mPlane

an Intelligent Measurement Plane for Future Network and Application Management

ICT FP7-318627

Design of Analysis Modules

Author(s):	Author names
A-LBELL	D. Papadimitriou
ALBLF	Z. Ben Houidi, S. Ghamri-Doudane
ENST	D. Rossi
Eurecom	M. Milanesio
FTW	P. Casas, A. D'Alconzo
FUB	E. Tego, F. Matera
NEC	M. Dusi
NETvisor	T. Szemethy, D. Máthé
POLITO	S. Traverso, A. Finamore
TID	I. Leontiadis, L. Baltrunas, Y. Grunenburger
ULg (editor)	B. Donnet, G. Leduc, Y. Liao

Document Number:	D4.1
Revision:	1.0
Revision Date:	31 Oct 2013
Deliverable Type:	RTD
Due Date of Delivery:	31 Oct 2013
Actual Date of Delivery:	31 Oct 2013
Nature of the Deliverable:	(R)eport
Dissemination Level:	Public

Abstract:

This public deliverable describes the design and specification of a first set of basic analysis modules for addressing the use cases identified in WP1. The document focuses on the required algorithms, which use as input the measurements and analysis provided by the lower layers (WP2 and WP3) of the mPlane architecture to provide more advanced analysis and answers towards the resolution of the problem addressed by the use case. These analysis modules include both stream and batch processing algorithms and address issues such as classifications, estimations, predictions, detections, correlations and diagnosis.

Contents

1 Introduction

Deliverable D3.1 has already defined and described basic algorithms that operate on the data storage and large-scale analysis layer (WP3) of the mPlane architecture, generally on very large amounts of data. Their associated repositories expect to receive input data from a multitude of probes, as defined in WP2. This amount of data is processed, analyzed and aggregated in the storage layer using parallel and scalable frameworks, and later served to the data analysis layer (WP4), which makes use of it through more advanced and with higher-visibility algorithms and through the Reasoner.

Basic algorithms in WP3 thus produce data that is consumed by other algorithms (recursively at WP3, or at WP4) and/or by the Reasoner (WP4). By construction, WP4 algorithms most often accept data of smaller size, relying on other algorithms defined in WP3 that already aggregate data and extract a set of features conveying a summary of information and which size is more convenient and feasible to be treated by the algorithms at WP4. Note that WP4 algorithms might receive as input data directly coming from WP2, but only for specific time frames and/or data-size-limited analysis tasks, specially in the case of instantiation of new functionality at the probes, during the iterative process.

The algorithm definition is use case driven and follows the use cases described in D1.1. For each use case we identify:

- the available measurements produced by the probes at WP2 (available in D2.1 and D5.1),
- the basic algorithms necessary to process them at WP3 (available in D3.1),
- the WP4 analysis algorithms (the focus of this document).

The design of the different analysis algorithms in WP4 is related to the different use cases, but whenever possible, re-utilization of the different algorithmic techniques will be envisaged. In this sense, the analysis algorithms should be designed and implemented as analysis modules, which will expose different functionalities to its final user, i.e., the Reasoner, which will be further investigated in D4.2.

Therefore the structure of this deliverable is as follows. The next chapter will describe the analysis algorithms associated with each and every mPlane use case, while some other more generic analysis algorithms which are applicable to several use cases, possibly beyond those considered in mPlane right now, are postponed to a subsequent chapter.

2 Analysis algorithms associated with mPlane use cases

Each use case is the topic of one section, and it is described according to the following structure: first, a brief **introduction to the use case** is provided, with an emphasis on the splitting of its functionality in the various mPlane layers: probes used (WP2), stored and pre-processed data (WP3), and need for further analysis (WP4). The goal is to provide a high level description of the steps to solve the problem addressed by the use case, matching the mPlane layered structure. Then, a **description of the analysis algorithms** themselves is provided. Finally, and only in those cases where available, **preliminary results** showing the application of the corresponding algorithms are included. Note that at this stage, the preliminary results presented in this deliverable are not strictly considering the complete integration of the mPlane, but rather showing the feasibility and applicability of the designed algorithms.

This is the main focus of this document. As described in the Introduction and in the DoW, the algorithms in WP4 are used to dig into the data gathered and pre-processed across the lower layers of mPlane (both WP2 and WP3).

The complexity of the WP4 algorithms depends on the particular use case they are intended to address, but in general terms, WP4 algorithms are more specialized and have a broader picture of the problem they are tackling than in previous analysis stages performed either locally at WP2 probes or globally at WP3. For example, let us consider the interaction between WP3-type data analysis and the analysis done by WP4 algorithms; while WP3 performs basic statistical data analysis on large amounts of historical and highly distributed data coming from the WP2 probes, WP4 algorithms access only a reduced part of this pre-processed data for more elaborated analysis (e.g., analyze all those users who have sent more than N packets in the last hour). Similarly, while probes at WP2 perform local analysis at the vantage points where they operate, WP4 algorithms combine and correlate the information coming from multiple of these probes and at different times to get more clear and global view of the analysis.

The algorithms must have a clear spatio-temporal notion of the problem they address, as they are capable of merging the local real-time analysis performed by the probes of WP2 at different physical locations with the historical global data analysis performed at WP3.

Data mining, machine learning, statistical inference, signal processing and information theory approaches are the bases of WP4 algorithms. These algorithms provide different degrees of insight based on the quality and type of the data they use. For example, limited insights if only coarser-grained, fewer dimensional data is available, and more accurate and detailed insights if richer, more relevant and finer-grained data is used. The insights provided by WP4 algorithms serve as input to the Reasoner, which pushes forward the following steps of the iterative process.

For each use case, we provide a clear specification of the relevant analysis algorithm, as well as of its input (i.e., the link with the storage layer and possibly the use of specific probes) and output.

2.1 Supporting DaaS troubleshooting

2.1.1 Use case introduction

Running desktop-as-a-service (DaaS) solutions in remote data centers is an emerging means of delivering virtual PCs in an inexpensive, secure, and easy-to-maintain way. The fact that such solu-

