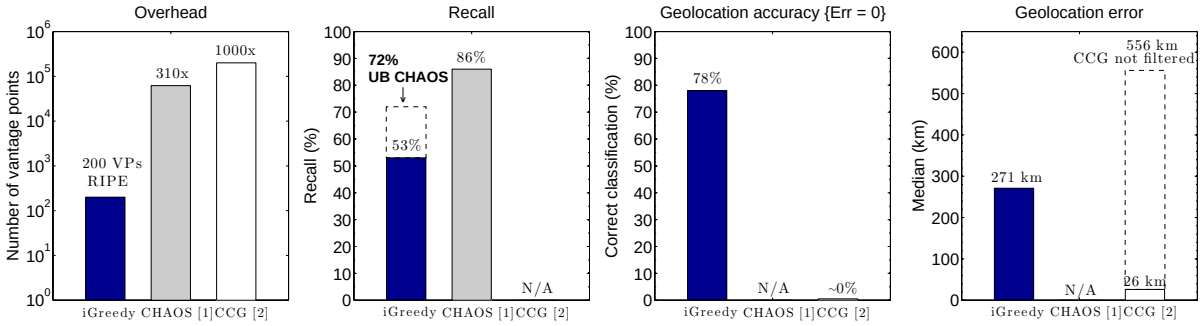


Figure 2.66: Performance of iGreedy: Comparison of probing overhead, recall, geolocation accuracy, and geolocation error for misclassified instances with the state of the art [15, 33]

250K). Refinement of the classification technique is part of our future work (e.g., using logarithmic instead of linear weighting of city population, including distance in the AS space [42], etc.).

Comparison at a glance. Fig. 2.66 summarizes the results discussed in this section. As references for comparison, we take [15, 33]. Notice that the enumeration results of [33] are *directly quantitatively* comparable, as [33] employs F and other root servers as a case study. Geolocation results of the Client-Centric Geolocation (CCG) technique recently proposed in [15] are instead *only qualitatively* comparable, as they target the Google infrastructure. Yet, qualitative comparison allows to gain some interesting methodological insights, that are worth illustrating.

At a glance, Fig. 2.66 shows that iGreedy (a) reduces the measurement overhead by several orders of magnitude with respect to [15, 33], (b) has comparable yet lower enumeration recall than [33], (c) is able to correctly guess instance location 3/4 of the time unlike [15, 33], (d) has a comparable geolocation error to [15] for the misclassified anycast instances. Not shown in the picture, our methodology is protocol agnostic, unlike [15, 33] that both rely on DNS.

Enumeration. Enumeration results are directly comparable, as [33] employs root servers as a case study. We therefore dig further the comparison in Fig. 2.67. Interestingly, while [33] achieves 100% recall for root servers F and I and 94% and 73% recall for root servers K and L, it does so

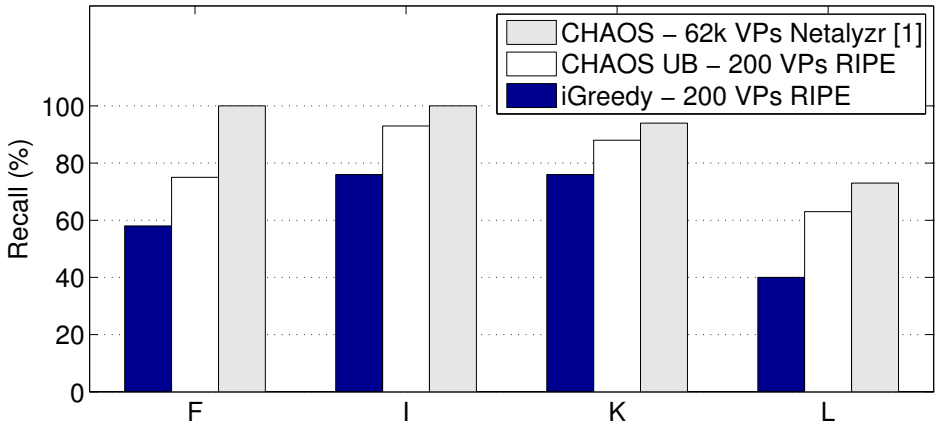


Figure 2.67: Enumeration of anycast servers: Comparison of recall with iGreedy over 200 RIPE nodes vs. method in [33] from 62k Netalyzr vantage points.

by issuing DNS CHAOS queries from 62K vantage points (the Netalyzr dataset). In contrast, using only 200 RIPE nodes and the same type of queries, we achieve close recall values for root servers I, K and L (93%, 88%, and 63% respectively). Intuitively, anycast detection relies on the availability of geographically dispersed vantage points. Intrinsically, this means that the datasets used in [33] are highly redundant. Specifically, while the Netalyzr [57] dataset contains over 62k data points, these include possible multiple trials from the same users; similarly, if Netalyzr is popular in a given region, the availability of several points is not useful to increase the overall recall. The same goes for the 200k recursive DNS resolvers exploited by [15, 33] – a clear overkill.

As expected, iGreedy enumerates then only a subset of the CHAOS upper bound (UB) with a recall that remains above 55% (except for root server L). While these results are already satisfactory as they are performed from only a handful of vantage points, and additionally allow precise geolocation of the anycast instances, as future work, we aim at increasing iGreedy recall. A promising direction is to merge multiple iGreedy solutions (which is possible due to the very low running time of iGreedy), performed over several selections of fewer vantage points from the same dataset (as chances of overlap reduce, each selection hopefully yields to discovery of some new instances; and since selections are independent, the expected overall recall would increase as well).

Geolocation. We finally qualitatively compare geolocation accuracy against CCG [15] in Fig. 2.68, where it is worth stressing once more that [15] focuses on the geolocation of Google front-end servers, so that the results are not directly comparable in quantitative terms. Yet, as our study is the first to propose anycast geolocation, [15] is the closest technique we can compare with.

First, CCG [15] leverages 200k recursive DNS resolvers to issue over 4M special EDNS-client-subnet queries including IPv4 ranges that would be typical of real Internet clients – hence the Client-Centric Geolocation (CCG) name. Depending on the IPv4 range, the Google DNS server returns a specific replica to be contacted: CCG geolocates such replica in the center of mass of all client IPs directed to this replica, where each client IP is geolocated via MaxMind. As it can be seen from Fig. 2.68, the CCG has a non negligible error that is intrinsically tied to the accuracy of the MaxMind database [81]. In contrast, in expressing the geolocation task as a classification problem, our technique yields 0 km error for 78% of the enumerated servers and 271 km median error for the misclassified instances.

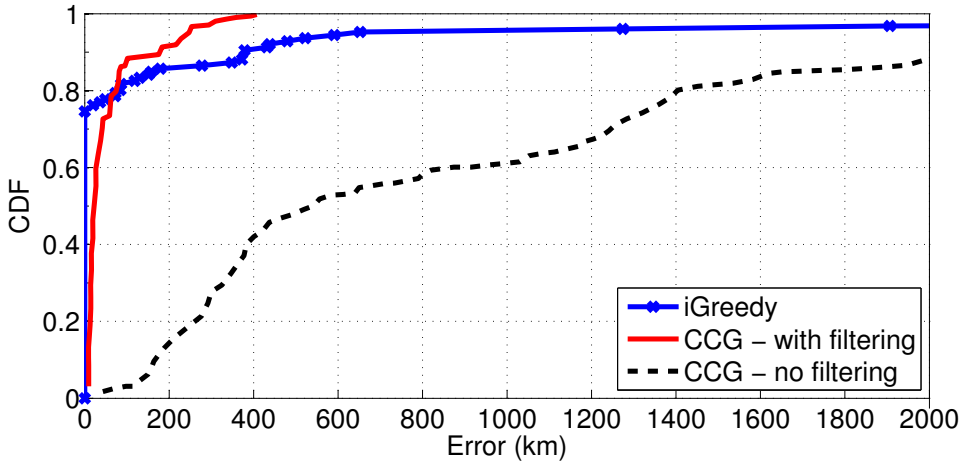


Figure 2.68: Geolocation: Comparison of distance error with iGreedy over 200 RIPE nodes vs. CCG [15] methodology issuing 4M requests over 200k open recursive DNS resolvers.

Additionally, to get rid of noise, CCG proposes to filter out from the cluster vantage points having a distance that exceeds the average by one standard deviation (over each cluster). While this filtering step improves significantly the accuracy of the geolocation, as it can be seen from Fig. 2.68, iGreedy is still better for the majority of the cases (notice the crossover of the two curves), despite the tail of the error distribution is higher (that could be upper-bounded with filtering, say all circles whose radius exceeds the average by one standard deviation – which we prefer to avoid).

Additionally, CCG filtering also leads to squander of networking and processing resources. In other words, not only are a huge number of vantage points used (200k, which is 1000x more than in our approach), but also quite a significant number of the results from these points (e.g., about 20% or 44k in case delays were normally distributed) are discarded a posteriori. In contrast, our geolocation technique starts from a parsimonious number of vantage points. It then selects only a single reliable vantage point as output of the enumeration phase (i.e., likely among the closest to the instance), after which it performs an informed decision (biased on the city population and the distance from the border).

2.11.4.3 Robustness with Respect to Vantage Point Selection

Finally, we assess the robustness of the methodology to changes in the vantage points selected. Table 2.13 summarizes the performance of iGreedy across 4 datasets of different cardinality. Interestingly, we observe that the number of vantage points has little effect on the recall. By applying a naive selection policy on the 6000 RIPE vantage points which consists in selecting 200 vantage points that are at least 100 km distant from each other, we are able to enumerate 127 servers out of the 149 discovered by the full set of vantage points (with a mean geolocation error of 361 km). The recall declines with the PlanetLab dataset which hints that the 200 vantage points selected there do no have good geographical coverage. Clearly, since the number of anycast instances does not exceed 246 for the selected root servers, it is enough to choose hundreds of vantage points that cover the top ASes and the highly populated areas for the enumeration task. A systematic study of vantage point selection is on our research agenda.

Table 2.13: Robustness with respect to vantage point selection

	RIPE			PlanetLab
Subset	full	rand	strat	full
Dataset cardinality	6000	500	200	200
iGreedy / CHAOS UB	76%	52%	73%	73%
iGreedy / GT	61%	28%	53%	26%
CHAOS UB / GT	80%	54%	72%	36%
Geolocated	76%	63%	78%	74%
Mean geolocation error (km)	333	569	361	162
iGreedy	149	68	127	65
Dataset CHAOS UB	196	132	174	89
Published GT [F,I,K,L]	246	246	246	246

2.12 Distributed Monitoring

This section aims at showing the computational limits of "centralized" computation at the root of any decision process as mPlane envisions them. For this purpose, we consider the task of joint traffic sampling by means of monitoring agents distributed along traffic paths; further referred to as cooperative monitoring problem. This example is nevertheless representative of a larger class of problems that can be formulated as generalization of the capacitated facility location problem and that finds broad applicability in the networking space. On the other hand, solving this problem is already of intrinsic value for mPlane as the dynamic placement and configuration of passive monitoring agents following the spatio-temporal dynamics of traffic paths remains largely unaddressed since so far. Consider a network graph as depicted in Fig. 2.69, the problem consists knowing the traffic demands to place and configure passive monitoring points such that exactly $k\%$ of the traffic flowing along each path is monitored. This general problem differs from the one consisting in extracting at least $k\%$ of the total amount of traffic. Indeed, in the present case, each traffic flow is monitored whereas in the second case the problem is unbalanced (some path can remain unmonitored $k = 0\%$ while for others the entire traffic they carry can be sampled). This situation is illustrated in Fig. 2.69. Assume that each traffic flow shall be monitored with $k = 80\%$. Along, the path from source node $s = 1$ to destination $t = 5$, a monitoring point is installed at the head-end of arc (1,6) sampling 10% of the traffic, at the head end of arc (6,4) sampling 20% of the traffic and at the head end of arc (7,5) sampling 50% of the traffic, leading to a total of $k = 80\%$. Observe also from this figure that along arc (4,7) the sampled traffic remains at 0% implying (if no other flow is monitored along that arc) that no monitor would be installed along that arc.

In turn, this motivates the introduction of collective solving of decision problems formulated as mixed integer programs. Collective solving implies very often to the master problem (using, e.g., Benders decomposition method) into subproblem solving per-node/agent in order to fit the nodal-decision process (local traffic capture/sampling, per-hop routing decision, etc.). When the optimization problem involves non-linear objective/constraints, its decomposition becomes even harder (e.g., using generalized Benders decomposition method). On the other hand, when the problem involves uncertainty in input data for instance (which is often the case in the networking domain due to the spatio-temporal variability of traffic, client demands/requests, etc.), domain-knowledge is still required to specify uncertainty sets to formulate their robust counterparts. Data-driven statistical learning methods can be customized or extended for this purpose.

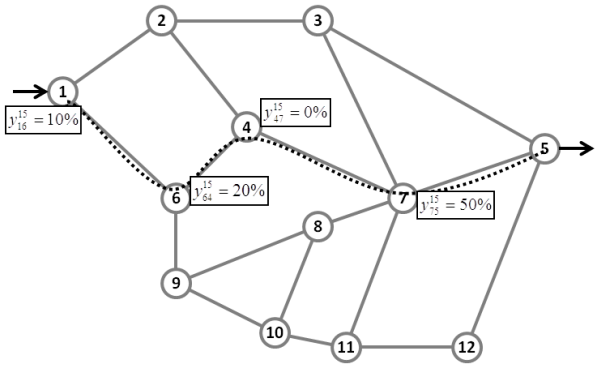


Figure 2.69: Traffic flow monitoring - Example

The cooperative monitoring problem consists, given a network topology represented by the graph $G = (V, A)$ with vertex set V and arc set A, in determining a set of arcs $(i, j) \in A$ where to place a set of monitoring points together with their optimal configuration in order to cooperatively realize a joint task of monitoring traffic flows. Monitoring a total fraction k of the traffic flowing along a given path involves the cooperation between monitoring points since the traffic sampled at a given monitoring point is not sampled again at another point along the same path. The corresponding optimization problem consists, knowing the traffic demands, in minimizing the total monitoring cost such that a given traffic monitoring task can be jointly realized. The monitoring cost includes the cost associated to the installation of a monitoring point along arc (i, j) and the cost associated to its configuration. The latter translates the capacity required at each monitoring point proportionally to the fraction of traffic captured at that point.

As the monitoring cost depends on the spatial distribution and the temporal properties of the traffic flows established across the network, as determined by the routing strategy, the resulting optimization problem formulates as a mixed-integer non-linear program (MINLP) where the continuous monitoring fraction variables are multiplied by the continuous flow variables. The latter can be derived by resolving the min-cost flow (MCF), the min-cost multicommodity flow (MMCF) or the multicommodity capacitated network design (MCND) problem depending on the routing strategy adopted. The formulation can also be dualized to determine the monitoring utility gain obtained when varying the budget constraint imposed on the total monitoring cost. The corresponding optimization problem consists in maximizing the monitoring utility given budget constraints on the monitoring installation and configuration cost.

The proposed formulation differs from the one developed in [19] translating the task of monitoring imperceptibly a fraction k out of the total amount of traffic. In the present case, for each flow part of a (preselected) set of demands, a total fraction k of the traffic is sampled per-flow instead of assuming that some flows can remain unmonitored ($k = 0$) while for others the entire traffic can be sampled ($k = 1$). Following the formulation documented in [96], the sampling rate for each flow at each monitoring point can only be adjusted independently from the others along the same path. A global sampling strategy is adopted in [16] where the sampling rate is multiplied by the traffic load of each arc instead of individual flows as proposed in this section.

This section is structured as follows. Sec. 2.12.1 details the monitoring cost minimization and monitoring utility maximization problems together with their respective formulation. The results of their execution on representative topologies are reported in Sec. 2.12.2 and compared with respect to the flow routing strategy. Finally, Sec. 4 concludes this section and provides for the main directions in terms of future work.

2.12.1 Optimization Models

2.12.1.1 Monitoring Cost Minimization

We are given a network topology modeled by a directed graph $G = (V, A)$ where V represents the finite set of nodes and A the finite set of arcs denoted by (i, j), i denoting the head-end and j the tail-end of arc (i, j). Each arc (i, j) installed from node i to node j provides a nominal maximum capacity κ_{ij} and an installation cost a_{ij}. We also have at our disposal a demand matrix D where $D(s, t)$ is the total amount of traffic sent from source node s to destination t, for any pair $s, t \in V,\ s \neq t$. The total fraction of traffic k to be monitored along each routing path is also provided as input parameter. When the formulation is capacitive, additional constraining capacities b_{ij} are associated to the monitoring points.

The following variables are defined. The binary variables v_{ij} indicate whether or not arc (i, j) is installed between head-end i and tail-end node j. The binary variables x_{ij} designate if a monitoring point should be installed at the head end along arc (i, j). The continuous variables represent the fraction of traffic flow y_{ij}^{st} sampled on the monitoring point installed along arc (i, j) out of the amount of traffic flowing on arc (i, j) from source s to destination t as indicated by the continuous variable f_{ij}^{st}.

The monitoring cost includes the installation cost c_{ij} for installing a monitoring point along arc (i, j) and the configuration cost. The configuration cost of each monitoring point installed along arc (i, j) is defined proportionally to its load as $\ell_{ij} = \sum_{s,t \in V} f_{ij}^{st} y_{ij}^{st}$, i.e., the sum over all source-destination pairs of the fractions y_{ij}^{st} of the traffic flows f_{ij}^{st} routed along arc (i, j) that are sampled at that point. Let m_{ij} denote the associated unit cost, the total configuration cost over all installed monitors is defined as $\sum_{(i,j) \in A} m_{ij} \ell_{ij}$. We also define the cost functions related to the link capacity installation when the MCND routing strategy is enforced and the routing cost function when either of the MCF, MMCF or MCND strategies is considered. The link capacity installation cost a_{ij} incurs each time an arc (i, j) of nominal capacity κ_{ij} is activated between node i and j. The routing cost $\phi(\kappa_{ij}, l_{ij})$ depends on how close the load $l_{ij} = \sum_{s,t \in V} f_{ij}^{st}$ of each arc $(i, j) \in A$ is to κ_{ij}. Any increasing convex cost function $\phi(\kappa_{ij}, l_{ij})$ can be considered to account for a routing cost per arc that increases proportionally to its utilization; here, we assume that this function is piecewise linear.