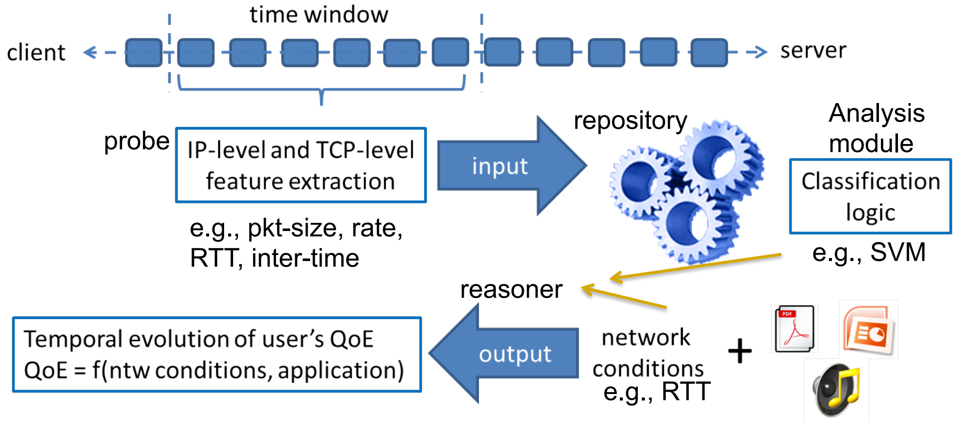

time window
client
server
probe
IP-level and TCP-level feature extraction
e.g., pkt-size, rate, RTT, inter-time
input
repository
Analysis module
Classification logic
e.g., SVM
reasoner
Temporal evolution of user's QoE
QoE = f(ntw conditions, application)
output
network conditions
e.g., RTT
+

Figure 1: Detecting the QoE of users accessing content using Desktop-as-a-Service solutions through thin-client connections: overall schema in the mPlane context.

tions rely on the presence of connectivity between users and their virtual PCs poses a challenging operational question that mPlane can address: *what is the quality of experience of the user when running a particular application over the thin-client protocol?*

The goal of this scenario is to exploit mPlane to understand whether the path between the thin client and the remote server has enough resources to sustain the rendering of the specific application. The idea is to use mPlane to collect and correlate information about the application being requested and the available network resources, in order to infer the quality of the service as perceived by the end-user and to pinpoint the reasons of any performance degradation.

Here follows the overview of the end-to-end process taking place within the mPlane stack to detect the Quality of Experience of users accessing content using Desktop-as-a-Service solutions through thin-client connections.

An mPlane probe continuously and passively collects features that can be accessed from the thin-client connection while it is running. The idea is that each probe monitors all the thin-client connections that are happening over time and collects the features needed to infer the application running on top of them, and eventually the QoE perceived by the users. As thin-client connections usually make use of encryption to protect the content being exchanged, the mPlane probes collect IP-level features of the packets of the connections, such as packet size, rate, inter-arrival time, and TCP-level features such as payload length and number of observed packets, whether they carry data or acknowledge only, TCP flags, etc. These features 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 implemented on the Reasoner 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.

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 mPlane probes, which periodically send them to the central repository.

The overall schema of the process is outline in Figure 1: the role of each mPlane component, that is, probe, repository and reasoner, is shown.

2.1.2 Description of the associated analysis algorithms

To detect the application on top of a given thin-client connection, we consider the use of statistical classification techniques. The main goal here is the design and tuning of an effective statistical classification technique which can effectively take advantage of the available features provided by the mPlane probes.

We plan to evaluate the accuracy of several supervised statistical classification techniques widely-used in the traffic classification field [84], such as Support Vector Machines, Random Forest, Naive Bayes and Decision Tree, in detecting the applications running on top of the thin-client connections.

Supervised approaches require a training phase before being exploited for classification. During the training phase, a supervised machine-learning algorithm requires as input a collection of samples for each class of interest. Starting from these samples, the algorithm extracts the features needed to build a mapping function between the samples themselves and the class they belong to. In our case, the class of interest is a particular application running inside the thin-client connections, or a class of them thereof, such as *Data*, *Audio*, *Video*; the features will be instead the ones collected by the mPlane probes, and stored in the repository, from which the Analysis module can query them.

During the classification, the features related to the thin-client connection will be tested against the training models and the connection will be labeled as belonging to one (or none) of the available models.

Given the class of applications run by the thin-client user, the Reasoner has to combine the average Round Trip Time of the connection within an observation window against a set of threshold values, and returns a QoE category. Threshold values are set for each class of applications, i.e., *Data*, *Audio*, *Video*, and are based on latency values: network latency is the key factor that affects the QoE of users when they interact with thin-client applications, as shown in [73]. As a result, for each class of applications we are able to identify requirements in terms of RTT values that make the users experience a good, sufficient or bad quality of the RDP connections.

Once the application currently running on top of the thin-client connection is known, the QoS requirements are evaluated with respect to passive measurements like RTT, available capacity on the path, packet loss rate, etc. When one of the above observations exceeds a certain threshold, additional measurements are triggered to identify the bottleneck along the path. Furthermore, the mPlane reasoner can react to these conditions, for instance by migrating the virtual machine where the cloud service runs closer to the user.

Information about the bottleneck along the path can be then derived on demand via an active measurement tool (like a traceroute) or via the mPlane Repositories (that continuously stores information about the network status in various segments of the network). Such sequential measurements can be iterated until the root cause of the problem is identified.

2.1.3 Preliminary evaluation

As a first evaluation of these methods, we started collecting an initial dataset of Remote-Desktop-Protocol (RDP) connections and investigating how the techniques perform when the dataset used for training and the one used for testing include the same class of applications and are collected under the same network conditions. To this purpose, we considered as testing set the data collected with a bandwidth of 1Mbps uplink and 6Mbps downlink and no impairments on the network by running the same class of applications as the one included in the training set, such as web-browsing, media-player video and audio data.

Given an epoch, we observe the traffic flowing into the RDP connection, extract the features for each epoch and classify it to the class of application it belongs to. By considering time-windows (epochs) of 10 seconds, we achieved maximum accuracy (over 90%) when applying SVM, whereas we achieved around 85% with Decision Tree and Random Forest, and 78% when using Naive bayes.

2.2 Estimating content and service popularity for network optimization

2.2.1 Use case introduction

The capability of estimating the future popularity trends of services and contents (both managed and user-generated) has a wide range of applications. For instance, operators could optimize their resource management to improve QoS, as well as CDN providers could design smart caching mechanisms or improve the spatial distribution of caches within the ISP network (e.g., in the aggregation network [12] or in the user set-top-box [68]).

Currently, very simple yet unrealistic workload models are used: rather generally indeed, the object catalog is stationary, and additionally the popularity of each object is stationary as well. Given the importance of video on today Internet, we clearly see that this is clearly not in line with the typical catalogs of popular Internet movie-streaming platform (e.g., Netflix) or video-on-demand portals (e.g. YouTube), that significantly evolve over time (e.g., see [12] for an analysis of the Neflix dataset). Hence, as stationary workloads currently in use are unrealistic, there is need for more realistic models. While trace-driven evaluation (e.g., as we use for the assessment of caching performance of the Netflix [12] and YouTube [68] catalogs) constitutes a first step in this direction, modeling of the system dynamics would offer the community a more powerful tool. A non-stationary workload model, fit on real traces, would not only be extremely useful for realistic performance evaluation (being simpler to share than actual datasets for privacy reasons, and due to the sheer size of the dataset) but also possibly enhance system performance (e.g., by allowing to exploit forecast of popularity evolution). Having a general methodology that is not bound on the specific application (e.g., video as in the previous example), will allow mPlane solution to be useful even in case of changes in the Internet ecosystem.

The goal of this scenario is to employ mPlane to monitor the arrival process of requests towards the services and the contents and store this information to run predictive algorithms. The architecture of mPlane perfectly fits for this process, since we aim to collect large amounts of traffic information such as the contents/services being requested by the users in several points of the network and produce a list of the most likely to be popular in the future.

Here follows the overview of the end-to-end process taking place within the mPlane stack to estimate the popularity trends of contents by a passive observation of the network traffic.

Several mPlane probes are located in different points of the network to continuously and passively collect information about contents and services being accessed from users. As web contents are

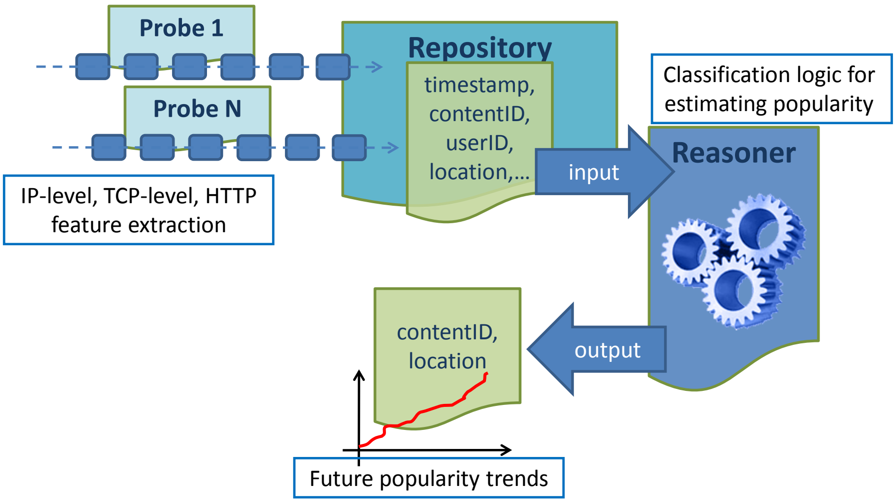

Probe 1
Probe N
IP-level, TCP-level, HTTP feature extraction
Repository
timestamp, contentID, userID, location,...
input
Classification logic for estimating popularity
Reasoner
output
contentID, location
Future popularity trends

Figure 2: Predicting the popularity trends of contents by a passive observation of network traffic.

usually delivered through HTTP, the probes collect traffic at transport layer (TCP-level), so that we can easily rebuild HTTP conversations between users' devices and the servers hosting the requested contents. The probes are programmed to generate a time-series of requests, reporting the content being requested, a timestamp and the network location of the probe.

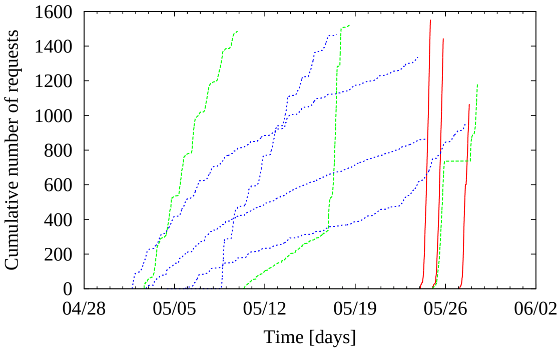
At regular time intervals, say every few minutes, the probe sends the extracted information to the central repository, which stores them for the Predictor module to use. The Predictor module implemented on the Reasoner extracts the arrival process of requests for each observed content and separately for each probe location. Then, employing statistical classification techniques, request arrival processes are grouped to retrieve their possible future growth rate.

Finally, the Predictor generates a list of contents that are likely to become popular in the future, separately for different portions of the network. In Figure 2 we report the overall prediction process. The role of each mPlane component, that is, probe, repository and reasoner, is shown.

2.2.2 Description of the associated analysis algorithms

Conceptually, there are three component steps to understanding and estimating the future popularity of contents. In order of appearance, we must first collect a sufficient sample of the data in order to be able to build a feature space that we can characterise. In effect we build a training set. Although the term a 'training set' implies that we are focused on supervised learning techniques alone, this is not in fact the case. Having a sufficient training set means sampling the problem space in order to be able to characterise it. Second, we come to the characterisation, although this may be merged with the next stage of estimation, conceptually here, we are focused on identifying the patterns of popularity evolution displayed by the data. Identifying such patterns enable us to relate new and so far unseen objects to past observations. Finally, estimation, here we are interested in ascribing a value to the future popularity of content, based on the understanding we have built of its growth process.

Within mPlane, we are interested in applying data driven techniques to understand the how users' consume content, in affect the evolution in the demand for individual contents.



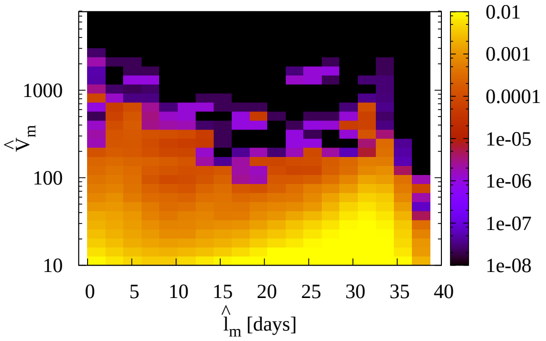


Figure 3: Cumulative number of requests over time for a subset of videos observed in a traffic trace spanning one month of 2012; only the requests within the life-span of the content are shown.

Figure 4: Density map of contents with $V_m \geq 10$ observed in a traffic trace, based on estimated number of requests V_m and effective life-span l_m.

2.2.3 Preliminary classification approach

With respect to this, we analysed a trace of passive traffic measurements, and we characterised the popularity profiles of YouTube videos observed in the operational network of a large Italian ISP. [1]

From Fig. 3, we see that YouTube videos display extremely heterogeneous request distributions and exhibit strong time-localities. For instance, we observe that the popularity of some videos (red lines in the plot) vanishes after only a few days - these are usually videos belonging to the news or sport categories -, while others (green lines in the plot) continue to attract requests for almost the entire duration of the trace - such as videos belonging to the music category -, reflecting the diversity in user interest. As a result, to capture the evolution of content popularity over time, we focus just on the this cause, and characterise each individual content object m with the following two parameters:

- The total number of requests V_m generated by the content.
- The effective life-span l_m of the content, which is defined as the duration of the interval in which we see the bulk of its requests.