The problem of minimizing the total cost of monitoring traffic flows routed following a given strategy can be formulated by means of the MINLP described in Fig. 2.70 (and similarly for the MCND and MCF routing strategies). The first term of the objective function (2.15) corresponds to the routing cost and the second to the monitoring cost (i.e., the sum of the monitoring point installation and configuration cost). The flow conservation and demand satisfaction constraints (2.16) ensure that the demand flow requirements of each node as given by the matrix $D(s, t)$ are met. Constraints (2.17) impose that flows are sent only if the capacity of the arcs are not exceeded as $\min(\kappa_{ij}, \sum_{s,t \in V} D(s, t))$ defines a tight upper bound on the total load on each arc (i, j). Constraints (2.18) ensure that monitoring can be performed along arc (i, j) only when a monitor is being installed at the head end of that arc. Constraints (2.19) impose that for each monitoring point installed at the head end of arc (i, j), the fraction of the monitored traffic on all flows routed along that arc does not exceed the corresponding monitoring point capacity. Constraints (2.20) guarantee that the sum of the fractions k of the traffic sampled along each flow meets the monitoring task objective. Finally, to prevent dispersion, we impose with the constraints (2.21) that a monitoring point is installed only if a flow is to be monitored at a sampling rate of at least 1%.

For the formulation involving the MCND problem, the objective function includes as part of its term

$$\min\ w_1 \left[\sum_{(i,j)\in A} \phi\left(\kappa_{ij}, l_{ij}\right) \right] + w_2 \left[\sum_{(i,j)\in A} \left(c_{ij} x_{ij} + m_{ij} \sum_{s,t\in V} f_{ij}^{st} y_{ij}^{st} \right) \right] \tag{2.15}$$

subject to:

$$\sum_{j:(i,j)\in A} f_{ij}^{st} - \sum_{j:(j,i)\in A} f_{ji}^{st} = \begin{cases} D(s,t) & \text{if } i = s \\ -D(s,t) & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \qquad i,s,t \in V \tag{2.16}$$

$$\sum_{s,t\in V} f_{ij}^{st} \leq \min(\kappa_{ij}, \sum_{s,t\in V} D(s,t)) \qquad (i,j) \in A \tag{2.17}$$

$$y_{ij}^{st} \leq x_{ij} \qquad (i,j) \in A, s,t \in V \tag{2.18}$$

$$\sum_{s,t\in V} f_{ij}^{st} y_{ij}^{st} \leq b_{ij} x_{ij} \qquad (i,j) \in A \tag{2.19}$$

$$\sum_{(i,j)\in A} y_{ij}^{st} = k \qquad s,t \in V \tag{2.20}$$

$$y_{ij}^{st} \geq 0.01 \qquad (i,j) \in A, s,t \in V \tag{2.21}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{2.22}$$

$$f_{ij}^{st} \geq 0 \qquad (i,j) \in A, s,t \in V \tag{2.23}$$

Figure 2.70: Mixed integer programming formulation (with MMCF)

the link installation cost $\sum_{(i,j)\in A}(a_{ij}v_{ij})$ in addition to the routing cost; the second term remains unchanged compared to (2.15). The RHS of the capacity constraints (2.17) is multiplied by the binary variable v_{ij} such that the installed link capacities are not exceeded by the sum of the flows routed along arc (i,j). Additional constraints $f_{ij}^{st} \leq \min(\kappa_{ij}, D(s,t))v_{ij}$ are imposed to ensure that the flow sent along each arc (i,j) is null when the corresponding link is not installed. The other constraints are identical.

The complexity of these formulations comes from the multiplication of the continuous flow variables f_{ij}^{st} with the continuous traffic monitoring fraction variables y_{ij}^{st}. The problem can nevertheless be resolved by generating the flow variables out of the flow routing model and by projecting them in the monitoring cost minimization problem. A more elaborated method involves composite objectives, i.e., by generating a first model using an unlimited monitoring cost or an approximated monitoring cost function and resolve the problem under these conditions. For instance, by considering that the monitoring load would be proportional to the fraction k multiplied by the capacity of each arc (i,j) or the projected load of each arc (i,j). We can then generate a second model but now by using the exact monitoring function where traffic flows are parametrized and solve the minimum monitoring cost problem given the minimum routing cost obtained from the previous iteration. On the other hand, we can also generate the model by replacing in the second term of (2.15) the flow variables f_{ij}^{st} by a binary flow variable $b_{ij}^{st} \in \{0,1\}$ multiplied by the actual value of the traffic demand provided that network flow routing decision is limited to a single next-hop (no load balancing of traffic flows). To meet this condition, the formulation of Fig. 2.70 needs to be extended with additional routing variables $r_{ij}^{t} \in \{0,1\}\ \forall (i,j) \in A, t \in V$ verifying the additional constraints $\sum_{j:(i,j)\in A} r_{ij}^{t} = 1\ \forall i,t \in V, i \neq t$. For the MCND formulation, the capacity constraints $f_{ij}^{st} \leq \min(\kappa_{ij}, D(s,t))v_{ij}$ have also to be adapted by replacing the binary installation variables v_{ij} with the routing variables r_{ij}^{t} such that the flows directed to destination t are routed along arc (i,j) only if this arc is included in node i routing table for that destination. This modification comes together with the addition of the constraints $r_{ij}^{t} \leq v_{ij}\ \forall (i,j) \in A, t \in V$. For the MMCF formulation, an auxiliary binary variable is introduced to perform the equivalent transformation for the constraints (2.17) as the routing decision binary variables r_{ij}^{t} are defined per destination t. Following these changes, one can then replace in (2.15) the resulting product $b_{ij}^{st} y_{ij}^{st}$ by a single

$$\max \sum_{s,t \in V} u^{st}\Big(\sum_{(i,j)\in A} y_{ij}^{st}\Big) \tag{2.24}$$

subject to:

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \leq \mathcal{M} \tag{2.25}$$

$$\sum_{(i,j)\in A} m_{ij} \sum_{s,t\in V} \varphi_{ij}^{st} y_{ij}^{st} \leq \mathcal{N} \tag{2.26}$$

$$y_{ij}^{st} \leq x_{ij} \qquad (i,j) \in A, s,t \in V \tag{2.27}$$

$$\sum_{s,t\in V} \varphi_{ij}^{st} y_{ij}^{st} \leq b_{ij} x_{ij} \qquad (i,j) \in A \tag{2.28}$$

$$\sum_{(i,j)\in A} y_{ij}^{st} = k \qquad s,t \in V \tag{2.29}$$

$$y_{ij}^{st} \geq 0.01 \qquad (i,j) \in A, s,t \in V \tag{2.30}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{2.31}$$

Figure 2.71: Mixed integer programming formulation for utility maximization problem

continuous variable $\gamma_{ij}^{st} \geq 0$ (since the bounds of the continuous monitoring fraction variables are known: $0 \leq y_{ij}^{st} \leq k$) together with the additional constraints $\gamma_{ij}^{st} \leq k\ b_{ij}^{st}$, $\gamma_{ij}^{st} \geq 0$, $\gamma_{ij}^{st} \leq y_{ij}^{st}$, and $\gamma_{ij}^{st} \geq y_{ij}^{st} - k\ (1 - b_{ij}^{st})$. The same transformation can be applied to the monitoring capacity constraints (2.19) when solving the capacitive version of the monitoring placement and configuration problem.

2.12.1.2 Utility Maximization

The problem consists in placing and configuring a set of monitoring points such as to maximize the utility of monitoring traffic flows without violating the budget constraint imposed on the total monitoring cost. In addition to the input considered for the cost minimization problem documented in Sec. 2.12.1.1, we assume that a fixed monitoring installation budget $\mathcal{M}$ and a configuration budget $\mathcal{N}$ are known. The modeling of this problem involves the binary variables x_{ij} and continuous variables y_{ij}^{st}. The objective is to maximize the sum of the utilities of monitoring individual traffic flows without violating the budget constraints on the installation and configuration cost. For this purpose, we assume that the utility function u associated to each flow is non-decreasing (increasing the monitoring fraction improves utility) and of the form $\alpha - \exp(-\beta \sum_{(i,j)\in A} y_{ij}^{st})$ where, $\alpha, \beta \in \mathbb{R}_0^+$. Thus, after reaching a certain threshold, increasing the monitored fraction of traffic yields relatively less benefit. For computational purposes, we approximate this concave continuous function by means of piecewise-linear continuous fit (see [37]). Assuming that the flow variables have been obtained by resolving the flow routing problem, we can then formulate the utility maximization problem as described in Fig. 2.71 where the traffic flows appear as parameters φ_{ij}^{st} in the constraints.

2.12.2 Numerical Experiments

2.12.2.1 Implementation and Problem Instances

The formulations developed in Sections 2.12.1.1 and 2.12.1.2 have been implemented using IBM ILOG OPL modeling language and solved with CPLEX 12.6. Their execution has been performed on a computer equipped with 8 Intel Xeon quad-core processors and 512GB of DDR3 RAM. The evaluation of the formulations has been realized on a set of topologies documented in the SNDlib library [77] including links capacity, links installation cost and traffic demands. Table 2.14 summarizes the properties of the topologies considered. We set the derivative of the cost function $\phi(\kappa_{ij}, l_{ij})$ associated with each arc $(i, j) \in A$ such that $\phi(\kappa_{ij}, 0) = 0$ similarly to [78].

Table 2.14: Network Topologies and Properties

Topology	Nodes	Links	Min, Max, Avg Degree	Diameter
france	25	45	2;10;3.60	8
geant	22	36	2;8;3.27	5
germany50	50	88	2;5;3.52	9
india35	35	80	2;9;4.57	7
norway	27	51	2;6;3.78	7

2.12.2.2 Simulation Results and Analysis

Table 2.15 summarizes the results obtained for the monitoring cost minimization problem formulated in Fig. 2.70 applied to the topologies listed in Table 2.14. We set the fraction k from 0,1 to 0,9 with steps of 0,1. For each routing strategy (MCF, MMCF or MCND), the first column (C_I) lists the monitoring point installation cost, the second (C_C) the monitoring point configuration cost, the third (C_{Tot}) the total monitoring cost, the fourth (N_m) the number of monitoring points and the fifth column (L_{avg}) the cumulative monitored fraction of traffic averaged per monitor. From these results we can observe that the MCND strategy leads to the fewer number of installed monitoring points. The gain ranges from a factor 2 for *germany50* to 40% for *india35* compared to the MCF strategy. It is also interesting to observe that monitoring traffic flows individually with the objective of obtaining a given fraction k leads to higher cost than monitoring the corresponding fraction of traffic imperceptibly (out of the total amount of traffic or load per arc).

Table 2.16 summarizes the results obtained for the monitoring utility maximization problem formulated in Fig. 2.71 applied to the topology listed in Table 2.14. For each execution running up to 3600s, we increase the total monitoring cost and determine the reward obtained while maximizing the total fraction of monitored traffic implying an inequality in constraints (2.30) instead of an equality. For this purpose, we add the following constraints in the formulation $\mathcal{M} + \mathcal{N} \leq T_{\text{cost}}$ and give the total cost as input to the execution. Hence, the fraction k does not apply equally anymore to each traffic flow; otherwise, the results would be less informative. In the utility function, we set the parameters $\alpha = 1$ and $\beta = 6.3$. In addition to the installation cost (C_I), configuration cost (C_C) and total cost (C_{Tot}) reported in Table 2.16 for each routing strategy, the column k lists the average monitoring fraction for each traffic flow and the column u the monitoring utility. Following these results, we can observe that CPLEX efficiency decreases as the size of the instances increases. For the smaller instances, i.e., *france*, *geant* and *norway*, CPLEX provides a solution with an optimality gap of about 15% for the higher values of the total monitoring cost. For the larger instances, i.e., *germany50*, and *india35* this gap increases up to reaching the situation where CPLEX does not pro-

vide any acceptable solution for some of the input cost values (as indicated by the entries marked with $-$). Obviously, one could increase the computation time to lower this gap though these results are indicative of the computational behavior trend. For *norway*, we observe an interesting phenomena where the configuration cost may be decreasing as the installation cost increases, providing an instance where installation vs. configuration cost tradeoff is observable. From a computational perspective, our formulation involves a number of constraints which grows cubically with the number of network nodes; their resolution reaches the limits of state-of-the-art MIP solvers such as CPLEX. More efficient resolution methods would be thus required to cope with the combinatorial explosion of the monitoring utility maximization problem.

2.12.3 Conclusion

In this section, we propose a first model the combined (capacitated) monitoring point placement and configuration - multicommodity network flow optimization problem and present a mixed integer programming formulation. Simulation results obtained by transforming the original nonlinear formulation shows that mixed-integer programming software, such as CPLEX allows to solve medium-scale instances but as the size of the instances increases, their efficiency decreases. The proposed formulation can also be dualized to determine the monitoring utility gain when varying the total budget constraint on the monitoring cost. As part of our future work in determining limits of centralized problem solving, we will extend the proposed formulation to multiple time period problems (instead of a single period), resolve its robust counterpart to account for the uncertainty in the traffic demands (which may compromise feasibility and optimality of the solution), and specify uncertainty sets by means of data-driven statistical learning methods.

Table 2.15: Numerical Results

k	MCF					MMCF					MCND				
	C_I	C_C	C_{Tot}	N_m	L_{avg}	C_I	C_C	C_{Tot}	N_m	L_{avg}	C_I	C_C	C_{Tot}	N_m	L_{avg}
France															
0.1	11400	2940	14340	57	0.52	11000	2424	13424	55	0.54	10200	2253	12453	51	0.59
0.2	11400	5881	17281	57	1.05	11000	4792	15792	55	1.09	10200	4420	14620	51	1.18
0.3	11400	8821	20221	57	1.58	11000	7159	18159	55	1.64	10200	6588	16788	51	1.76
0.4	11400	11762	23162	57	2.10	11000	9526	20526	55	2.18	10200	8755	18955	51	2.35
0.5	11400	14702	26102	57	2.63	10000	11893	22893	55	2.73	10200	10922	21122	51	2.94
0.6	11400	17643	29043	57	3.16	11000	14260	25260	55	3.27	10400	12873	23273	52	3.46
0.7	11400	20583	31983	57	3.68	11000	16627	27267	55	3.82	10400	15004	25404	52	4.04
0.8	11400	23524	34924	57	4.21	11000	18994	29994	55	4.36	10400	17134	27534	52	4.61
0.9	11400	26465	37865	57	4.74	11000	21361	32261	55	4.91	10400	19264	29664	52	5.19
Geant															
0.1	75922	28898	104820	72	0.64	75140	21842	96982	70	0.66	64437	19553	83990	63	0.73
0.2	75922	57131	133053	72	1.28	75140	41116	116256	70	1.32	64437	35933	100370	63	1.47
0.3	75922	85364	161286	72	1.92	75140	60390	135530	70	1.98	64437	52313	116750	63	2.20
0.4	75922	113596	189518	72	2.57	75140	79664	154805	70	2.64	64437	68693	133130	63	2.93
0.5	75922	141829	217751	72	3.21	75140	98398	174079	70	3.30	64437	85072	149509	63	3.67
0.6	75922	170062	245984	72	3.85	75140	118213	193353	70	3.96	64437	101452	165889	63	4.40
0.7	75922	198295	274217	72	4.49	75140	137487	212627	70	4.62	64437	117832	182269	63	5.13
0.8	75922	226528	302450	72	5.13	75140	156761	231901	70	5.28	64437	134212	198649	63	5.87
0.9	75922	254761	330683	72	5.77	75140	176035	251175	70	5.94	64437	150591	215028	63	6.60
Germany50															
0.1	314180	23	314203	87	0.76	310460	23	310483	86	0.77	310460	21	310481	86	0.77
0.2	314180	45	314225	87	1.52	310460	45	314225	86	1.52	310460	42	310502	86	1.54
0.3	314180	68	314248	87	2.28	310460	68	310528	86	2.31	310460	62	310522	86	2.31
0.4	314180	91	314271	87	3.04	310460	91	310551	86	3.08	310460	83	310543	86	3.08
0.5	314180	114	314294	87	3.80	310460	114	310574	86	3.85	310460	104	310564	86	3.84
0.6	314180	136	314316	87	4.56	310460	136	310596	86	4.62	310460	125	310585	86	4.62
0.7	314180	159	314339	87	5.33	310460	159	310619	86	5.39	310460	145	310605	86	5.39
0.8	314180	182	314362	87	6.09	310460	182	310642	86	6.16	310460	166	310626	86	6.16
0.9	314180	205	314385	87	6.85	310460	205	310665	86	6.93	310460	187	310467	86	6.93
India35															
0.1	15560	299	15859	80	0.74	15560	268	15828	80	0.74	9160	294	9454	48	1.24
0.2	15560	598	16158	80	1.49	15560	529	16089	80	1.49	9160	586	9746	48	2.48
0.3	15560	898	16458	80	2.23	15560	791	16351	80	2.23	9160	879	10038	48	3.72
0.4	15560	1197	16757	80	2.98	15560	1052	16612	80	2.98	9160	1171	10331	48	4.96
0.5	15560	1496	17056	80	3.72	15560	1313	16873	80	3.72	9160	1464	10624	48	6.20
0.6	15560	1795	17335	80	4.46	15560	1574	17135	80	4.46	9160	1756	10916	48	7.44
0.7	15560	2095	17655	80	5.21	15560	1836	17396	80	5.21	9160	2049	11209	48	8.68
0.8	15560	2394	17954	80	5.95	15560	2097	17657	80	5.95	9160	2341	11501	48	9.92
0.9	15560	2693	18253	80	6.69	15560	2358	17918	80	6.69	9160	2634	11794	48	11.16
Norway															
0.1	422598	530	423128	102	0.69										
0.2	422598	1061	423659	102	1.38	401192	933	402125	97	1.45	327259	1006	328265	80	1.75
0.3	422598	1591	424189	102	2.06	401192	1385	402577	97	2.17	327259	1503	328762	80	2.63
0.4	422598	2121	424719	102	2.75	401192	1837	403029	97	2.90	327259	2000	329259	80	3.51
0.5	422598	2652	425250	102	3.44	401192	2290	403482	97	3.62	327259	2496	329755	80	4.39
0.6	422598	3182	425780	102	4.13	401192	2742	403934	97	4.34	327259	2993	330252	80	5.26
0.7	422598	3713	426311	102	4.82	401192	3194	404386	97	5.06	327259	3490	330749	80	6.14
0.8	422598	4243	426841	102	5.51	401192	3646	404838	97	5.79	327259	3987	331246	80	7.02
0.9	422598	4773	427371	102	6.19	401192	4099	405291	97	6.51	327259	4484	331743	80	7.90