The density map in Fig. 4 reveals that, as expected, contents exhibit wide heterogeneity in terms of estimated life-spans l_m and estimated volumes V_m. The map also shows that there exists a peculiar correlations between l_m and V_m. This suggests that a traffic model should consider the joint distribution of these metrics. In fact, from the results, we observe that a non-marginal share of videos (7-10%) exhibit a very small life-span ($l_m \leq 5$ days), while 2% of videos have $V_m \geq 10$, but account for a share of requests that is greater than 27% (these results hold for all the traces considered in the data set analysed in [80]). These two observations should be carefully taken into account during the classification process.

[1] Measurements were collected on both incoming and outgoing traffic carrying YouTube videos for a period of three months, from mid March 2012 to late May 2012. During this period, we observed the activity of more than 60,000 end-users accessing the Internet normally.

Class	Life-span [days]	Trace	%Reqs	%Videos	$E[l_m]$	$E[V_m]$
Class 1	$\hat{l} \leq 2$	Trace 1	9.15	3.17	1.14	86.4
		Trace 2	10.05	4.17	1.09	76.2
		Trace 3	9.44	3.73	1.04	76.0
		Trace 4	7.77	3.34	1.06	74.0
Class 2	$2 < \hat{l} \leq 5$	Trace 1	6.80	4.9	3.36	41.9
		Trace 2	12.55	7.83	3.34	50.7
		Trace 3	6.55	4.54	3.32	43.3
		Trace 4	6.12	4.06	3.41	48.0
Class 3	$5 < \hat{l} \leq 8$	Trace 1	5.87	2.95	6.40	59.5
		Trace 2	6.72	4.74	6.31	44.9
		Trace 3	6.05	2.87	6.42	63.3
		Trace 4	5.14	2.71	6.45	60.3
Class 4	$8 < \hat{l} \leq 13$	Trace 1	5.49	4.45	10.53	36.9
		Trace 2	10.79	8.61	10.86	39.6
		Trace 3	4.84	3.68	10.62	39.5
		Trace 4	5.34	4.48	10.65	37.8
Class 5	$\hat{l} > 13$	Trace 1	72.69	84.58	24.61	25.7
		Trace 2	59.89	74.65	19.29	25.3
		Trace 3	73.11	85.17	28.19	25.8
		Trace 4	75.63	85.41	24.59	28.1

Table 1: Observed features for different classes of contents.

It is also worth emphasising that the two parameters V_m and l_m alone do not completely characterise the temporal evolution of content popularity, which as shown in Fig.3, can exhibit complex growth patterns. In fact, recent studies [86, 56, 5] conducted on much larger data sets reveal that the popularity of different contents, including videos, follow a limited number of "archetypal" temporal profiles, which essentially depend on the nature of the content and on the way it becomes popular. However, as shown in [80] specifically for the case of CDNs, the single cache performance is essentially driven by the parameters V_m and l_m, while the shape of the popularity profile has only a second-order effect.

In Table 1 we report the result of a time-based approach to classify YouTube videos: considering all the traces available in our data set, for each observed content we measured its life-span l_m and the number of requests V_m it attracted, i.e. features fairly easy to extract from a passive traffic trace. We thus divided contents by looking at their life-spans. Observe that this classification has been performed for all contents showing $V_m \geq 10$. For each content class, Table 1 reports the percentage of total requests attracted by the class, the percentage of contents belonging to it, and their average estimated values V_m and l_m. Notice from Table 1 that contents in Class 1, whose life-span is smaller than 2 days, represent less than 4% of the total number of contents but attract approximately 10% of all requests. Therefore, referring to the context of CDNs, since these contents exhibit a large degree of temporal locality, they can be expected to have significant impact on cache performance. Observe also from Table 1 that the values related to each class are quite similar across the four traces (within a factor of 2). This is significant, because it suggests that this broad classification captures some invariant properties of the considered traffic. Further details about the classification approach and employed traffic traces can be found in [80].

2.2.4 Advanced classification approaches and popularity prediction

Above results about the classification of contents are preliminary for our objective of exploring a range of more sophisticated statistical classification and inference techniques (i.e. mixture modeling, and decision trees) in order to identify underlying patterns of growth, and use this information to build profiles that can be utilized by the Reasoner to classify and estimate the future popularity of contents.

Given the popularity profiles of each request arrival process, the Reasoner has to combine the future expected number of requests against a set of threshold values, and returns a popularity category. As a result, the Reasoner will pinpoint those contents and service which are expected to become hot in the near future. Once the Reasoner has retrieved the required popularity information for each observed network portion, it can, for instance, directly contact the CDN provider to suggest contents to be pro-actively pushed to the caches, thus improving the QoE perceived by the users.

Furthermore, given the wide range of services and web contents available in the current Internet, the Reasoner must be able to automatically detect upcoming trends in order to evolve with traffic profiles observed. For this purpose, the repository should support storing long term statistics regarding the popular of contents and services observed in the network.

2.3 Passive content curation

2.3.1 Use case introduction

Content and media curation is the act of using both automatic and human resources for the purpose of aggregating, sorting, organizing and presenting only ``interesting'' content to end-users. The rationale behind its advent is that the Internet today contains much more content than what humans can individually consume and sort by themselves. We, as users of the Internet, need therefore tools that ``cure'' the content for us so that we see preferably the content that is more likely to interest us. Examples of such tools include Reddit [67], Digg [30] and Pinterest [64]; even social networks like Twitter can be themselves seen as media curation tools [48].

A family of curation tools uses mainly the so-called ``wisdom of the crowd'' to present and select interesting contents. The rationale behind this approach is that the collective answer of a large group of individuals to a given question is often better than the individuals' responses. With this respect, we argue that ISPs are better positioned to provide such service thanks to the Internet-wide view that they have on the traffic.

By ``simply'' monitoring content requests that flow within their networks, ISPs can infer the content that is capturing the largest interest. By tracking trends and with appropriate models, it would be even possible to predict early enough which content will become popular, and therefore present it early enough to users. Such a service can be done passively without user engagement, and in an aggregate manner, thus not offending users' privacy. In fact, we assume that if a content (e.g., a URL) is visited, it is because it captured some user's interest. The more a content is clicked, the higher the chances that it deserves to be watched. Finally, unlike current curation systems whose results can be manipulated by a small community of users [61], passive media curation is by design more immune to such biases.

Fig. 5 gives an overview of the media curator achitecture, our ISP-provided content curation system. At the bottom we have ``probes'' installed at network level that extract requested objects. The requests from different probes are aggregated and sent to a content filtering and popularity analysis module which elects the most popular contents, and sends the rankings to the presentation module which will make these content items (pointers to them) available online.

The raw data extraction module exposes the following information about content requests: probe location, timestamp, URL, an anonymized user id, as well as the *referrer* and *user agent* fields ex-

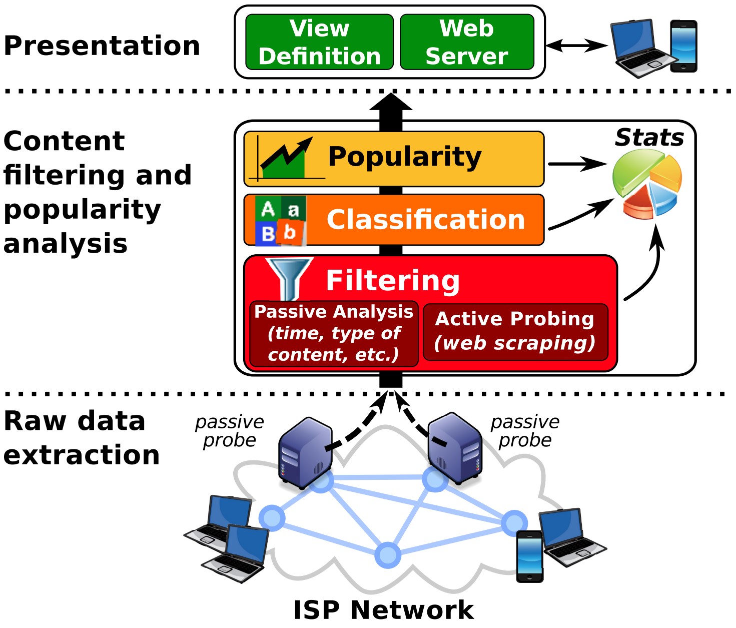


Figure 5: An overview of the content curation system.

posed in the HTTP requests.

The content filtering and popularity analysis module, de facto the core of this system, processes the data generated by the content requests extractors. As an output, it provides a list of ``hot'' *contents*, e.g., URLs pointing to pages that are likely to interest Internet users. Basically, it builds ``popularity'' measurement to rank objects. This module can work in both push and pop mode. It first can push alarms about content items which are likely to become hot. It can also respond on demand to provide statistics about the history of popularity. As such, it should be able to return the list of the most popular contents in a given period of time.

This module consists of five sub-blocks as depicted in Fig. 5. The first block applies *filtering* to discern relevant URLs. In order to check their relevance, this block might need to actively query some specific URLs. This can be done thanks to an *active probe* sub-block (i.e., a web scraper). The filtered URLs are then analyzed to predict their future popularity trend: first, contents are divided in categories by the classification sub-block. The role of this block is to cluster the filtered contents so that different *predictors* blocks can run on them. Finally, a last sub-block will take care of providing statistics about the history of popularity of observed content.

The output of the content filtering and popularity analysis module is then provided to the presentation module whose aim is to present the popular contents, gouping them by topic. The presentation module also runs similarity detection algorithms to group together different URLs pointing to the same news or object. Finally, it uses a web scraper to query URLs to offer a preview of the content to be shown on the frontpage.

In the following we describe the algorithms which compose the content filtering and popularity analysis module.

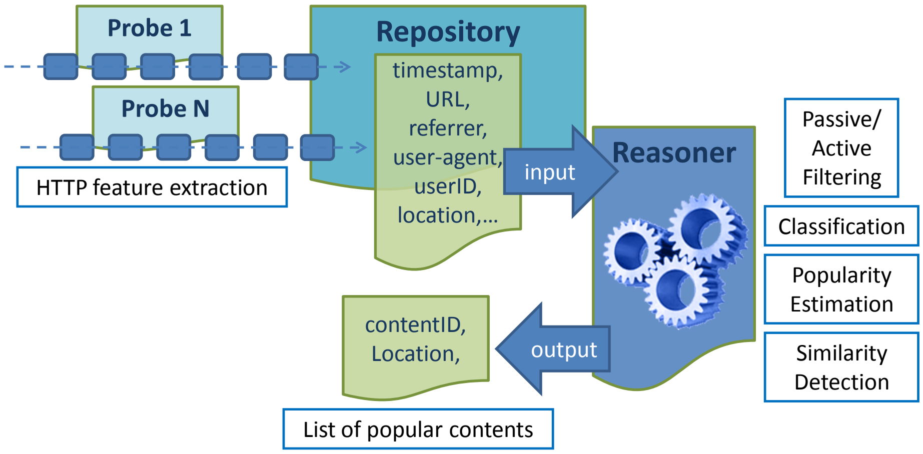


Figure 6: Extraction and curation of contents flowing on the network.

2.3.2 Description of the associated analysis algorithms

Here we present the algorithms that constitute the content filtering and popularity analysis module, i.e., the core of the media curatordesign. The ultimate goal of this module is to provide a ranked list of the most popular content observed in the network. We consider as content the URL of a web page that was visited by a user.

2.3.2.1 Identification of user-URLs

First of all, most of the HTTP requests flowing in a network are not relevant for our system. For instance, many URLs point to different objects such as CSS and javascript files. These are not the type of contents media curatorshould promote. Therefore a crucial step in the content popularity analysis module is the filtering of irrelevant URL requests. We distinguish between two types of URLs: *browser-URLs*, i.e., HTTP requested objects that are part of a web page and thus transparently downloaded by the browser; and *user-URLs*, i.e., URLs of pages intentionally visited by users. media curator seeks only user-URLs.

The first step in the filtering is to distinguish browser-URLs from user-URLs, i.e. contents that have been automatically required by the browsers from those that have actually been clicked by the user. Several methods have been proposed in the literature [23, 11, 43]. The most accurate for nowadays web traffic is the one presented in [43]. However, its methodology needs to monitor HTTP requests and corresponding responses, which dramatically increases the complexity of the content extractor module. Besides, this method does not work for all web content, but only on pages that contain the google analytics beacon. According to the authors [43], this beacon is present in only 40% of the pages.

Therefore, we believe that new and simpler methods must be devised to identify user-URLs. As a design choice, we rely only on the parsing of only HTTP requests (and not responses) to reduce the algorithm complexity.

In this context, we devised several filtering mechanisms that we are currently studying. We enu-

merate them as follows.
1) Referer-based filter: This method exploits the complex structure of web pages and the *referer*[2] field. When the URL of a web page is clicked, a sequence of HTTP requests is generated by the browser to retrieve all the objects that are necessary to render this web page. As such, all these object URLs have, as a referer the first clicked URL. Therefore, by focusing on the *referer* field instead of requested URL, we are sure of capturing the original user-URL. In reality, this method captures all complex web objects whose loading requires the browser to load other objects (e.g. ``complex'' css files calling other css files).

2) Type-based filter: This method is similar to [23]. It filters out URLs based on their type. However, instead of relying on the content type, we inspect the extensions of the objets queried by the HTTP requests: in particular, we discard URLs pointing to *.js*, *.css* and *.swf* files.

3) Ad-based filter: This approach relies on the observation that a large amount of advertisement is nowadays embedded in web pages. Unfortunately, since advertisements can be complex html objects, they can be detected as user URLs using the refer-based filter above. To counter this phenomenon, we blacklist such URLs (using the AdBlock filter [4]). This method has however, the counter effect of removing advertisement that were actually intentionally visited by the users.