Table 2.16: Numerical Results

Step	MCF					MMCF					MCND				
	k	C_I	C_C	C_{Tot}	N_m	u	k	C_I	C_C	C_{Tot}	N_m	u	k	C_I	C_C	C_{Tot}	N_m	u
France																		
1	0.34	1800	6118	7918	9	805	0.33	1600	6186	7786	8	786	0.31	1400	5567	6966	7	732
2	0.54	3200	13164	16364	16	1260	0.52	2800	13210	16010	14	1222	0.49	2600	11049	13649	13	1167
3	0.66	4400	19881	24281	22	1556	0.65	3600	20581	24181	18	1527	0.62	3600	17648	21248	18	1461
4	0.76	5600	26451	32051	28	1786	0.74	4600	28123	32723	23	1739	0.71	4800	23189	27989	24	1675
5	0.82	7400	30938	38338	37	1931	0.81	5600	35258	40858	28	1905	0.79	5800	29381	35181	29	1857
6	0.87	8800	36903	45703	44	2053	0.86	7200	37976	45176	36	2025	0.85	6600	35326	41926	33	1999
7	0.91	9400	45084	54484	47	2153	0.90	8400	43454	51854	42	2127	0.89	8000	38792	46792	40	2106
8	0.95	10600	49445	60045	53	2232	0.94	9400	49086	58486	47	2210	0.93	9000	42949	51949	45	2193
9	0.97	11000	57977	68977	55	2293	0.97	10200	52884	63084	51	2278	0.96	9000	52559	61559	45	2270
10	0.99	11200	65209	76409	56	2335	0.99	10800	57918	68718	54	2329	0.99	10000	55848	65848	50	2333
Geant																		
1	0.68	15241	45361	60602	33	2484	0.64	11606	42102	53708	26	2317	0.64	10000	37443	47443	25	2316
2	0.81	24010	101537	125547	46	2965	0.78	19983	95005	114988	39	2826	0.78	15935	87102	103037	34	2842
3	0.88	35160	147857	183017	55	3212	0.85	25436	144889	170235	46	3088	0.86	24077	130128	154205	42	3108
4	0.92	38025	207582	245607	58	3354	0.89	32588	205041	237629	54	3231	0.90	30468	163493	193961	48	3278
5	0.94	57743	228615	286358	64	3435	0.93	38136	266183	304319	56	3361	0.93	34275	233199	267474	52	3352
6	0.96	60683	281741	342424	67	3498	0.95	43799	330343	374142	58	3447	0.95	37740	270469	308209	54	3461
7	0.98	63281	354481	417762	69	3545	0.97	54017	363003	417020	63	3506	0.97	48642	292056	340698	57	3513
8	0.98	75658	396485	472143	71	3578	0.98	57188	392876	450064	67	3551	0.98	49462	371619	421081	59	3556
9	0.99	75922	463759	539681	72	3603	0.99	69565	429227	498792	69	3586	0.99	57635	398930	456565	62	3589
10	0.99	75922	577350	653272	72	3620	0.99	75140	466867	542007	70	3612	0.99	64437	428121	492558	63	3614
Germany50																		
1	0.51	31330	426	31756	9	2685	0.51	30900	411	31311	9	2662	0.49	30900	388	31288	9	2536
2	0.71	62660	537	63197	18	3698	0.71	61800	512	62352	18	3675	0.69	602320	602	60832	17	3620
3	0.81	93990	621	94611	27	4206	0.80	90270	579	90489	26	4177	—	—	—	—	—	—
4	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
5	—	—	—	—	—	—	0.91	154800	662	155462	43	4768	—	—	—	—	—	—
6	0.94	186560	671	187231	52	4889	0.94	186130	668	186797	52	4912	0.94	186130	710	186840	52	4899
7	0.96	217890	676	218566	61	4987	0.96	216600	675	217276	61	4989	0.96	215460	656	216117	60	4982
8	0.97	249650	678	250329	70	5056	0.97	248080	673	248754	69	5054	0.97	248080	665	248475	69	5056
9	0.99	282700	686	283386	79	5135	0.98	278980	682	279662	78	5124	—	—	—	—	—	—
10	—	—	—	—	—	—	0.99	309600	689	310289	86	5186	0.99	309170	666	309836	86	5189
India35																		
1	0.31	1280	2007	3287	7	1438	0.31	1310	2059	3369	8	1428	0.25	750	1811	2561	4	1167
2	0.51	2400	6784	6784	13	2399	0.52	2590	4059	6649	15	2425	0.49	1620	2956	4576	9	2090
3	0.66	3720	5835	9555	19	3099	0.66	3900	5486	9386	23	3106	0.60	2490	4006	6496	13	2827
4	0.77	5140	7025	12165	27	3581	—	—	—	—	—	—	—	—	—	—	—	—
5	0.85	6710	7564	14274	36	3954	0.85	6280	7216	14036	37	3972	0.82	3910	6892	10802	21	3851
6	0.90	8250	8331	16581	44	4207	0.91	8430	7539	15970	45	4233	0.89	4780	7783	12563	26	4176
7	0.94	9940	8510	18450	52	4390	0.95	10070	7826	17896	53	4418	0.95	5590	9051	14641	30	4423
8	0.96	11700	8616	20316	60	4502	0.97	11810	7891	19701	63	4523	0.97	6610	9394	16004	36	4548
9	0.98	13450	8682	22132	71	4590	0.98	13530	7983	21513	72	4610	0.98	7710	9508	17218	42	4613
10	0.99	15270	8703	23973	79	4658	0.99	15520	7939	23189	79	4659	0.99	8840	9790	18630	47	4656
Norway																		
1	0.61	39594	12294	51888	15	3363	0.64	37484	12749	50233	19	3509	0.63	29950	13072	43022	17	3494
2	0.77	81493	14486	95979	29	4269	0.80	77504	13908	91412	32	4410	0.77	62765	14254	77019	28	4195
3	0.86	123646	15738	139382	39	4752	0.88	117227	16929	134152	44	4840	0.89	94786	17853	112639	35	4914
4	0.92	165848	17436	183284	52	5067	0.92	157533	18183	175716	51	5072	0.93	127818	17860	145678	43	5136
5	0.94	208356	13352	226708	63	5199	0.94	178011	17565	235576	61	5204	0.95	160881	18903	178974	51	5263
6	0.96	251313	16783	268096	74	5285	0.96	238069	20920	258989	71	5294	0.96	193886	18773	212659	59	5341
7	0.97	293739	18141	311880	83	5358	0.97	277807	27026	304833	78	5360	0.98	227224	17559	244783	66	5398
8	0.98	336650	16299	353579	90	5413	0.98	318506	25293	343799	85	5415	0.99	255362	47694	303056	71	5439
9	0.99	379291	17363	396654	97	5467	0.99	360132	16269	376395	91	5462	0.99	290411	26894	317305	76	5476
10	0.99	422063	17055	439118	101	5497	0.99	400657	16138	416795	96	5497	0.99	326724	17359	344083	79	5497

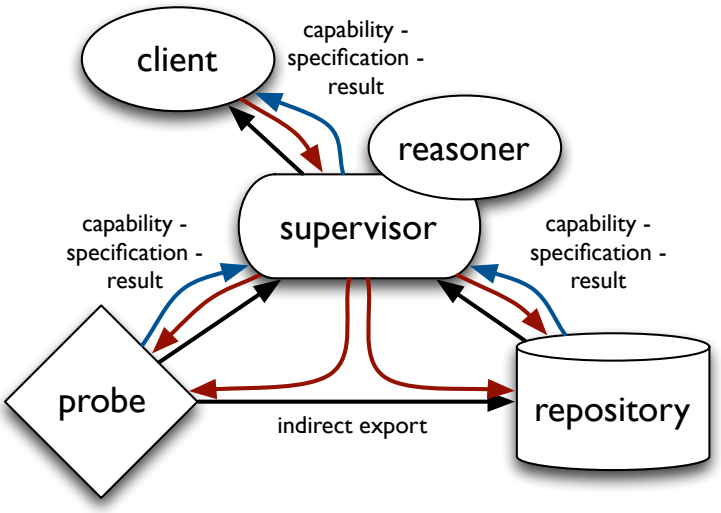


Figure 3.1: The mPlane architecture. The Reasoner coordinates the measurements and the analysis performed by probes and repositories, actuating through the Supervisor

3 Use-Case Workflow

In this chapter, we provide a description of the mPlane workflow for each use case described in Chapter 2. In particular, we describe how each mPlane component acts with the use case.

Fig. 3.1 recalls the architecture of the mPlane. The Reasoner coordinates the measurements and the analysis performed by Probes and Repositories, actuating through the Supervisor. It is responsible for the orchestration of the iterative analysis and the correlation of the results exposed by the analysis modules. Such a reasoning-based system is capable of generating conclusions and triggering further measurements to provide more accurate and detailed insights regarding the supported traffic monitoring and analysis applications. As such, the reasoner offers the necessary adaptability and smartness of the mPlane to find the proper high-level yet accurate explanations to the problems under analysis in the different use cases.

The Reasoner has different specific roles, depending on the use case to tackle (Table 3.1 maps each analysis module to the corresponding use case). 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.

3.1 Supporting DaaS Troubleshooting

Fig. 3.2 outlines the complete workflow related to the *Supporting DaaS troubleshooting* the use-case.

		Use Case					
		DaaS Troubleshooting	Content Popularity	Content Curation	QoE Web	Mobile Network RCA	Anomaly Detection
Analysis Module	HTTP traffic classification		*	*			*
	YouTube QoE					*	*
	Web QoE				*	*	
	DaaS QoE	*					
	Statistical Anomaly Detection				*		*
	Entropy-based Analysis						*
	Statistical Traffic Classification	*					
	Content Classification		*	*			
	URL-based Analyser		*	*	*		
	Forecasting Algorithms		*				
	Decision Trees	*				*	*
	Rule Mining						*
	Sub-Space Clustering				*	*	*
	Probe Geolocation	*			*		*
	Prediction of Unmeasured Paths	*			*	*	*
	Topology Discovery	*				*	
	Anycast Detection				*	*	*

Table 3.1: Analysis modules mapped to use cases

At first, Probes continuously monitor thin-client connections and passively collects IP-level features that can be accessed from the thin-client connection while it is running, such as packet size, rate, inter-arrival time, and TCP-level features such as payload length and number of observed packets, whether they carry data or acknowledge only, TCP flags, etc. These features are collected on a per-connection basis, i.e., on a per-thin client basis, and within sliding observation time-window.

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

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

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,

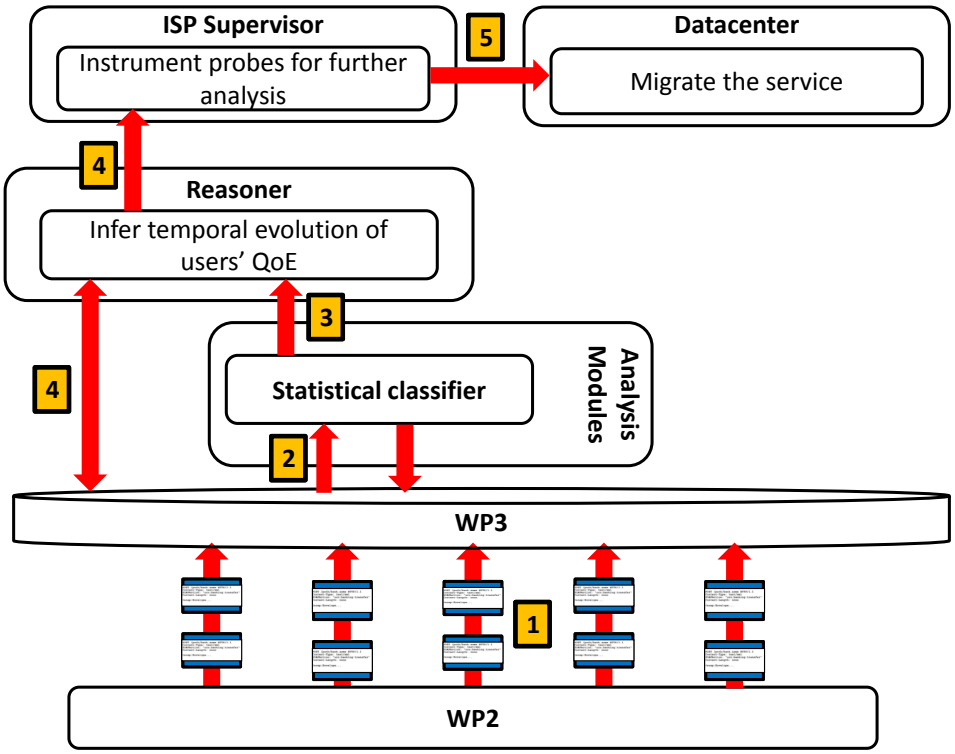


Figure 3.2: Desktop-as-a-Service troubleshooting: workflow. Probes monitor thin-client connections and passively collects IP-level features, which are stored at the Repository (step 1). The Analysis module is responsible for classifying the connection (step 2) and the Reasoner combines the information from the Analysis module with the network conditions along the path (step 3). In case of a poor QoE, the Reasoner, through the Supervisor, instruments probes to run further analysis, and get historical data from the Repository (step 4). Once detected the issue, the Supervisor solves it by triggering the migration of the service or the change of path for the flow (step 5)

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.

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

In case the result of the measurements returns that the bottleneck is due to a node along the path, the Reasoner tries to circumvent the responsible node by migrating the remote server to another datacenter. Alternatively, if the bottleneck is due to the remote server, the Reasoner triggers the migration of such server within the same datacenter, to offload the machine where the service is

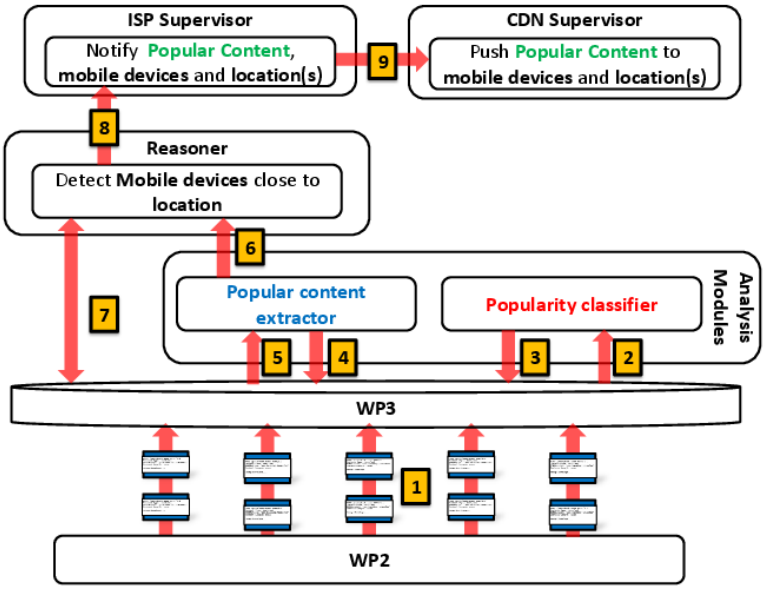


Figure 3.3: The analysis and reasoning modules composing the WP4 layer for the "Estimating Content and Service Popularity for Network Optimization". Given a set of users' request for videos, the Reasoner observes as input the evolution of the popularity of videos over time, and triggers actions based on how the popularity is predicted to evolve

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.

3.2 Estimating Content and Service Popularity for Network Optimization

We present in this section an updated version of the complete workflow for this use-case that we described in the private deliverable D4.2 [73].

Fig. 3.3 outlines the role of each mPlane's component, and more specifically, the WP4 Analysis modules and their interactions with the Repository and the Reasoner. At first, Probes located in different points of the network continuously collect information about the requests of the users, and stream requests to the central Repository. For each request we store the associated timestamp and the network location of the Probe (**Step 1**). Based on these features, the Analysis modules are responsible for predicting the future popularity of each requested contents at each part of the network (i.e., the probe location). This task is accomplished by employing two different modules: the first one, named Popularity Classifier, takes as input the popularity history for a content (**Step 2**), it generates a signature for its request arrival process using the Heterogeneous Mixture Modelling technique, and classify such content in a Hierarchical Clustering Structure, that is stored at the repository (**Step 3**).

In parallel, the Online Predictor, for each observed content explores the Hierarchical Clustering

Structure (**Step 4**) to find the popularity pattern which maximizes a likelihood function (**Step 5**). Once the popularity pattern has been found, we use it to predict its future popularity. Thus, if the number of future views overcomes a static threshold N, an event is triggered to the Reasoner (**Step 6**) to notify which contents are becoming popular and where they are. Then, the Reasoner may query the repository to obtain the IDs of mobile devices, base stations and caches that are located within the same area of the probe, and may be interested at prefetching popular contents (**Step 7**).

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

3.3 Passive Content Curation

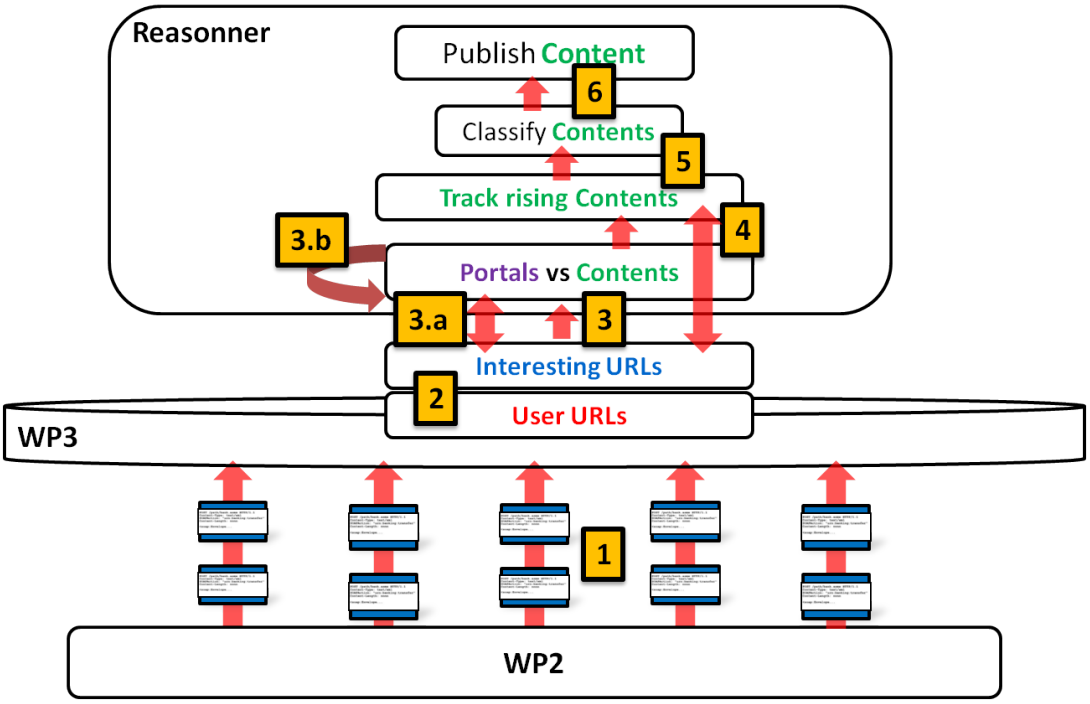


Figure 3.4: The analysis and reasoning modules composing the WP4 layer for the "Passive Content Curation" use-case, and the workflow involving them.

We depict the workflow for this use case in Fig. 3.4. The probes extract HTTP logs from the network and stream them to the repositories (**Step 1**) where online scalable data analysis algorithms run continuously to extract *user* clicks (**Step 2**), infer *interesting* clicks (clicks that are likely to attract other users' attention). The portal vs Contents analysis module of the reasoner continuously pulls interesting URLs to infer clicks that pertain to single *content* items (as opposed to *portals* that aggregate multiple content items) (**Step 3**). As explained above, the reasoner leverages the history of the URLs timeseries to build a knowledge database containing known portals (with very high confidence). This knowledge database is continuously updated as new unknown URLs join the system.

Content-URLs are then notified to the Reasoner through events (**Step 4**). The Reasoner then appends them to the Track rising content module which updates the popularity evolution of each

content-URL (**Step 4**). The detection of a promotable URL creates an event notified to the reasoner, and calls for the content classification analysis module which group contents by categories, e.g., to distinguish videos from news (**Step 5**). Promoted and classified content-URLs are then pushed for publication.

Note that some of our analysis modules (and scalable WP3 data analysis algorithms) might incorporate in the future some of the reasoning tasks done by the reasoner. This is the case for the interesting URLs module as well as the "content versus portal" module. The first module takes as an input user URL visits (user clicks) as elected by the user URLs filtering module. Upon the event of the election of a user URL, we use an active probe (a web scraper to query the page looking for social meta data, in order to understand whether this page is potentially interesting for other users or not. The content versus portal module could be in charge of analyzing the interesting URLs as well as their timeseries (**Step 3.b** and learn in order to detect which URLs pertain to content aggregation URLs (e.g. the front page of an online news website that compiles a lot of news) and which ones relate to a single content item (e.g., a particular news).

3.4 Service Provider Decision Tree for troubleshooting Use Cases

The workflow represented in Fig. 3.5 highlights the interaction between the Analysis module and the Reasoner for the SP decision tree case.

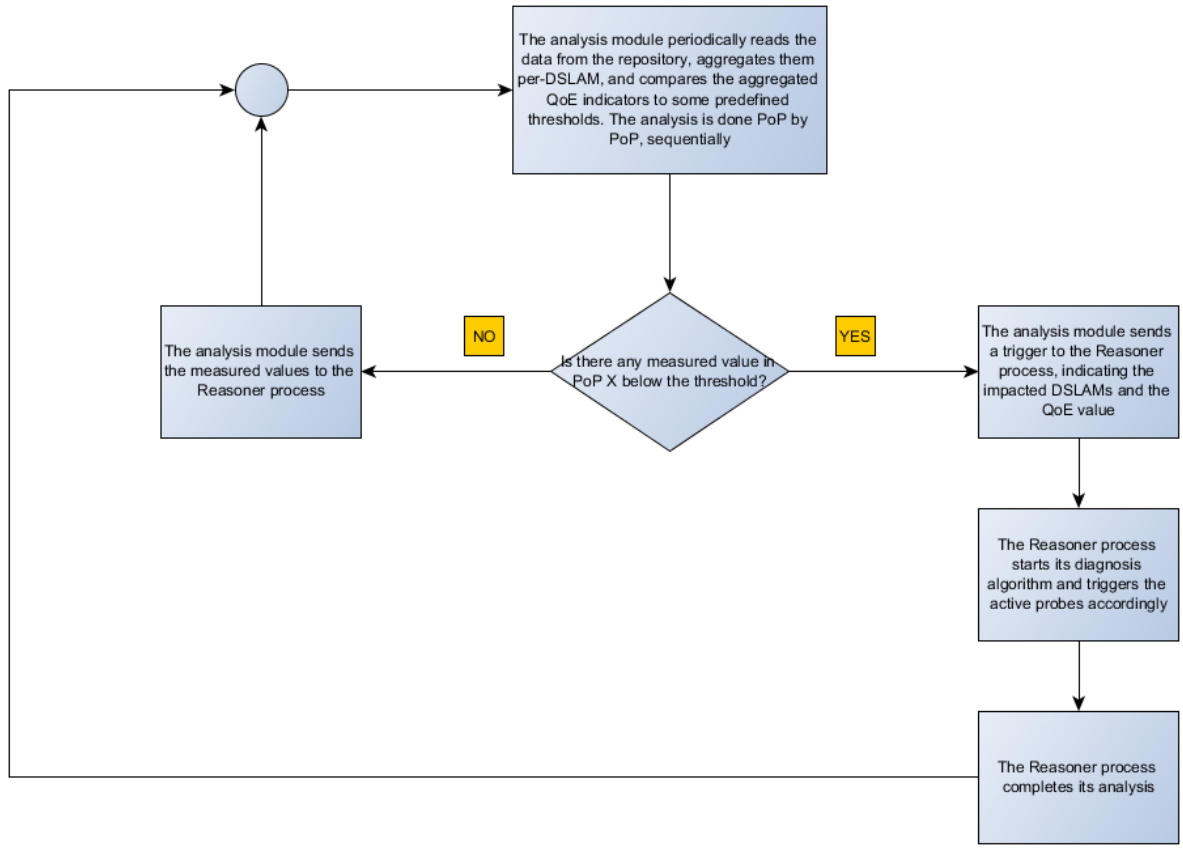


Figure 3.5: Workflow

After the registration phase is completed, the passive Probes start measuring and the results are stored in the Repository. Measured data include the average throughput and Round Trip Time for each flow. The Analysis module periodically reads the measurement results from the Repository and performs an aggregation per-DSLAM. The aggregated QoE value is then compared to some pre-defined thresholds. This is done PoP by PoP, where each PoP has a separate passive probe instance to perform the measurements. After the single PoP has been evaluated, data are sent to the Reasoner process via the Supervisor. Specifically, an "OK" or "NOT OK" is sent, together with all the measured values aggregated per-DSLAM. If the Reasoner receives a "NOT OK", this triggers the start of the diagnosis algorithm, described in deliverable D4.2 [73]. Therefore, the analysis module is responsible for estimating if the perceived quality is over or below the chosen threshold, while the Reasoner is focused on the root cause process to pinpoint the origin of the problem.

3.4.1 Cooperation between different mPlane instancies

The mPlane architecture offers the possibility to envision some forms of cooperation between different mPlane instances. One form of cooperation implies the possibility to ask a different Supervisor for measurement data. For example, an mPlane instance managed by a Service Provider A could run a Reasoner algorithm, as described in Sec. 2.4.2, to pinpoint the root cause of a QoE degradation for traffic from Content Provider X. In case the algorithm suggests that the problem lies outside the SP's network, the Reasoner (through its Supervisor) could ask the Upstream Provider's Supervisor if there's any issue between the Upstream Provider's network and Content Provider X. From a business point, of view this kind of interworking is essentially based on a customer-provider relationship. On the other hand, SP A could also have in place an mPlane relationship with Content Provider X and directly interact with CP's Supervisor. A third kind of mPlane cooperation is between two Service Providers in competition with each other. From a business perspective this could seem unlikely to happen, but it could be figured out a sort of relationship similar to a private peering agreement for BGP route exchange: a sort of "do ut des" relationship where both players have benefits without revealing business sensitive information. Considering the example described above, the Reasoner managed by SP A could benefit from knowing if other "peer" SPs are also experiencing the same problem, given that it has an "external" cause. In any of the previous cases, the relationship between two Supervisors is exactly the same that exists between a probe and a Supervisor. Supervisor B will advertise one or more capabilities to Supervisor A: those capabilities correspond to measurements that Supervisor A can ask for to Supervisor B. These measurements could be "real" measurements triggered on specific probes or they could result from aggregation of data in the Repository. In any case, Supervisor B can mask all sensitive details to Supervisor A and just expose the result it can deliver to the peer. Considering the QoE degradation problem, the capability exposed by Supervisor B to Supervisor A could be the average RTT to the Content Provider, measured every 5 minutes for the last 30 minutes. This could help (the Reasoner linked to) Supervisor A understand if the problem affects also other SPs or not. Fig. 3.6 shows how the cooperation with different Supervisors could fit in the flow chart of Fig. 2.7.

3.5 Quality of Experience for Web browsing

Starting from the workflow described in deliverable D4.2 [73], we depict in another perspective in Fig. 3.7.

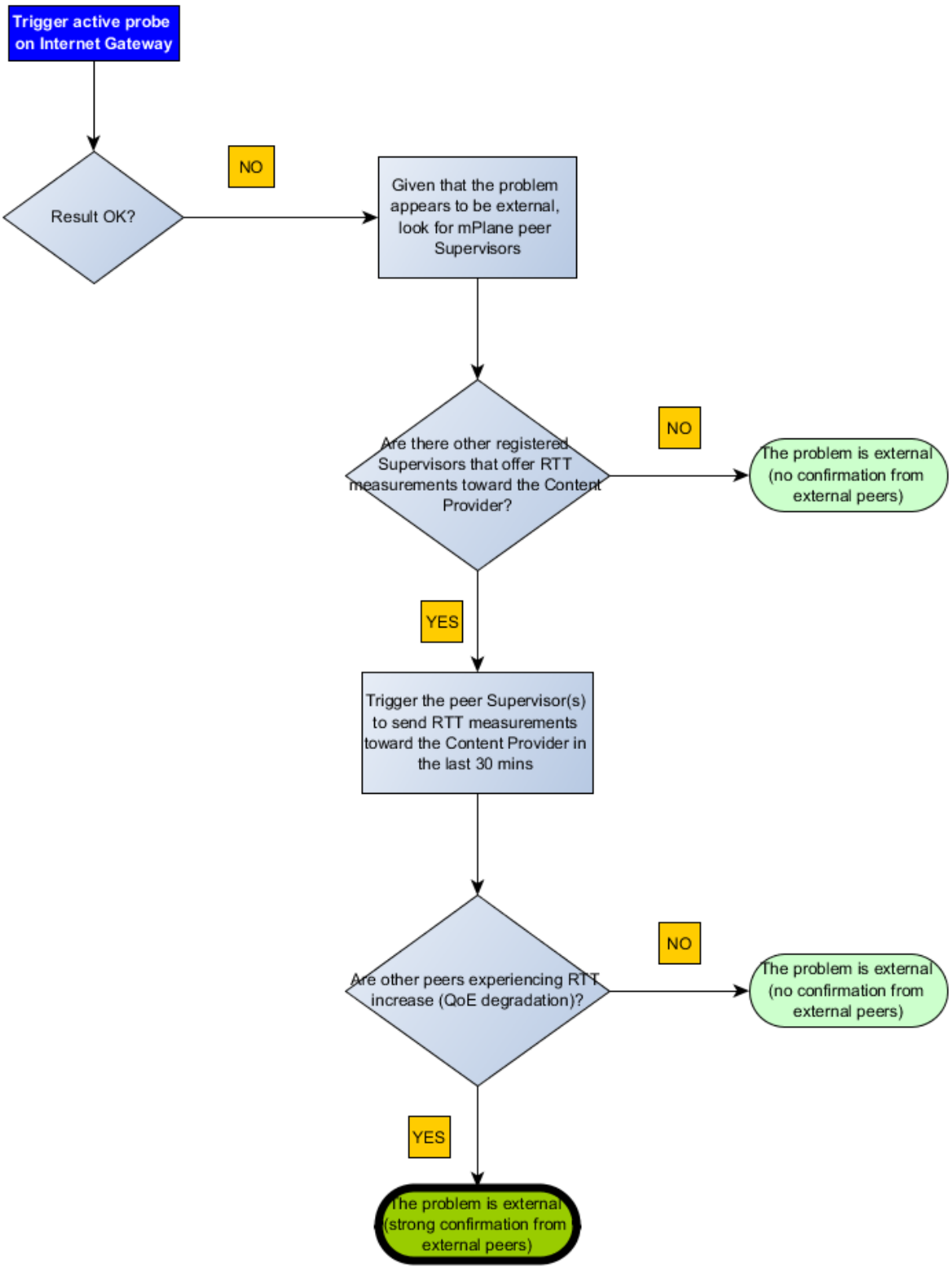


Figure 3.6: Cooperation between different Supervisors to enhance the troubleshooting algorithm

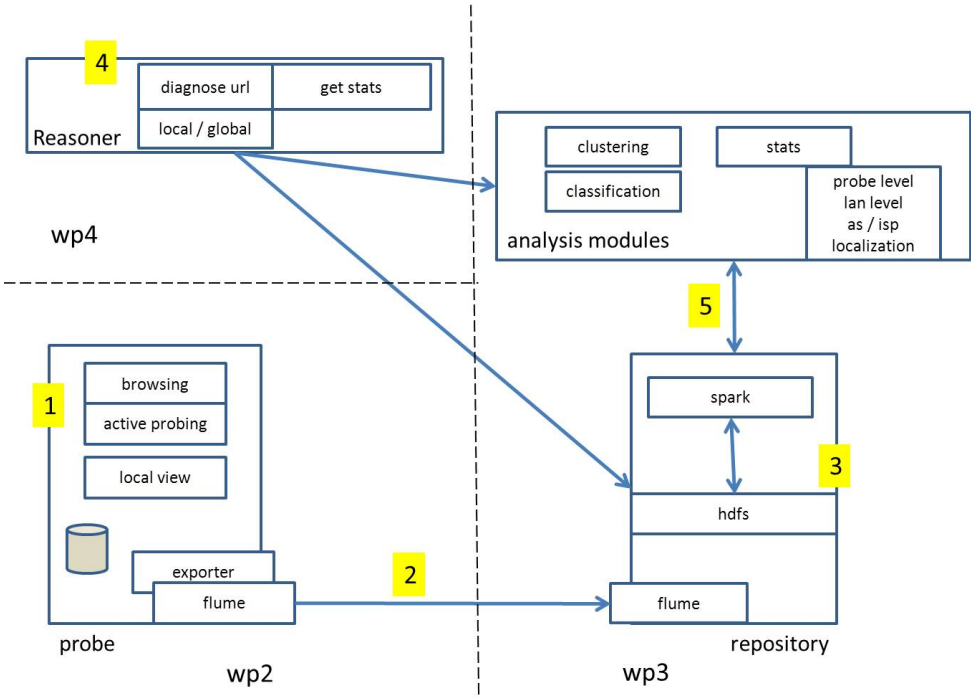


Figure 3.7: The modules for the "Web browsing QOE" use case

From the passive data collected by the instrumented browser on the Probes, equipped with a modified version of Tstat, we extract the involved IP addresses and perform active probing on them by means of `ping` and `traceroute`. Data are collected in a database on the Probe, preprocessed and sent to the Repository (Steps 1-2).

The Flume sink stores data in HDFS, where batch processing modules are carried by Spark, extracting and storing statistical and historical values and distributions of various measurements collected at the Probe side (e.g., time series, RTTs to IP address 'X.X.X.X') (Step 3).

The Repository's capabilities are presented to the Reasoner as tools to perform diagnosis over a poor quality of experience in a web browsing session. When diagnosing a URL (Step 4), the Reasoner can exploit the data provided by the analysis module, or submit further jobs to the Spark engine. The result of the diagnosis is sent both to the client and stored in the Repository (Step 5), for further statistical classification and analysis (e.g, the most common problem when browsing site 'Y').

3.6 Mobile network performance issue cause analysis

In our platform each entity is running an instance of the reasoner, however, each of these instances are considered as a black box.

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

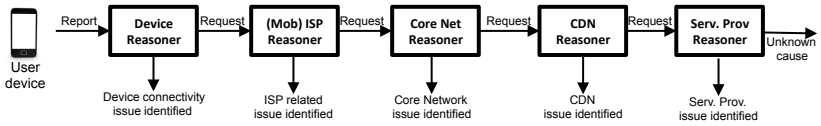


Figure 3.8: The iterative process of consulting with with reasoners across different entities

- Finally, the agents across different organizations are using the proposed architecture to collaborate in order to identify the exact cause of a problem. In that process only the abstracted information is revealed between the involved parties.

- A query to identify an issue is only forwarded to the next entity along the path of the data only when the local data indicate that there is no local problem.

Troubleshooting is usually initiated by the user's device.

For instance, Fig. 3.8 shows an example of a user reported connectivity problem. In this example:

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

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

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

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

In terms of organization of data. The probes store the data in a mongoDB database (WP3). The data are then extracted throught mPlane's Reference Implementation (RI) to the reasoners.