4) Children-based filter: By counting the number of URLs seen with a given *referer* URL, it is possible to know the number of objects (children) composing the corresponding parent URL (in the referer field). Given the tendency of today's complex web pages to include a large number of objects, we can safely filter out URLs that have a very low number of children, e.g., simple objects that include only few other objects.

Finally, in our work, we also test the *Time-based filter* [11, 43]. This method relies on the intuition that a cascade of browser-URLs always follow the initial user-URL: browser-URLs would be tightly grouped in time after the user-URL request has been observed. As such, the first request after a given inactivity period is considered to be the user-URL, while all the following HTTP requests that come after a given threshold are assumed to be browser-URLs. Both the inactivity period or the threshold are clearly dependent on user habits (e.g. opening multiple tabs at the same time), the network conditions and DNS response times etc. They are as such difficult to estimate.

2.3.2.1.1 Preliminary evaluation To evaluate these methods, we manually collected, similarly to [43], a ground truth trace by visiting URLs hosted by the top 100 most popular web sites according to Alexa ranking. In each of these sites, we randomly visit up to 10 links contained in them. We store all the clicked URLs as they are shown in the browser bar. In parallel, we capture all HTTP requests generated by the browser. This resulted in a total list of 905 user-URLs, corresponding to 39025 browser-URLs.

We start our analysis by applying the referer method first. Fig. 7 depicts the resulting Venn diagram, which shows that the referer method alone has a high ``filtering capacity''. In fact, it reduces the set of browser-URLs from 39025 URLs to only 2616 URLs. Besides, it has a high recall[3] (98,34%). How-

[2] A field in the HTTP request that specifies from which previous page the URL has been referred

[3] The number of URLs correctly labeled as user-URLs (true positives, here 890) over the real number of user-URLs (ground truth, here 905).

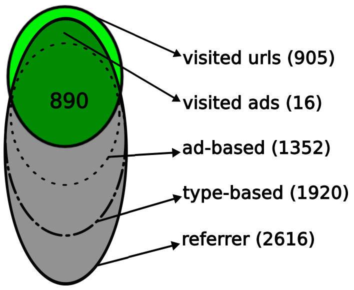


Figure 7: Venn diagram reporting the effects of different filtering methods on our ground truth data set.

Method (applied **after** referer)	Recall	Precision
Time-based (0.01s)	97,90%	37,67%
Time-based (0.1s)	96,13%	41,09%
Time-based (1s)	87,51%	55,15%
Children (remove <= 1)	94,8%	43,13%
Children (remove <= 2)	93,14%	49,76%
Type-based	98,34%	46,35%
Ad-based	96,57%	44,14%
Type-based + ad-based	96,57%	66,41%
Type + Ad + children-based (remove <= 1)	93,14%	72,67%

Table 2: Performance of filtering methods

ever, its precision[4] is low (34%). Therefore, to reduce the set of false positives, we apply our other methods on top of the URLs detected by the referer method. We also consider different thresholds. The results are summarized in Tab. 2.

Both the time-based and the children-based methods enhance the precision when we increase respectively the time threshold and the minimum number of children. However, they come at the cost of a decreasing the recall (invalidating valid visited URLs). On the other hand, the type-based and the ad-based let us enhance the precision, with almost no impact on the recall. In reality, the ad-based filter removes 16 valid user URLs, but these were unintentionally visited when we collected the trace[5].

As a start, we opt for the most conservative approach that favors recall over precision: we retain the type-based coupled with ad-based filters applied after the referer method. We leave their enhancement for future work.

2.3.2.1.2 Additional problems/filtering

When trying to apply the above-mentioned filtering mechanisms on real data, we noticed some additional sources of error that called for more ingenuity. In particular, we encountred two main problems; we propose two simple mechanisms to factor them out:

i) URLs generated by non-browser applications: Most applications use HTTP to automatically download content, e.g., Dropbox or games on smartphones. The queried URLs are clearly not interesting

[4] The number of true positives over the number of items labeled as positive by the method (here 2616).

[5] Although we acknowledge that some advertisements might interest people, we decide to skip them for now.

for media curator, and must be ignored. We can easily identify those URLs by inspecting the *user-agent* field[6] in the HTTP request header.

ii) Inflated popularity induced by some users: Sometimes browsers generate multiple HTTP requests for the same content, e.g., automatically reloading a page, or downloading videos in chunks. This phenomenon inflates the popularity of some URLs. We counter this effect by counting a URL only once for each user-id.

2.3.2.2 Pinpointing Interesting URLs

Applying the retained filtering method on our data set, and looking manually at the most popular URLs, we found that they still do not correspond to what we expect the media curator portal to promote. In fact, among the popular user URLs, we find popular web pages that might not interest users (e.g the web portal of an online bank). Therefore, the next step in the filtering is how to go from user-URLs to *interesting URLs*, i.e., URLs that are likely to attract users' attention.

Finding a measure of interest is challenging since it involves human subjects and tastes. We develop a preliminary simple heuristic that leverages online social network meta-information. We assume that interesting URLs should be rich with ``social'' features (e.g., share buttons). The idea is that if a web page is meant to be shared, then it might interest other people. Based on this rationale, we propose two methods to understand which user-URLs are ``social-networks enabled'':

1) Active method: In order to know if a user URL is as well an interesting URL, this method actively queries the URL and parses the returned HTML header looking for the presence of the OpenGraph protocol[7] [63]. If the protocol is present, the user URL is classified as an interesting URL. This method is meant to use the web scraping capability of the content popularity analysis module.

2) Passive method: This approach aims at passively detecting if a web page contains any of the well known social networks buttons. We inspect a user-URL children (URLs seen with the user-URL as a referer) and match them against a list of URLs necessary to load such buttons. We study the different social networks web development guidelines to construct such a list.

2.3.3 Preliminary evaluation

As a first evaluation of these methods, we test if they already work on platforms that are known to be interesting. In particular, we test them against Google News. For this purpose, we visit 1000 URLs promoted by Google News. Applying the active and the passive methods on this trace, we find that they classify as interesting respectively 79% and 70,72% of Google News URLs. By inspecting manually the misses, we find that (1) not all the web pages implement the OpenGraph protocol and (2) there are ad-hoc ways of implementing the social networks buttons that our list fails to identify. In particular, YouTube uses an ad-hoc solution, but it is OpengGraph compliant, so our active method correctly detects it.

[6] The user-agent informs about which application generated the HTTP request

[7] The OpenGraph protocol was developed by Facebook to help web pages getting integrated in ``the social graph''. The OpenGraph metadata in web pages helps the social web scrapers (e.g. Facebook scraper) forming a preview of a page when it is shared in a social network.