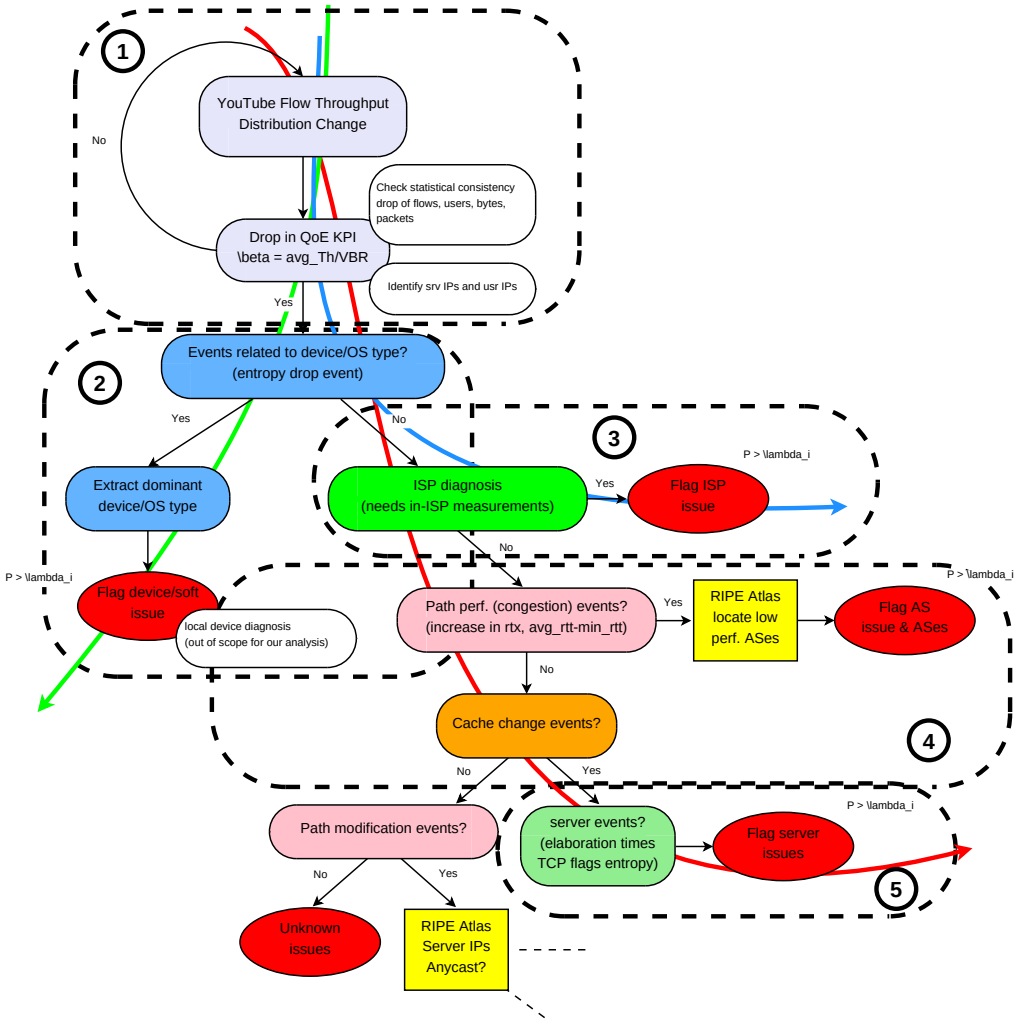


Figure 3.9: Diagnosis graph associated to the detection and troubleshooting support of large-scale QoE-based anomalies in YouTube

3.7 Anomaly Detection and Root Cause Analysis in Large-Scale Networks

As an example of the complete workflow of mPlane in this use case, we present a detection and diagnosis scenario of major anomalies in the delivery of YouTube videos impacting the Quality of Experience (QoE) of a large number of users. This workflow builds on the more detailed analysis of the mPlane reasoner presented in D4.2. For the sake of completeness, Fig. 3.9 depicts the decision graph which guides the complete analysis workflow.

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

analyzed in the iterative process:

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

1. time-series are continuously updated and analyzed for abrupt modifications for the following features: # flows, # bytes, # users, flow throughput, β_2 QoE KPI, empirical entropy of QoE classes (bad, average, excellent), fraction of flows in the lowest QoE class, min RTT, average RTT, server elaboration time, fraction of retransmitted bytes per flow, empirical entropy
2. empirical distribution of average flow throughput per server IP is computed, and analyzed by the anomaly detection algorithm.
3. empirical distributions of number of flows and bytes served per server IP or group of servers sharing the same network prefix (e.g., /24) are computed, and analyzed by the anomaly detection algorithm.

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

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

Note that other events have to be similarly tracked at the different components of the end-to-end service, for example, at the access ISP, both at the access and at the core (related to the internal routing tables, the routers, the aggregation network, the mobile stations, etc.), at the end-devices which are downloading the monitored YouTube flows, etc..

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

1. all traffic flows are analyzed by Tstat at the vantage point, and those belonging to YouTube are exported into DBStream.
2. the anomaly detection algorithm runs continuously on the YouTube flows within DBStream, considering as KPIs the per-flow average download throughput (to detect performance issues) and the number of flows served per /24 CDN subnetwork (to detect Google cache selection changes).
3. when an anomaly is detected as a major shift in the distribution of flows throughput towards lower throughput values, the diagnosis analysis is triggered (part (1) in Fig. 3.9).
4. the first step is to verify if the detected anomaly is statistically consistent, i.e., that it is not caused because of a big drop in the number of samples considered in the empirical distribution computation.

5. then the analysis verifies if this detected anomaly is actually impacting the QoE of the users, by analyzing the β_2 QoE-based KPI. The analysis is not done on the distribution of β_2, but on the average and median values for all the downloaded flows. This is to better spot major QoE anomalies in YouTube.

6. the first diagnosis event to verify is a main drop on the time-series related to the empirical entropy of the operative system type of the devices downloading the captured flows. A drop in the empirical entropy would flag a major concentration on the distribution of the OS type of the devices, indicating a possible relation to the OS type (part (2) in Fig. 3.9).

7. the second diagnosis event to verify corresponds actually to a set of events related to the access ISP (part (3) in Fig. 3.9). We do not detail the specific events to evaluate at the ISP level.

8. the third event to verify is the occurrence of performance degradation in the corresponding end-to-end paths carrying the impaired YouTube flows (part (4) in Fig. 3.9). Events tracked on the time series related to packet re-transmissions, queuing delay, etc. are checked in order to identify path congestion.

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

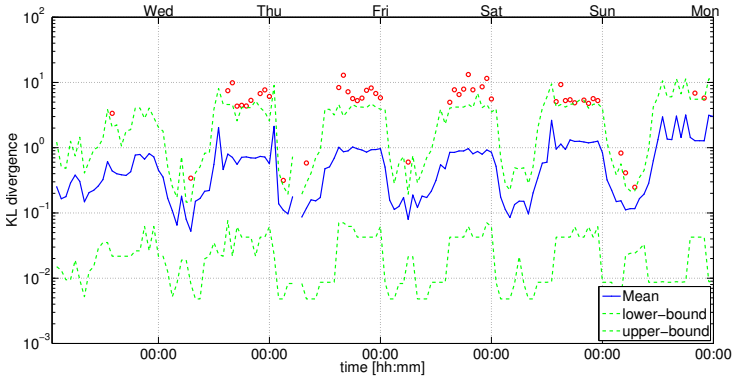
10. if no path performance degradation is observed, the analysis checks for events related to load balancing and cache selection modifications in the Google CDN serving the YouTube flows.

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

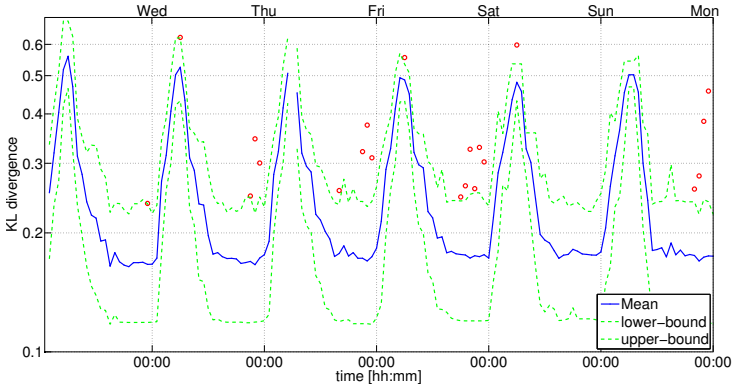
12. if cache selection modifications are present, then the analysis focuses on understanding if the new selected servers are the origin of the problems. For doing so, different application-level KPIs are verified on top of the monitored traffic, such as server elaboration times, TCP flags, etc..

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

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



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



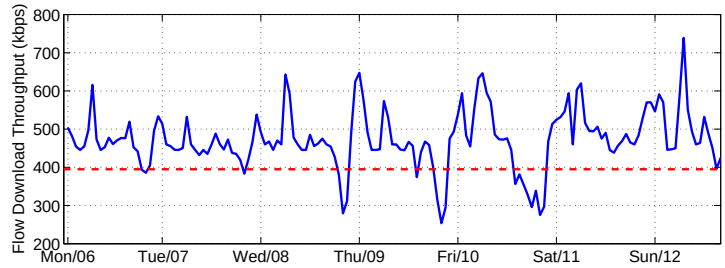
(b) Anomalies in the video flows average download rate across the YouTube users.

Figure 3.10: Detection of anomalies in YouTube traffic. Alarms and acceptance region for the distribution of (a) volume and (b) video flows average download rate. The red markers correspond to the flagged anomalies

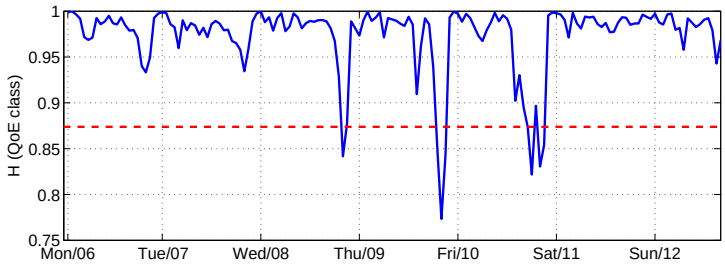
Traffic flows are collected with Tstat at a link of a European fixed-line ISP aggregating 20,000 residential customers who access the Internet through ADSL connections. The complete data spans more than 10M YouTube video flows, served from more than 3,600 Google servers. Using Tstat filtering and classification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

Fig. 3.10 depicts the output of the Anomaly Detection algorithm. Fig. 3.10(a) considers the per Google CDN /24 subnet served volume as the monitored feature. The red markers indicate when an anomaly is detected. From Wednesday the 8th of May onward the algorithm systematically rises alarms from 15:00 to 00:00, which correspond to a change in the cache selection policy, as we shall see next. In addition, Fig. 3.10(b) reports the same information for the average video flows download rate. In this case, the analysis module detects some anomalies only between peak hours (21:00-23:00) from the 8th onward, suggesting that the throughput degradation are linked to high utilization of resources. Let us begin by understanding if the detected drop in the YouTube flow throughput has an impact on the QoE of the end users.

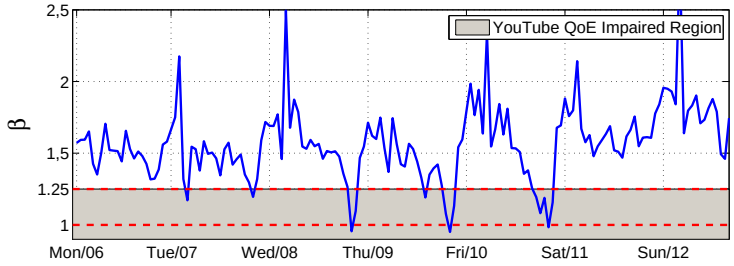
Fig. 3.11 plots the time series of three different performance indicators related to the YouTube



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



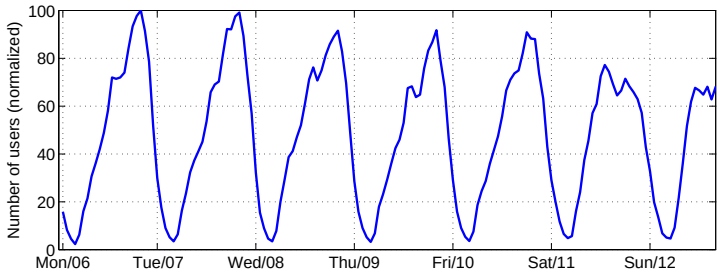
(b) Entropy of QoE classes per hour for all YouTube flows.



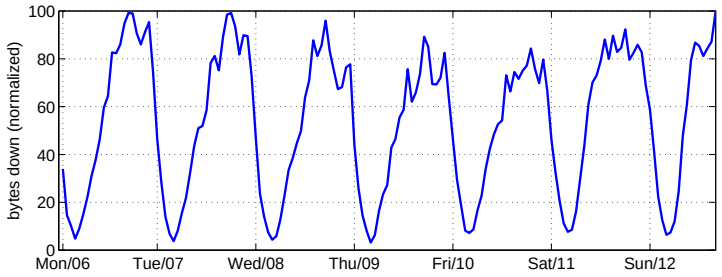
(c) 50%-p of β_2 per hour for all YouTube flows.

Figure 3.11: Detecting the QoE-based anomaly. There is a clear drop in the download flow throughput from Wednesday till Friday at peak-load hours, between 20hs and 23hs. The combined drop in the entropy of the QoE classes and in the KPI β_2 reveal a significant QoE degradation

download performance and to the end-user QoE. Fig. 3.11(a) depicts the median across all YouTube flows of the download flow throughput during the complete week. There is a normal reduction of the throughput on Monday and Tuesday at peak-load time, between 20hs and 23hs. However, from Wednesday on, this drop is significantly higher, and drops way below the bad QoE threshold $T_{h_1} = 400$ kbps, flagging a potential QoE impact to the users. Fig. 3.11(b) plots the entropy of the QoE classes built from thresholds $T_{h_1} = 400$ kbps and $T_{h_2} = 800$ kbps, consisting of bad QoE for flows with average download throughput below T_{h_1}, fair QoE for flows with average download throughput between T_{h_1} and T_{h_2}, and good QoE for flows with average download throughput above T_{h_2}. Recall that these thresholds correspond to the QoE mappings presented in Fig. 2.23, which only cover 360p videos. The drop in the throughput combined with the marked drop in the time series of the QoE classes entropy actually reveals that a major share of the YouTube videos are falling into the bad QoE class. Finally, Fig. 3.11(c) actually confirms that these drops are heavily affecting the users experience, as the time series of the KPI β_2 falls well into the video stalling region, depicted in Fig. 2.24.



(a) Users count (normalized).



(b) Bytes down count (normalized).

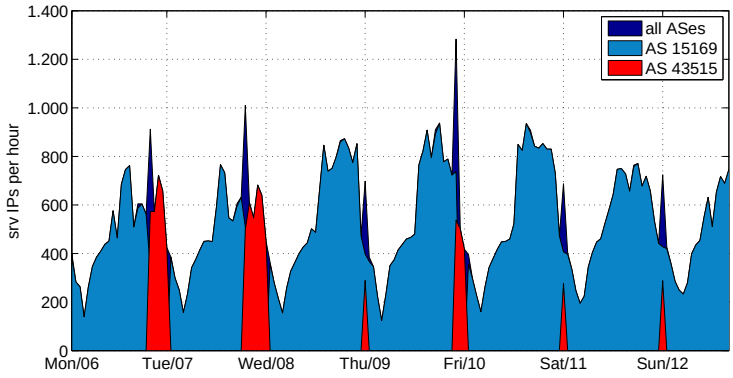
Figure 3.12: Users and bytes down during the week of the anomaly. There are no significant changes during the specific times of the flagged anomaly

3.7.1 Anomaly Diagnosis

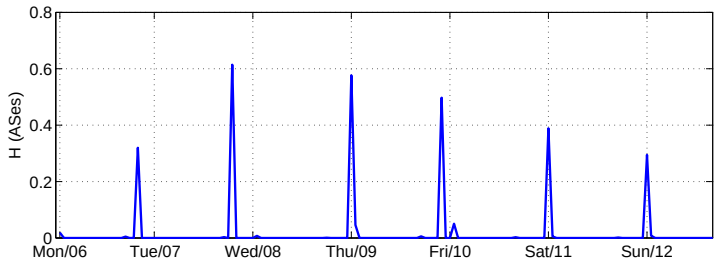
The root causes of the detected anomalies can be multiple: the Google CDN server selection strategies might be choosing wrong servers, the YouTube servers might be overloaded, path changes with much higher RTT from servers to the customers might have occurred, paths might be congested, or there might be problems at the access network. Diagnosing problems at the access network is straightforward for the ISP, as this network belongs to itself. However, diagnosing the problem outside its boundaries is a much more complex task. As we said before, the ISP internal RCA did not identify any problems inside its boundaries, so we focus on the YouTube servers and on the download paths.

Fig. 3.12 depicts the time series of the per hour users and bytes down normalized counts during the analyzed week. While there is a drop in the number of bytes down from Wednesday afternoon on, there are no significant variations on the number of users during the working week (i.e., Monday till Friday), so we can be sure that the throughput and QoE strong variations observed in Fig. 3.11 are not tied to statistical variations of the sample size. Using the results in Fig. 2.24(c), we can say that the drop in the bytes down suggests that the bad QoE affected the users engagement with the video playing, resulting in users dropping the watched videos when multiple stalling occur (i.e., when $\beta < 1.25$).

We study now the YouTube servers selection strategy and the servers providing the videos. Fig. 3.13(a) depicts the number of server IPs providing YouTube flows per hour. As depicted in Fig. 3.13(b), where the entropy of the AS number of the monitored server IPs is presented, there is a sharp shift of servers from AS 15169 to AS 43515 around peak-loud hours. In addition, there is an important reduction on the number of servers selected from AS 43515 on the days of the anomaly. This suggests that a different server selection policy is set up exactly on the same days when the anomalies



(a) IPs per hour hosting YouTube.



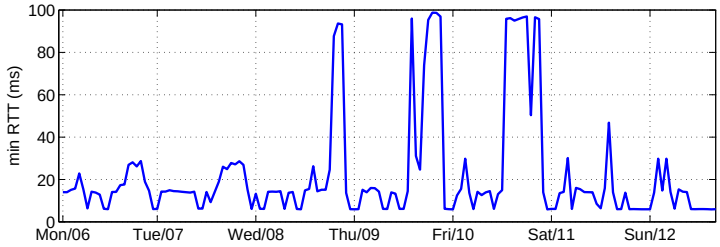
(b) Entropy of the AS numbers time-series.

Figure 3.13: IPs hosting YouTube during the week of the anomaly.

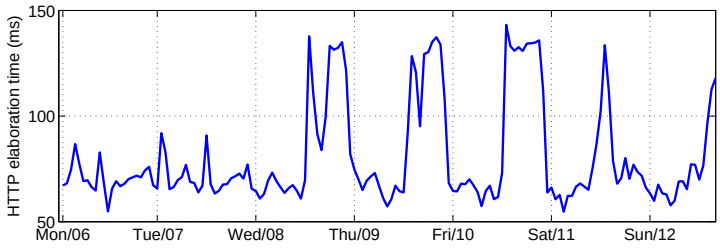
occur.