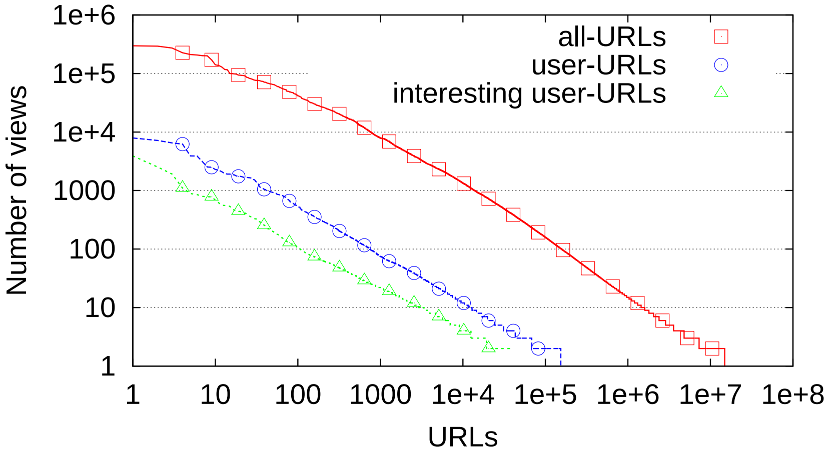


Figure 8: Content items popularity before and after filtering the considered traffic trace.

Given the early stage of development of our heuristics, these results look promising, and we leave their enhancement for future work. Finally, in order to have a preliminary estimate of their ``filtering capacity'', we apply them on the more neutral (not necessarily entertainment) Alexa ranking artificial trace. Both methods gave similar results: 33% of URLs were labeled interesting using the active method, and 35% using the passive one.

2.3.3.1 Ranking and Classifying URLs

Once contents have been properly filtered, the next step is to run popularity prediction algorithms to retrieve, early enough, content items that are most likely to attract users' attention. With this respect, the algorithms presented in Sec. 2.2 will be of a great help.

As shown in Fig. 6 the output of this final process will be a list of the hottest contents observed in the network. This list can be further personnalized and provided on a per-region basis, by ``zooming'' on particular probes in the network.

To get an idea about how such a ranking would look like and how our retained filtering algorithms would behave on real data, we apply them on a three-days long http trace. The trace contained around 190 millions of HTTP requests.

Fig. 8 shows the popularity of all-URLs, user-URLs and interesting user-URLs. As expected, the user-URLs represent a tiny fraction of all the URLs flowing in the network: our method detects that only 1% of all-URLs are actual user-URLs. Among these user URLs, around 25% were detected as interesting.

We analyze the interesting user-URLs. We first manually classify the top 1000 URLs to build a set of rules to help us having a preliminary classification of the rest of the URLs. Among the top 1000 URLs, we find that 482 are news (or blogs), 336 are services (e.g., online shops, travel engines, etc.) and only 91 were videos. Extending a similar classification on the rest of the interesting user-URLs, we find that (at least) 18% correspond to Video coming from 9 different platforms (YouTube alone 15%) and 22% correspond to news coming from around 80 different news web sites. This observation confirms that the ISP is probably the only entity that has such a rich cross-OTT view on Internet content.

2.4 Measurements for multimedia content delivery

2.4.1 Use case introduction

Evaluating the quality for multimedia stream delivery is important for

1. ISPs who are interested in assessing the true quality of their networks for popular applications (such as multimedia streaming)
2. Streaming service (infrastructure) providers who are interested in knowing the quality of their services as delivered by ISPs to different locations over different access technologies

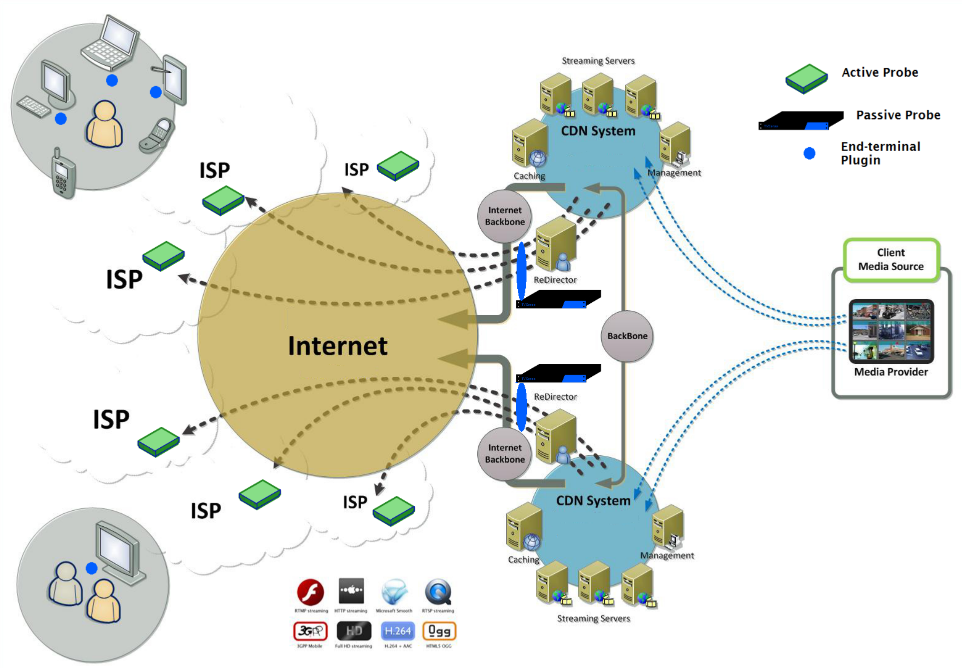


Figure 9: Architecture for monitoring the multimedia service infrastructure

In both cases, active testing substitutes or augments monitoring performed at ``real'' end-user terminals. Performing end user measurements in many cases is problematic - among the reasons are users' unwillingness to participate, lacking support for various end user devices by the measurement platform, or the unability to install additional measurement plugins for administrative reasons etc.

Active probes on the other hand are deployed at known (key) locations, are able to perform measurements any time, and can be configured to receive and measure any of the streams carried by the network. Active probes are capable to evaluate *adaptive* streaming protocols, such as Apple HLS or Microsoft Smooth Streaming: these protocols provide multiple, different quality (i.e. bandwidth) alternatives for each stream. These protocols also provide means for smooth (i.e. flicker-free) switching between these alternatives, thus a player can dynamically set the stream quality to the momentary bandwidth conditions. Support for smooth switching is usually achieved by encoding the stream into time-synchronized equal-duration *segment sequences* of different bitrates

- reaching the end of a segment the receiver can continue by playing out the next segment from a different-bitrate alternative. The active probes discussed implement buffered player emulation, that is able to perform this adaptation and measure the achieved quality (bitrate).