To further investigate this CDN server selection policy change, Fig. 2.26(a) in Sec. 2.7 shows the TSP of the video volume served by the different IPs in the dataset per hour, aggregated in /24 subnetworks, for 11 consecutive days. The TSP clearly reveals that a different subnet set is used during the second half of the day from the 8th of May on, revealing a different cache selection policy. This change is also visible in the CDFs of the per subnet volume depicted in Fig. 2.26(b). Indeed, we can see that the same set of subnets is used between 00:00 and 15:00 before and after the anomaly, whereas the set used between 15:00 and 00:00 changes after the 8th, when the anomaly occurs.

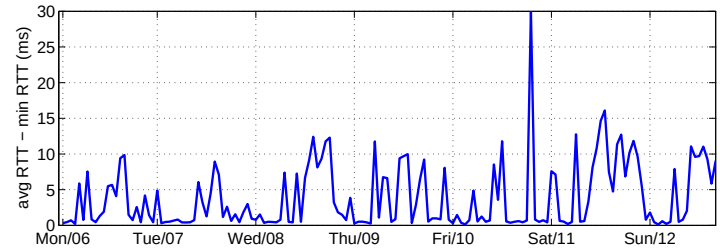
Given this change in the server selection policy, we try to find out if the problem arises from the newly selected servers, or if the problem is located in the path connecting these servers to the users. Fig. 3.14 studies the latency from users to servers during the complete week. Fig. 3.14(a) depicts the median of the min RTT per hour as measured on top of all the YouTube flows. The marked increase in the RTT evidences that the servers selected during the anomaly are much farther than those used before the anomaly. This increase impacts directly on the HTTP elaboration time (i.e., time between HTTP request and reply), as depicted in Fig. 3.14(b). To understand if these latency increases are additionally caused by path congestion, Fig. 3.14(c) plots the time series of the difference between the min RTT and the average RTT values; in a nutshell, in case of strong path congestion, the average RTT shall increase (queuing delay), whereas the min RTT normally keeps constant, as it is directly mapped to the geo-propagation delay. The differences before and during the anomalies do not present significant changes, suggesting that the paths between servers and clients are not suffering from congestion. This is also confirmed by the analysis of the packet retransmissions, which do not present significant variations.



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



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

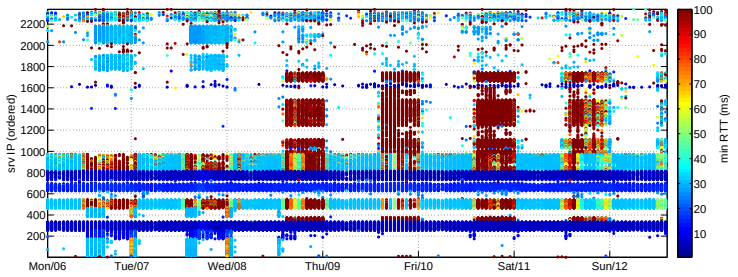


(c) 50%-p of avg RTT - min RTT per hour for all YouTube flows.

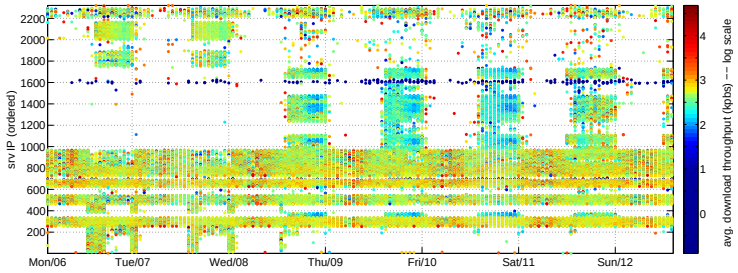
Figure 3.14: The servers selected during the anomaly are much farther than those used before. While there is a marked increase in the server elaboration time, the difference between avg. and min. RTT remains bounded during the anomaly, so we discard the hypothesis of path congestion

The last part of the diagnosis focuses on the YouTube servers. Fig. 3.15 depicts the average (a) min RTT and (b) download flow throughput per server IP in a heatmap like plot. Each row in the plots corresponds to a single server IP. The previously flagged min RTT increase is clearly visible for the new set of IPs which become active from 15:00 to 00:00 from Wednesday on. For those server IPs, Fig. 3.15(b) shows the important throughput drop during peak-load hours. Note however that large min RTT values do not necessary result in lower throughput, as many of the servers used before and during the anomaly are far located but provide high throughput. Fig. 3.16 further studies this drop, comparing the relation between min RTT and average download flow throughput before and during the anomaly. The increase of the min RTT is not the root cause of the anomaly. However, there is a clear cluster of low throughput flows coming from far servers during the peak-load hours.

The conclusion we draw from the diagnosis analysis is that the origin of the anomaly is the cache selection policy applied by Google from Wednesday on, and more specifically, that the additionally selected servers between 15:00 and 00:00 were not correctly dimensioned to handle the traffic load during peaks-hours, between 20:00 and 23:00. This shows that the dynamics of Google's server selection policies might result in poor end-user experience, on the one hand by choosing servers which might not be able to handle the load at specific times, or even by selecting servers without considering the underlying end-to-end path performance.



(a) Average min RTT per server IP.



(b) Average download flow throughput per server IP.

Figure 3.15: There is a new set of server IPs providing YouTube videos from Wednesday on from farther locations. As visible in (b), the average download flow throughput for each of these new server IPs is much lower than the one obtained from other servers

3.8 Verification and Certification of Service Level Agreements

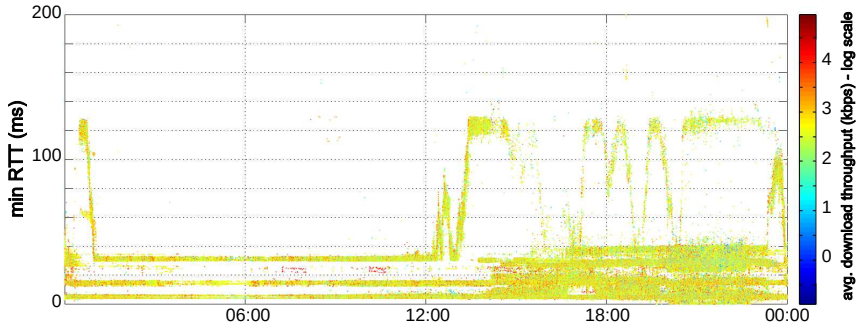
This section presents the workflow of the verification and certification of service level agreement, in short SLA. We determine SLA in terms of:

1. RTT measured using `ping`
2. TCP bandwidth measured by continue packet generation with the objective to saturate the link
3. UDP bandwidth and jitter, measured in the same way as TCP, but with UDP packets.

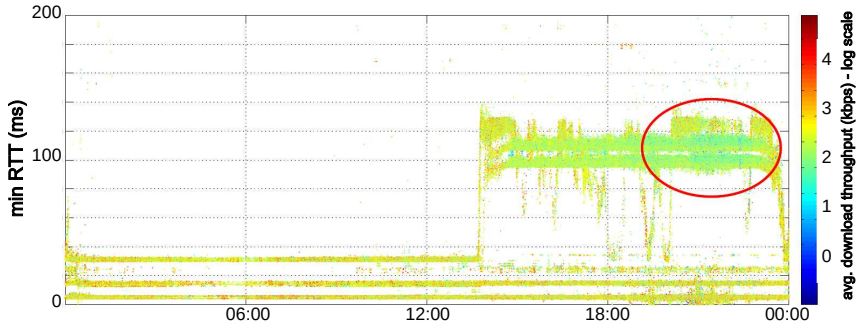
To describe the workflow of the whole procedure of SLA certification we could refer to Fig. 3.17.

In detail the work flow of SLA is as follows, based on numbers form [1-12] on Fig. 3.17:

1. On the first step, are the measurements that are done by the Probes, 10 `ping` samples, more than 10 seconds of measurements for TCP and UDP bandwidth.
2. The Probe does the measurements for the first time and transfers the data to the Repository, on the data measured are, RTT (10 `ping` samples), TCP throughput and UDP throughput.
3. After the Repository has received the data does simple elaboration to identify if there are any errors, like UDP error, or if there is any anomaly on the data, if it finds one it generates an alarm to alert the Reasoner.



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



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

Figure 3.16: The increase of the min RTT is not the root cause of the anomaly, as there are no major issues previous to the anomaly. However, there is a clear cluster of servers offering low throughput during the peak-load hours on an anomalous day

4. The Reasoner gets the data that generated the flag and does a check to see if it has to deploy the algorithm for UDP error checking of the one for anomaly detection, or if it has to notify the Supervisor to release the certificate with the required data since the measurement is complete.

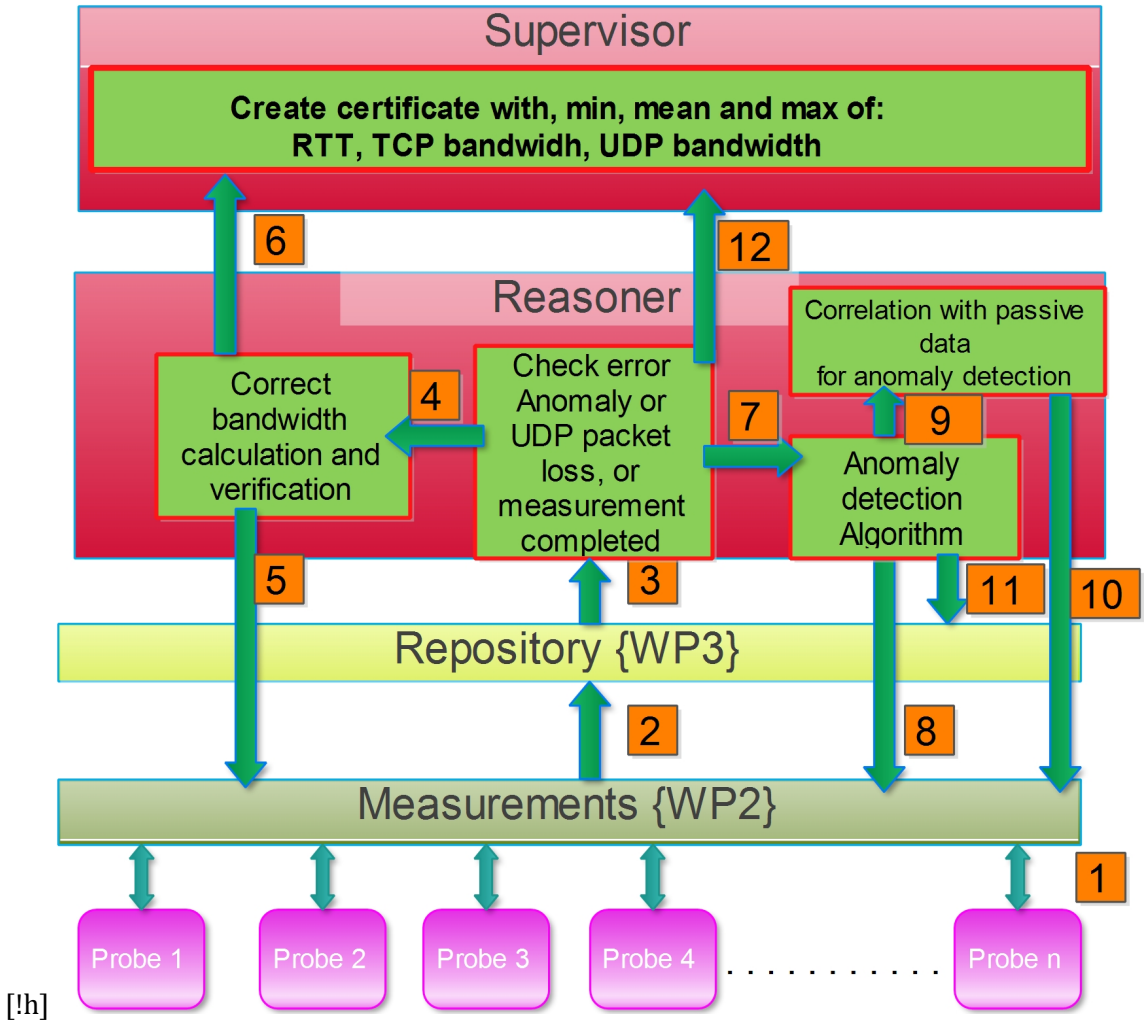
5. In case it identifies that there is an error generated by UDP, packet loss bigger than 0.1%, it starts the algorithm for identifying the correct speed that UDP test should be started. After the calculation of the correct UDP speed, the Reasoner tells to the measurement plane that it has to repeat the test at the calculated speed, after which the procedure from 2-5 repeats until the reasoner accepts the speed measured.

6. If the module of the calculation for the correct speed accepts the speed measured on the sequent test, it notifies the Supervisor to prepare a certificate with the min, mean, and max value of RTT, TCP throughput, and UDP speed.

7. If Reasoner after analyzes the data determines that an anomaly has been encountered, if starts the anomaly detection procedure

8. The module will determine if this is a temporal problem/ from the active probe, to be sure it will ask the measurement layer to make other measurements.

9. If the same problem is encountered, the procedure will go from 1-2-3-7 to step 9 in Fig. 3.17, to start the passive monitoring to analyze and correlate the active and passive data to find the root cause of the problem.



[!h]

Figure 3.17: Workflow of SLA use case for WP4

10. The algorithm for the correlation of the active and passive data will ask on its own to the measurement layer to make other measurements.

11. If after retrying the anomaly is not detected, or is not repeated anymore, the module of anomaly detection will communicate with the Repository to tell that that measurement should be flagged as unusable.

12. If there is no packet loss on the UDP test and there is no anomaly identified, the last notification is the one of terminating the measures, in this case the Reasoner identifies directly the Supervisor to release the certificate.

3.8.1 Probe Location

The SLA probe is composed of two parts:

1. The Agent: The part of the software that has to be on the clients PC

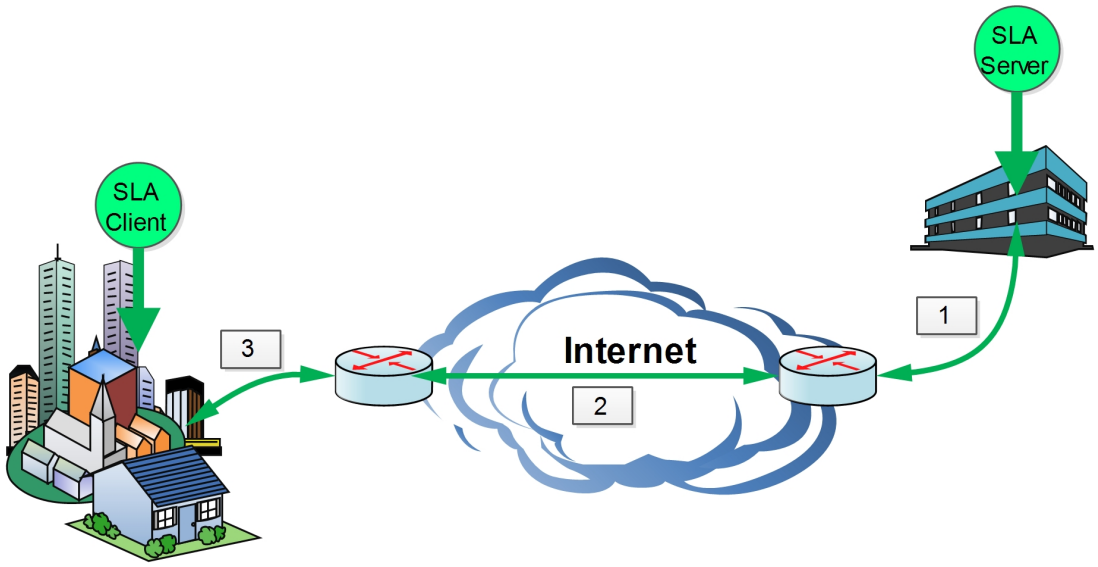


Figure 3.18: Workflow of SLA use case for WP4

2. The Server: The part of the software that has to be located on a server with a high speed connection

The agent is located at the client PC, always. The client is considered to be located usually at the terminal network. The agent can also be located anywhere, but with the concept in mind that the probe will always give the maximum bandwidth seen by client on the connection server client.

The server can be located anywhere, as long that his connection is higher than the one of the user. For example in Fig. 3.18, we have three links (1,2 and 3), the server probe should be located on a point of the network that allows the sequent conditions:

1. Link one has a higher capacity than link three.
2. In ideal condition link two has the highest capacity among all the links. It can also be allowed that link two has a smaller capacity than link one but a higher capacity than link three.
3. Minimum condition is that link three has to have the smallest capacity among all the links.

3.9 Locating probes for troubleshooting the path from the server to the user

Several mPlane use cases based on troubleshooting need some way of monitoring the paths from the servers hosting the monitored applications (YouTube, Web Browsing, Cloud Services, etc.) to the users. To this end, the knowledge of the actual path is necessary but not sufficient. As we have no control on the server, it is not possible to launch monitoring activities from that server towards the users. One way to circumvent this problem is to find some other nodes, referred to as *probes* in the sequel, near the server, and then trigger some path monitoring from them.

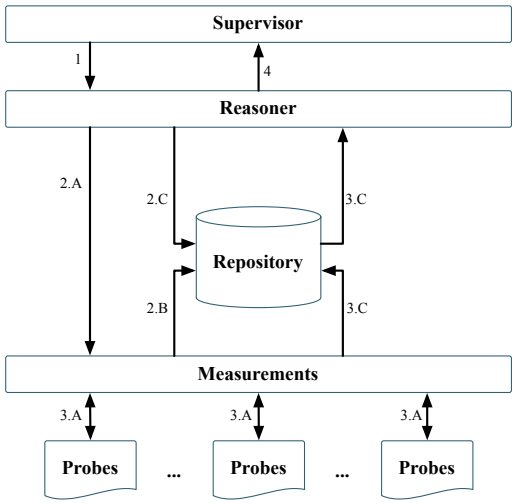


Figure 3.19: Probe location workflow

This section will address the first step of this process, namely the discovery of probes that are near the server, to allow various use cases to trigger the relevant monitoring activities based on standard tools.

Our problem can then be stated as follows.

In the considered application, a client, C, communicates with some point of interest (hereafter referred to as *PoI*), for example a CDN server streaming data to C.

When the performance is not acceptable, the client (or a proxy thereof) launches a troubleshooting procedure to diagnose the path from the PoI to C [1]. The first step consists, for the Supervisor, in triggering a Reasoner, R (see Step 1 in Fig. 3.19). The only data that need to be given to R are the IP addresses of C and the PoI.

To solve this problem, we assume the existence of geographically distributed probes (i.e., mPlane probes), whose IP addresses and ASNs are known (these have to be fetched and refreshed regularly). We also assume that the probes reply to `ping` and accept request from R to issue `traceroute` [104] to some IP addresses.

The sketch of the algorithm of the Reasoner is as follows:

1. Find the ASN of the PoI, knowing its IP address, using some IP-to-AS database(s). Candidate solutions here are using the TeamCymru service [98] or downloading BGP routing tables [103] and performing lookups.

2. Find the AS-Path from the PoI to C, using BGP data. This can be extracted from publicly available BGP data [103] (Steps 2.A and 2.B in Fig. 3.19).

3. Find a list of probes located near the PoI (the distance used here could be the RSD distance, or the estimated RTT) and rank them in order of increasing distances.

[1] Other tests can be used to diagnose the path from C to the PoI, but these tests, e.g., traceroute, are very easy, and therefore omitted here

4. Filter this list to remove false positives, namely initiate traceroutes from the probes in this list towards C and collect the result from the probes (Steps 3.A and 3.B in Fig. 3.19). The probes whose AS-Path towards C matches the AS-path from the PoI to C, or is close enough, are kept (e.g. they share a long common AS-path suffix).
5. Reiterate the process (steps 3 to 4) to find probes near intermediate nodes along the AS-path from PoI to C.
6. Output the list of probes in some structured order in decreasing order of common suffixes (Steps 4 in Fig. 3.19).

These two possible methods for step 3 were detailed in Sec. 2.9.1.

3.9.1 Ranking probes with respect to their distances to some point of interest

We now show how this method can be applied to our problem. For every probe, its RTT to the PoI will be estimated. We rely on the existence of a few dozen vantage points, typically mPlane nodes. The Reasoner will send to all the vantage points a list of nodes (probes + PoI) to ping. These vantage points will ping them, measure the RTTs, and return the results to the Reasoner. Given the symmetry of RTTs, they are equal to the RTTs that the PoI and the probes would themselves issue towards these vantage points.

Then the collected dataset at the Reasoner will be structured as follows:

- there will be one feature vector per Probe and an additional one for the PoI,
- these feature vectors have as many components as there are landmarks,
- and each component of these feature vectors is the RTT between the probe (or PoI) and the corresponding landmark.

Then, the Reasoner estimates the distances between all the probes and the PoI. Finally a ranking of probes is produced, which is a list of probes in increasing order of estimated RTT distances to the PoI.

In addition, the Supervisor advertises the capability to locate mPlane probes near some specified Point of Interest (PoI).

3.10 Topology

This section presents the mPlane workflow for network topology related algorithms described in Sec. 2.10. Those algorithms can be used by several use cases. For instance, `tracebox` may find a suitable usage in the mobile use case (with the `tracebox` port onto Android). Fig. 3.20 illustrates how topology discovery algorithms inserts themselves in the mPlane architecture.

3.10.1 MPLS Tunnel Diversity

Scamper [63] has been extended and ported into the mPlane architecture. scamper offers the opportunity to reveal, through `traceroute` probes, the presence of MPLS tunnels. Next, the LPR algo-

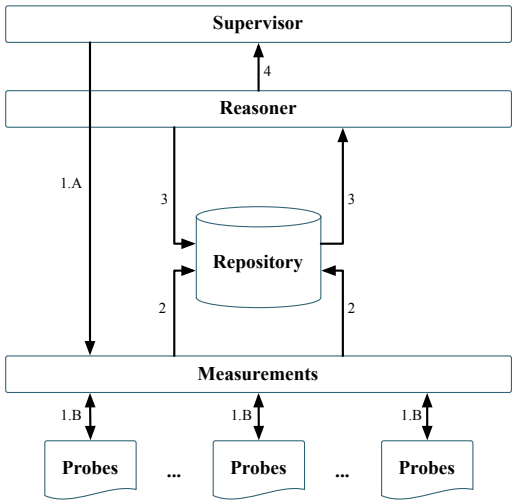


Figure 3.20: Topology discovery workflow

rithm (Sec. 2.10.1.2) is able to analyze the collected dataset to determine the potential MPLS tunnel diversity.

A basic workflow of the MPLS tunnel diversity analysis involves the following components:

- **Supervisor**. The Supervisor advertises the capability to determine, based on a set of traces going through MPLS tunnels, whether those tunnels allows for traffic engineering (TE) or not. It receives requests from clients and possibly asks Probes to perform measurement (Step 1.A on Fig. 3.20).

- **Scamper Probes**. The scamper [63] Probes, developed in mPlane WP2, are geographically distributed. On one hand, they may be configured to "constantly" probe the network through a list of destinations. The probing can be based on standard `traceroute` (ICMP, UDP, or TCP) or on ParisTraceroute [7] (also ICMP, UDP, or TCP). On the other hand, they can act "on demand" according to specific requirements by the Supervisor. Regarding the later, the Supervisor provides the Probes with a given list of destinations and the type of probing requires (`traceroute` or Paris Traceroute with the protocol – ICMP, TCP, UDP – considered). Both cases are illustrated by Step 1.B on Fig. 3.20. Data collected is exported in `warts` format with mplane-http or mplane-ssh to the Repository.

- **Repository**. Depending on how Probes have been configured (i.e., long term measurements or on demand measurements), the Repository will act differently. For long term measurements, raw data is simply stored within the Repository. For on demand measurements, the raw data is not only stored but also preprocessed by the Repository. Only MPLS tunnels seen as the scamper Probe address, <Ingress LER, Egress LER> pair, LSPs, and `traceroute` destinations are considered. The preprocessed data is dumped into a text file that is next sent to the Reasoner. This is Step 2 on Fig. 3.20. Data are sent to the Reasoner on demand, in a text file format (Step 3 on Fig. 3.20).

- **Reasoner**. The Reasoner runs the LPR algorithm (see Sec. 2.10.1.2) on a text file that aggregates `traceroute` measurements from geographically distributed scamper probes. The

Reasoner sends the results to the Supervisor (Step 4 on Fig. 3.20) that is, next, in charge of dispatching them to the clients.

3.10.2 Middleboxes Discovery

Regarding middleboxes detection, there are two possibilities. Either the client evolves in the mobile world and, consequently, will preferably use `tracebox` Android. Either the client is in the wired network and will preferably consider the scamper port of `tracebox`.

A basic workflow of the middlebox discovery in the mPlane architecture involves the following components:

- **Supervisor**. The Supervisor advertises the capability to reveal the presence (or not) of middleboxes along a given path. It receives requests from clients and possibly asks Scamper or Android Probes to perform measurement (Step 1.A on Fig. 3.20).

- **Scamper Probes**. The scamper [63] Probes, developed in mPlane WP2, are geographically distributed. We extended scamper so that it includes now `tracebox`. Each scamper Probe announces three capabilities, each one being divided into IPv4 and IPv6: "standard capability" (a simple `tracebox` probe over TCP without any option), "specific capability" (a `tracebox` probe that can be precisely specified by selecting IP fields value – ECN, DSCP, IPID/IPFLOW – , the transport layer – TCP or UDP –, and various TCP options – MSS, WScale, ECN, ...–) and "specific quote size capability" (the same as specific capability but it provides additional results, such as the ICMP quoting size used by each hop). Each scamper Probe measures the target provided in the specification message received and export the measurements with mplane-http or mplane-ssh to the Repository (Steps 1.B and 2 on Fig. 3.20).

- **Android Probes**. Android Probes, developed in mPlane WP2, are geographically distributed and must be explicitly installed by clients on their Android mobile device (that must be rooted first). Android Probes offer a single capability (under IPv4): "standard capability". Each Android Probe measures the target provided in the specification message received and export the measurements with mplane-http or mplane-ssh to the Repository (Steps 1.B and 2 on Fig. 3.20).

- **Repository**. Data is stored in the Repository. If the data comes from scamper Probes, traces are stored in a raw `warts` format with each measurement being identified by a $(run\ \#, saddr, daddr, key)$-tuple where the key is used to match possible concurrent probing runs. Data are sent to the Reasoner on demand (Step 3 on Fig. 3.20).

- **Reasoner**. The Reasoner compares Probes sent with messages received from routers along the paths in order to identify possible middleboxes. In addition, the Reasoner may confront the middlebox with the taxonomy (see Sec. 2.10.2) in order to define the type of path-impairment. Note that, in some cases, a single probing cannot be enough to reveal the presence of a middlebox. Thus, the Reasoner may elaborate new tests based on previous tests results. It may reconfigure Probes characteristics, change `tracebox` settings or modify destinations (Step 1.B on Fig. 3.20). The Reasoner sends the results to the Supervisor (Step 4 on Fig. 3.20) that is, next, in charge of dispatching them to the clients.

3.10.3 IGP Weight Inference

- **Supervisor**. The Supervisor advertises the capability to infer IGP weight of paths in a given AS. It receives requests from clients and possibly asks Probes to perform measurement (Step 1.A on Fig. 3.20).
- **MERLIN Probes**. The MERLIN [67] Probes are geographically distributed. Each MERLIN Probe comes with a single capability, i.e., "probe". It receives the list of IP addresses to target. Each MERLIN Probe measures the targets provided in the specification message received and export the measurements, in a text file format, with mplane-http or mplane-ssh to the Repository (Step 2 on Fig. 3.20).
- **Repository**. Data is stored and preprocessed in the Repository. In the context of IGP weight inference, the Repository stores two kinds of data: the MERLIN dataset, as text files, and the scamper dataset, as `warts` files (Step 3 on Fig. 3.20).
- **Reasoner**. Once data collected by MERLIN has been received, the Reasoner is in charge of retrieving, from the dataset, the largest connected component. From this component, the Reasoner generates a list of IP addresses for each subnet /24 within the component. This list is sent to scamper Probes (Step 1.B on Fig. 3.20), with the capability "Paris Traceroute with MDA". Once the Paris traceroutes have been done, the Reasoner builds the constraint system and solves it. Results are sent back to the Supervisor that is in charge of dispatching them to the client (Step 4 on Fig. 3.20).
- **Scamper probes**. The scamper [63] Probes, developed in mPlane WP2, are geographically distributed. Scamper Probes offer several capabilities. Here, we are interested by the "Paris Traceroute with MDA" [7, 8] capability that allows one to perform Paris Traceroute with the detection of load-balanced paths. This can be done in ICMP, UDP, TCP mode. Scamper also receives, from the Reasoner, the list of targets. Each scamper Probe measures the target provided in the specification message received and export the measurements, in `warts` format, with mplane-http or mplane-ssh to the Repository.

3.11 Internet-Scale Anycast Scanner

This section presents the workflow of an Internet-scale anycast scanner in mPlane based on the enumeration and geolocation methodology presented in Sec. 2.11. The main goal of the scanner is to periodically scan the Internet address space to track the evolution of anycast adoption and server deployments over time. More specifically, the tool identifies IPV4 prefixes using anycast. It also enumerates and geographically locates the various replicas per anycast prefix.

The anycast scanner can assist in the diagnosis of network performance anomalies with CDNs using anycast (e.g., EdgeCast networks [2] [30]). More specifically, it helps ISPs discover newly deployed servers, and thus explain abnormal shifts in the traffic they carry toward an anycast service. The tool can be seen as part of a larger effort from the research community to geographically map the Internet infrastructure [3, 15] and identify the various components of the physical Internet [28] (e.g., data centers, PoPs). Although most anycast service providers regularly publish physical maps

[2] EdgeCast serves popular Internet services such as Twitter, LinkedIn, and WordPress from geographically distributed anycasted servers around the globe

with the number and locations of their servers, this information is often incomplete and outdated. One reason for this is that the Internet is constantly evolving to meet client demands. Thus, periodic census of the Internet address space (ranging from several times a day, to once every few weeks depending on the usage) can provide timely representations of anycast adoption and deployments. As a side note, the tool can potentially detect BGP prefix hijacking.

3.11.1 mPlane Workflow

The anycast scanner has two components: a back-end Measurement and Analysis component and a front-end Visualization component. At the back-end, a Supervisor instructs mPlane FastPing Probes to periodically measure the latency to all /24 prefixes and send the results to a Repository. The Reasoner runs an algorithm to detect anycast IPs based on the aggregated measurements, and an algorithm to enumerate and geolocate replicas of the anycast IPs. The output is pushed to a front-end Repository. At the front-end, a web-based Visualization tool presents a historic and near real-time view of IPV4 anycast adoption and the geographical deployments of existing anycast services based on queries to the front-end Repository. Through the web-page, we plan to provide the user with a visual representation of the evolution of the number of IPs using anycast. Additionally, for each anycast IP, we plan to show maps of server deployments and deployment changes across time.

Fig. 3.21 depicts a basic workflow of the service in mPlane. The back-end component proceeds as follows:

- **FastPing Probes.** The FastPing [18] probes, developed in WP2, belong to the same mPlane domain. They are configured with the hostname of their Supervisor and therefore periodically report to the Supervisor the capability to "measure" and "export" measurements. The Supervisor maintains a list of all active Probes seen in the last X minutes.

- **Back-end Repository.** The Repository exports the capability to "collect" measurements. The capability statement specifies the URL (i.e., hostname and transport protocol number) at which the Repository will wait for the measurements.

- **Front-end Repository.** The Repository exports the capability to "store" measurements and answer queries at a specific URL. Both operations should be in JSON format (*application/x-mplane+json*).

- **Supervisor.** The Supervisor periodically (e.g., once per day) instructs the active Probes to "measure" the Internet address space and "export" the measurements with *mplane-http* or *mplane-ssh* to the back-end Repository. The specification statement from the Supervisor to the Probes includes a link section with the address of the Repository. The Supervisor sends a "collect" specification to the Repository with the duration of the measurement collection period. Moreover, the Supervisor specifies the address of the Reasoner.

  Each Probe sends a receipt back to the Supervisor acknowledging the reception of the "measure" and "export" specifications. Next, each Probe uploads its own measurements to the Repository. When the measurement collection period expires, the back-end Repository extracts the minimum number of RTT per probe-destination prefix tuple, outputs the results in Text-CSV format, and exports this file to the Reasoner.

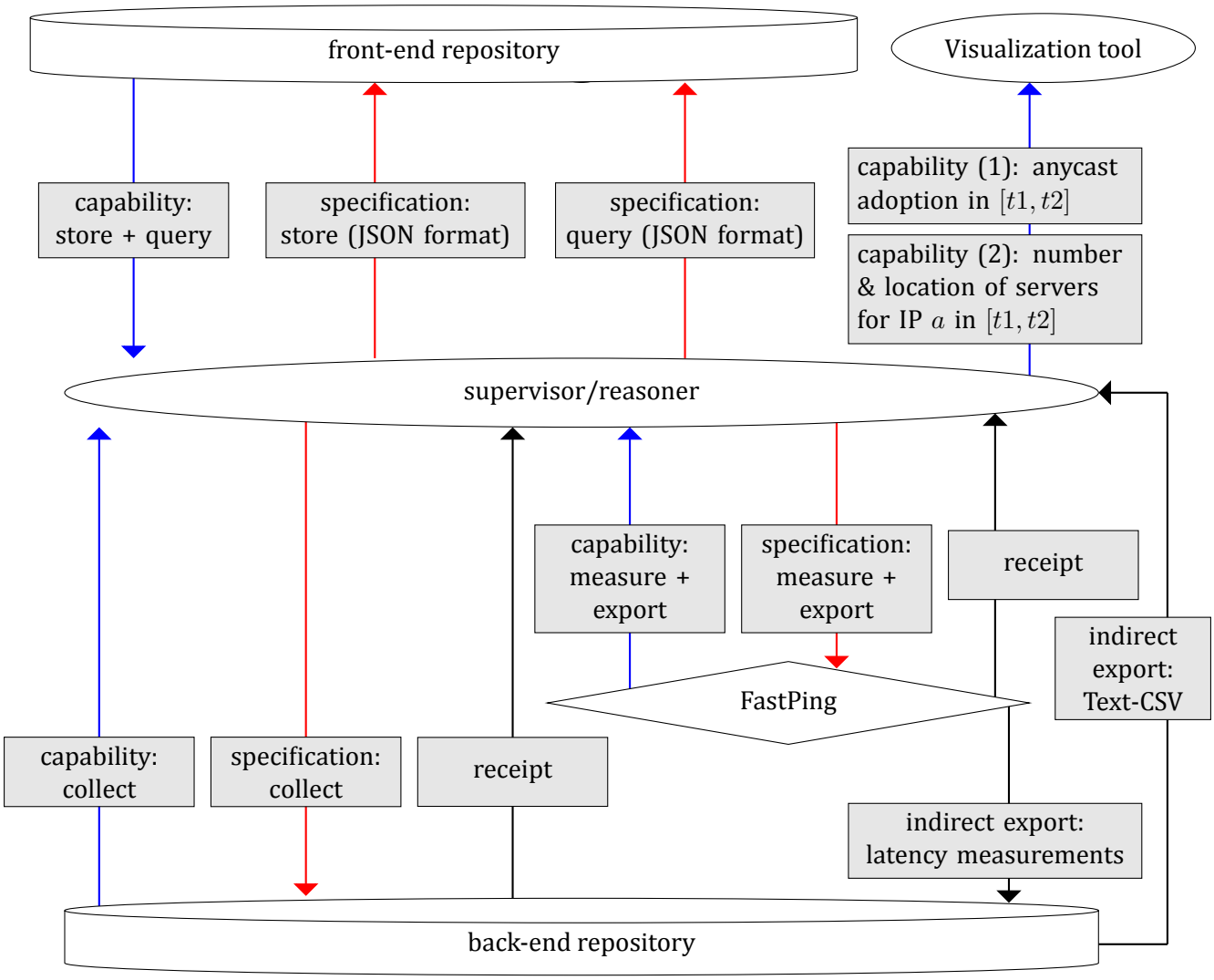


Figure 3.21: Anycast scanner: ”Capability push / Specification push” workflow in mPlane

- **Reasoner.** The Reasoner runs the anycast detection and enumeration algorithm on the CSV file which aggregates, for each IP prefix, the latency measurements from geographically distributed FastPing Probes. The Reasoner ships the results to the Supervisor, which stores them in the front-end Repository.

The front-end component is a web-based Visualization tool that provides users with information about the fraction of anycast prefixes detected, and maps of the number and locations of replicas for a specific anycast IP. Both pieces of information are bounded by a time interval $t1$ and $t2$ that the user has to enter on the web-page. The web-page allows users to browse the history of anycast adoption rate and the maps of server deployment over time for an anycast service. The web-page translates user requests into JSON specifications to the supervisor. The supervisor sends “query” messages to the front-end repository and relays the answers back to the web-page also in JSON.

3.11.2 Challenges

Anycast enumeration is challenging because instances can be global (i.e., highly reachable from everywhere in the Internet) or local (i.e., topologically scoped to a catchment area). Therefore, only probes present in catchments areas of new instances can discover them. However, in order to enumerate as many servers as possible, a naive strategy is to use all the available probes. Fan et al. [33] employs 62k vantage points deployed at end-user laptops and desktops to enumerate tens of DNS root servers and 300k open recursive name servers to discover 14 AS112 instances. Clearly, there is an imbalance (logarithmic relationship [33]) between the number of probes and the number of worldwide instances to discover (the vast majority of services have a few dozens instances, 100 being an upper limit rarely reached). Leveraging rDNS servers to perform redundant measurements (e.g., multiple probes querying the same anycast instance) is a waste of critical Internet infrastructure resources and might incur additional load on the overall DNS system. The main concern here is the significant use of processing and networking resources and the probing load incurred at the Probes and the targets. High probing traffic towards a given IP destination or range can appear suspicious and lead to the black listing of Probes.

We are currently exploring several metrics to bias a-priori the selection of Probes by the Supervisor.

4 Conclusion

This deliverable presents updates to analysis algorithms initially provided in Deliverable D4.1 [73] (Chap. 2). These algorithms perform analyses on the data collected by measurement probes (WP2) and/or stored and pre-processed in the repositories (WP3). Each use-case (WP1) has been addressed from this perspective, focusing on its analysis modules. In particular, we inspect each use case and discuss improvements and new validation results. In addition, we have proposed a series of more generic algorithms (e.g., related to network topology and routing) that will be useful to better support various use-cases currently considered and also in a larger context.

The deliverable also proposes the workflow descriptions of the different use-cases. They specify the iterative interaction loops between various mPlane modules (from WP2, WP3 and WP4), thereby allowing for a cross-checking of the analysis modules and the reasoner interactions. In each use case, we show how the relevant Reasoner coordinates the measurements and the analysis performed by Probes and Repositories, actuating through the Supervisor. This Reasoner is able to iteratively perform different analysis tasks, taking additional analysis steps based on the results of the previous observed results. As such, this 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.

Bibliography

[1] AS112 project. See `https://www.as112.net`.

[2] Root server technical operations. See `http://www.root-servers.org`.

[3] E. Aben. Router geolocation, Ocrober 2014. RIPE 67 (see `https://ripe67.ripe.net/presentations/341-2013-10-ripe67-router-geoloc-emile-aben.pdf`).

[4] B. Aboba and W. Dixon. IPsec-network address translation (NAT) compatibility requirements. RFC 3715, Internet Engineering Task Force, March 2004.

[5] D. Agrawal, S. Calo, K. Lee, J. Lobo, and D. Verma. *Policy Definition and Usage Scenarios for Self-Managing Systems*. IBM Press, 2008.

[6] L. Andersson, I. Minei, and T. Thomas. LDP specification. RFC 5036, Internet Engineering Task Force, October 2007.

[7] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *Proc. ACM Internet Measurement Conference (IMC)*, October 2006.

[8] B. Augustin, R. Teixeira, and T. Friedman. Measuring load-balanced paths in the Internet. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2007.

[9] D. Awduche, L. Berger, D. Gan, T. Li, and G. Srinivasan, V. ans Swallow. RSVP-TE: Extensions to RSVP for LSP tunnels. RFC 3209, Internet Engineering Task Force, December 2001.

[10] H. Ballani and P. Francis. Towards a global IP anycast service. In *Proc. ACM SIGCOMM*, 2005.

[11] H. Ballani, P. Francis, and S. Ratnasamy. A measurement-based deployement proposal for IP anycast. In *Proc. ACM Internet Measurement Conference (IMC)*, October 2006.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[13] A. Botta, A. Dainotti, and A. Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks (COMNET)*, 56(15):3531–3547, October 2012.

[14] M. O. Buob, T. Delac, and T. Friedman. Fakeroute, July 2013. See `https://code.google.com/p/paris-traceroute/wiki/Fakeroute`.

[15] M. Calder, X. Fan, Z. Hu, e. Katz-Bassett, J. Heidemann, and R. Govindan. Mapping the expansion of Google's serving infrastructure. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2013.

[16] G. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the monitor placement problem: Optimal network-wide sampling. In *Proc. 40th Annual Conference on Information Sciences and Systems*, March 2006.

[17] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. RFC 3234, Internet Engineering Task Force, February 2002.

[18] P. Casoria, D. Cicalese, D. Joumblatt, and D. Rossi. Fastping, May 204. `http://www.ict-mplane.eu/public/fastping`.

[19] C. Chaudet, E. Fleury, I. G. Lassous, H. Rivano, and M.-E. Voge. Optimal positioning of active and passive monitoring devices. In *Proc. ACM CoNEXT*, December 2005.

[20] k. claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov. Internet mapping: from art to science. In *IEEE Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, March 2009.

[21] A. Coyle, M. Kraetzl, O. Maennel, and M. Roughan. On the predictive power of shortest-path weight inference. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2008.

[22] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *Proc. ACM SIGCOMM*, August 2004.

[23] L. D'Acunto, N. Chiluka, T. Vinò, and H. J. Sips. Bittorrent-like P2P approaches for VoD: a comparative study. *Computer Networks (COMNET)*, 57(5):1253–1276, April 2013.

[24] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet. Revealing middlebox interference with tracebox. In *Proc. ACM Internet Measurement Conference (IMC)*, October 2013.

[25] B. Donnet, M. Luckie, P. Mérindol, and J.-J. Pansiot. Revealing MPLS tunnels obscured from traceroute. *ACM SIGCOMM Computer Communication Review*, 42(2):87–93, April 2012.

[26] W. Du, Y. Liao, P. Geurts, and g. Leduc. Ordinal rating of network performance and inference by matrix completion. cs.NI 1211.0447, arXiv, November 2012.

[27] W. Du, Y. Liao, N. Tao, P. Geurts, X. Fu, and G. Leduc. Rating network paths for locality-aware overlay construction and routing. *IEEE/ACM Transactions on Networking*, July 2014.

[28] R. Durairajan, S. Ghosh, X. Tang, P. Barford, and B. Eriksson. Internet atlas: A geographic database of the Internet. In *Proc. ACM Workshop on HotPlanet*, August 2013.

[29] M. Dusi, S. Napolitano, S. Niccolini, and S. Longo. A closer look at thin-client connections: Statistical application identification for QoE detection. *IEEE Communications Magazine*, 50(11):195–202, November 2012.

[30] Edgecast. Edgecast networks. `http://www.edgecast.com`.

[31] B. Eriksson, P. Barford, J. Sommers, and R. Nowak. A learning-based approach for IP geolocation. In *Proc. Passive and Active Measurement Conference (PAM)*, April 2010.

[32] B. Eriksson and M. Crovella. Understanding geolocation accuracy using network geometry. In *Proc. IEEE INFOCOM*, 2013.

[33] X. Fan, J. S. Heidemann, and R. Govindan. Evaluating anycast in the domain name system. In *Proc. IEEE INFOCOM*, April 2013.

[34] T. Flach, E. Katz-Bassett, and Govindan. Quantifying violations of destination-based forwarding on the Internet. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2012.

[35] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses. RFC 6824, Internet Engineering Task Force, January 2014.

[36] J. Frahom, O. Santos, and A. Ossipov. *Cisco ASA: All-in-One Firewall, IPS, and VPN Adaptive Security Appliance*. Pearson Education, 2014.

[37] A. Geoffrion. Objective function approximations in mathematical programming. *Mathematical Programming*, 13(1):23–37, August 1977.

[38] Google. `https://support.google.com/news/publisher/answer/40392`.

[39] B. R. Greene and D. McPherson. Sink holes: a swiss army knife ISP tool, June 2003. Nanog 28 (see `https://www.nanog.org/meetings/nanog28/presentations/sink.pdf`).

[40] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of Internet hosts. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2004.

[41] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. Internet Measurement Workshop (IMW)*, November 2002.

[42] G. Gürsun, N. Ruchansky, E. Terzi, and M. Crovella. Routing state distance: A path-based metric for network analysis. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2012.

[43] H. Haddadi, G. Iannaccone, A. Moore, R. Mortier, and M. Rio. Network topologies: Inference, modeling and generation. *IEEE Communications Surveys and Tutorials*, 10(2):48–69, April 2008.

[44] T. Hardie. Distributing authoritative name serveres via shared unicast addresses. RFC 3258, Internet Engineering Task Force, April 2002.

[45] C. Hennig and M. Kutlukaya. Some thoughts about the design of loss functions. *REVSTAT–Statistical Journal*, 5(1):19–39, 2007.

[46] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure. Are TCP extensions middlebox-proof? In *Proc. Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, December 2013.

[47] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2011.

[48] C. Huitema. An anycast prefix for 6to4 relay routers. RFC 3068, Internet Engineering Task Force, June 2001.

[49] S. . ITU-T. Parametric non-intrusive assessment of audiovisual media streaming quality. amendment 2: New appendix iii - use of p.1201 for non-adaptive, progressive download type media streaming. *ITU-T P.1201 Recommendation*, 2013.

[50] K. Jansen. Approximation algorithms for geometric intersection graphs. In *Proc. International Workshop on Graph-Theoretic Concepts in Computer Science*, June 2007.

[51] S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen. Analysis of vulnerabilities in Internet firewalls. *Computers & Security*, 22(3):214–232, April 2003.

[52] D. Katz, K. Kompella, and D. Yeung. Traffic engineering (TE) extensions to OSPF version 2. RFC 3630, Internet Engineering Task Force, September 2003.

[53] K. Keys. Internet-scale IP alias resolution techniques. *ACM SIGCOMM Computer Communication Review*, 40(1):50–55, January 2010.

[54] D. Kim, D. Meyer, H. Kilmer, and D. Farinacci. Anycast rendez-vous point (RP) mechanism using protocol independant multicast (PIM) and multicast source discovery protocol (MSDP). RFC 3446, Internet Engineering Task Force, January 2003.

[55] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet topology zoo. *IEEE Journal on Selected Areas in Communications (JSAC)*, 29(9):1765–1775, October 2011.

[56] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[57] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *Proc. ACM Internet Measurement Conference (IMC)*, November 20010.

[58] J. Ledlie, P. Gardner, and M. I. Seltzer. Network coordinates in the wild. In *Proc. USENIX Symposium on Networked System Design and Implementation (NSDI)*, April 2007.

[59] M. A. Lemley and L. Lessig. The end of end-to-end: Preserving the architecture of the Internet in the broadband era. Technical Report 2000-19, University of California at Los Angeles, October 2000.

[60] T. Li and H. Smit. IS-IS extensions for traffic engineering. RFC 5305, Internet Engineering Task Force, October 2008.

[61] Y. Liao, W. Du, P. Geurts, and G. Leduc. Decentralized prediction of end-to-end network performance classes. In *Proc. ACM CoNEXT*, December 2011.

[62] Y. Liao, W. Du, P. Geurts, and G. Leduc. DMFSGD: A decentralized matrix factorization algorithm for network distance prediction. *IEEE/ACM Transactions on Networking*, 21(5):1511–1524, October 2013.

[63] M. Luckie. Scamper: a scalable and extensible packet prober for active measurement of the Internet. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2010.

[64] D. Madory, C. Cook, and K. Miao. Who are the anycasters?, October 2013. Nanog 59 (See `http://research.dyn.com/wp-content/uploads/2014/07/NANOG59_Anycast.pdf`).

[65] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *Proc. ACM SIGCOMM Internet Measurement Workshop (IMW)*, November 2002.

[66] Y. Mao, L. Saul, and J. M. Smith. IDES: An Internet distance estimation service for large networks. *IEEE Journal On Selected Areas in Communications (JSAC)*, 24(12):2273–2284, December 2006.

[67] P. Marchetta, P. Mérindol, B. Donnet, A. Pescapé, and J.-J. Pansiot. Topology discovery at the router level: a new hybrid tool targeting ISP networks. *IEEE Journal on Selected Areas in Communication (JSAC)*, 29(6):1776–1787, October 2011.

[68] P. Marchetta, P. Mérindol, B. Donnet, A. Pescapé, and J.-J. Pansiot. Quantifying and mitigating IGMP filtering in topology discovery. In *Proc. IEEE Global Communications Conference (GLOBECOM)*, December 2012.

[69] MaxMind. Geolocation and online fraud prevention from MaxMind. `http://www.maxmind.com/`.

[70] A. Medina, M. Allman, and S. Floyd. Measuring interactions between transport protocols and middleboxes. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2004.

[71] P. Mérindol, B. Donnet, J.-J. Pansiot, M. Luckie, and Y. Hyun. MERLIN: MEasure the Router Level of the INternet. In *Proc. 7th Euro-nf Conference on Next Generation Internet (NGI)*, June 2011.

[72] R. Mok, E. Chan, and R. Chang. Measuring the quality of experience of HTTP video streaming. In *Proc. IFIP/IEEE International Symposium on Integrated Network Management (INM)*, May 2011.

[73] mPlane consortium. Public Deliverables. `http://www.ict-mplane.eu/public/public-deliverables`.

[74] K. Muthukrishnan and A. Malis. A core MPLS IP VPN architecture. RFC 2917, Internet Engineering Task Force, September 2000.

[75] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. IEEE INFOCOM*, June 2002.

[76] E. Normark. Stateless IP/ICMP translation algorithm (SIIT). RFC 2765, Internet Engineering Task Force, February 2000.

[77] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. *Networks*, 55(3):276–286, MAy 2010.

[78] D. Papadimitriou and B. Fortz. Time-dependent combined network design and routing optimization. In *Proc. IEEE International Conference on Communications (ICC)*, June 2014.

[79] C. Partridge, T. Mendez, and W. Milliken. Host anycasting service. RFC 1546, Internet Engineering Task Force, November 1993.

[80] PlanetLab Consortium. PlanetLab project, 2002. See `http://www.planet-lab.org`.

[81] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. IP geolocation databases: Unrealiable? *ACM/SIGCOMM Computer Communication Review*, 54(18):3373–3387, December 2010.

[82] Z. Qian and Z. M. Mao. Off-path TCP sequence number inference aattack - how firewall middleboxes reduce security. In *Proc. IEEE Symposium on Security and Privacy (SP)*, May 2012.

[83] Z. Qian, Z. M. Mao, and Y. Xie. Collaborative TCP sequence number inference attack: How to crack sequence number under a second. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, October 2012.

[84] RIPE NCC. RIPE atlas. See `https://atlas.ripe.net`.

[85] E. Rosen and Y. Rekhter. BGP/MPLS IP virtual private networks (VPNs). RFC 4364, Internet Engineering Task Force, February 2006.

[86] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. RFC 3031, Internet Engineering Task Force, January 2001.

[87] A. Rufini, M. Mellia, E. Tego, and F. Matera. Multilevel bandwidth measurements and capacity exploitation in gigabit passive optical networks. *IET Communications*, 8(18):3357–3365, December 2014.

[88] S. Sarat, V. Pappas, and A. Terzis. On the use of anycast in DNS. In *Proc. 15th International Conference on Computer Communications and Networks (ICCCN)*, October 2006.

[89] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. International World Wide Web Conference*, May 2001.

[90] Y. Shavitt and N. Zilberman. A geolocation databases study. *IEEE Journal on Selected Areas in Communications*, 29(10), 2011.

[91] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proc. ACM SIGCOMM*, August 2012.

[92] R. Sherwood, A. Bender, and N. Spring. Discarte: a disjunctive Internet cartographer. In *ACM SIGCOMM*, August 2008.

[93] J. Sommers, B. Eriksson, and P. Barford. On the prevalence and characteristics of MPLS deployments in the open Internet. In *Proc. ACM Internet Measurement Conference (IMC)*, November 2011.

[94] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.

[95] C. Srinivasan, L. P. Bloomberg, A. Viswanathan, and T. Nadeau. Multiprotocol label switching (MPLS) traffic engineering (TE) management information base (MIB). RFC 3812, Internet Engineering Task Force, June 2004.

[96] K. Suh, G. Yang, J. Kurose, and D. Towsley. Locating network monitors: Complexity, heuristics, and coverage. In *Proc. IEEE INFOCOM*, March 2005.

[97] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blažek, and H. Kim. Detection of intrusions in information systems by sequential change-point methods. *Statistical Methodology*, 3(3):252 – 293, July 2006.

[98] Team Cymry. Internet security research and insight, 2014. See `http://www.team-cymru.org`.

[99] R. Teixeira, K. Marzullo, S. Savage, and G. Voelker. In search of path diversity in ISP networks. In *Proc. ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2003.

[100] M. E. Tozal and K. Sarac. Subnet level network topology mapping. In *Proc. IEEE International Performance Computing and Communications Conference (IPCCC)*, November 2011.

[101] S. Traverso, E. Tego, E. Kowallik, S. Raffaglio, A. Fregosi, M. Mellia, and F. Matera. Exploiting hybrid measurements for network troubleshooting. In *Proc. 16th International Telecommunications Network Strategy and Planning Symposium (Networks)*, September 2014.

[102] G. Tsirtsis and P. Srisuresh. Network address translation-protocol translation. RFC 2766, Internet Engineering Task Force, February 2000.

[103] University of Oregon. Route views, University of Oregon Route Views project. See `http://www.routeviews.org/`.

[104] V. Jacobson et al. traceroute. man page, UNIX, 1989. See source code: `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`.

[105] N. Wang, K. Ho, G. Pavlou, and M. Howarth. An overview of routing optimization for Internet traffic engineering. *IEEE Communications Surveys and Tutorials*, 10(1):36–56, April 2008.

[106] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *Proc. ACM SIGCOMM*, August 2011.

[107] B. Wong, A. Slivkins, and E. Sirer. Meridian: A lightweight network location service without virtual coordinates. In *Proc. ACM SIGCOMM*, August 2005.

[108] X. Xiao, A. Hannan, and B. Bailey. Traffic engineering with MPLS in the Internet. *IEEE Network Magazine*, 14(2), April 2000.

[109] I. G. Young. The code that powers reddit.com. `https://github.com/iangreenleaf/reddit`.

[110] YouTube. Most viewed videos of all time, October 2012. `http://www.youtube.com/playlist?list=PLirAqAtl_h2r5g8xGajEwdXd3x1sZh8hC`.