Also in both cases introduced above, active measurements should be supplemented by passive measurements performed by high-performance passive (TSTAT) probes deployed at stream service entry points. For an ISP, it is the peering point with the content provider, and for the infrastructure provider it is the point(s) where the streams leave the server segments, typically implemented by HTTP load balancers. TSTAT probes log each TCP/HTTP/Stream delivery session with metrics such as total duration and bitrate, number of TCP retransmits, stream metadata (e.g. video format, depending on depth of inspection implemented). TSTAT probes can also measure generic, yet insightful, quality of service metrics (QoS) such as queueing delay, that can provide important hints on users' perceived quality of experience (QoE) through perceptual mapping[8]. In many cases, passive probing also indicates client system type (iOS, Android etc.). As TSTAT probes have the capability to measure and log *each* media access session, they provide input for comprehensive performance evaluation.

It must also be noted (as explained in D1.1), that measuring the quality of stream delivery is not sufficient for assessing the overall quality of a multimedia streaming service: there are also a number of auxiliary services, such as index pages listing content, search functionality, and - optionally - user feedback channels (e.g. forums, comments for content etc). While scientifically not as challenging, measuring the availability and functionality of these services make the monitoring task complete. Such auxiliary service are usually monitored by automating simple HTTP download operations. In the case of interactive functions (search, feedback channels), measurements are implemented via scripted HTTP transactions.

Figure 9 shows the schematic architecture of the monitoring system.

Streaming service providers usually purchase test subscriptions from ISPs within their service area, via different access technologies, and deploy active probes to continuously monitor service delivery at each end point.

2.4.2 Description of the associated analysis algorithms

2.4.2.1 Metrics collected

The analysis algorithm works on the following metrics collected by the active probes:

If the monitoring architecture also contains passive (TSTAT) probes, their logs are available as well, and can be correlated to probe measurements.

In addition, for each active probe the following information is available:

- connections's nominal bandwidth and other deployment characteristics of concern (i.e. access technology)

- topology and further classification information for the probe (e.g. responsible ISP)

[8] Mapping between QoS metrics and QoE metrics can be done through standard perceptual framework: as QoS to QoE mapping is not necessarily bound to a single application, such as the multimedia streaming of this use case, we defer a more thorough description in Sec. 3.3

Metric	Unit	Description
nominalBitrate	bps	media segment's advertised bitrate
actualBitrate	bps	actual bitrate achieved
bandwidthUtilization	%	actual vs. nominal bitrate
qualityIndex	%	bandwidth of selected stream alternative vs. best one
chunkInterArrivalJitter	msec	jitter of L7 data chunks received
replyDelay	msec	delay of 1st chunk of the segment
bufferLevel	%	buffer level of (emulated) player
httpCode	int	HTTP status code of the last reply
protocolError	enum	protocol violaton (e.g. malformed data)
serverAddress	ipaddr	IP address of the server providing the stream
queuingDelay	msec	queuing delay, per-chunk or windowed

Table 3: Metrics collected by active probes

The task of the analysis is

- identify subsets of probes with less than adequate performance
- match these subsets against probe topology and other deployment attributes available (e.g. ISP)
- and try to conclude if the degradations observed can be attributed probe deployment information

This way, the following problems can be identified:

- network bottlenecks on behalf of ISPs
- network bottlenecks tied to congestion in the user access paths
- regional problems caused by CDN mis-configuration or lack of servers
- less than adequate service coverage (group of endpoints with too little bandwidth to get streams)

2.5 Quality of Experience for Web browsing

2.5.1 Use case introduction

Surfing the Internet via a Web browser is the most common way of accessing information. When clicking on a Web page, the user expects that the page gets rendered quickly, otherwise he will loose interest and may abort the page load. However, in reality, multiple causes can affect the loading speed of a Web page, causing it to be slowly or not completely loaded. This is due to the number of different actors involved in a browsing session (i.e., a time interval in which the user accesses information on the Internet via a Web browser).

In Figure 10 we can identify 7 different elements which can cause a Web page to load slowly:

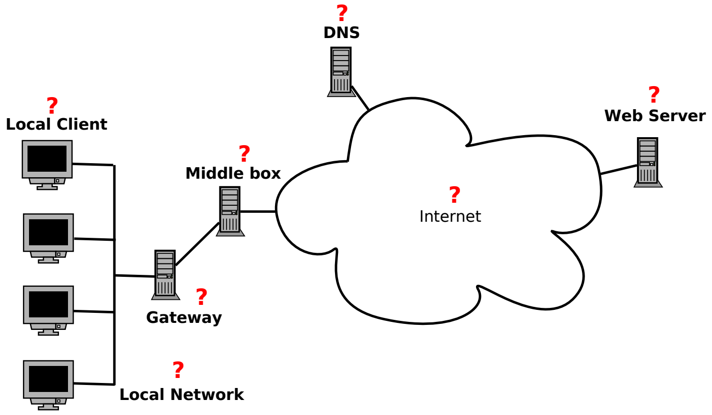


Figure 10: Segments involved in a browsing session.

1. Local Client: e.g., the local machine is overloaded, misconfigured, and so on;
2. Local Network: e.g., the local home network can be experiencing heavy traffic load due to other local clients;
3. Gateway: e.g., the gateway can be misconfigured;
4. Middle Box: e.g., firewall, NAT, ...;
5. DNS: e.g., failure to resolve properly one or more of the names referenced in the Web page;
6. Internet: e.g., something is wrong ``in the wild'', such as high loss rate between certain intermediate routers;
7. Web Server: e.g., the Web server serving the Web page is overloaded and will return its content slowly.

From the end user point of view, the result of a problem in any of the aforementioned segments will result in a poor quality of experience (QoE), being the Web page slowly loaded.

2.5.2 Description of the associated analysis algorithms

2.5.2.1 Measurement Metrics

The analysis algorithm takes as input data taken from the following metrics:

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

Table 4: Measurement Metrics

- Client idle time: a passive measurement representing the idle time periods while downloading a Web page;
- Total Web page downloading time: passively collected during the browsing session;
- DNS response time: the difference between the time at which the DNS request is sent and the time at which the response is received (passively collected);
- TCP response time: the time needed for receiving the TCP ACK packet corresponding the first TCP SYN packet sent (passively collected);
- HTTP response time: the time needed for receiving the first data packet corresponding the first http GET packet sent (passively collected);
- RTT towards a given node, collected by actively running a ping towards the node of interest, when a problem signal is raised by the user;
- Δ^i: the difference between RTTs towards consecutive routing hops $i-1$ and i (computed).

All the described metrics can be simply obtained by using the following tools:

- Tracebox: proposed in [3], it is used to detect the presence of middle boxes in the network
- Traceroute: used to find out the sequence of the first hops made by a packet when leaving the local network
- Firelog: this tool, that was first proposed in [26], is used to perform all the passive measurements

All the metrics are summarised in table 4, where we also report the tools that can be used to collect them.

As explained in the next section, the metrics in table 4 are used to update cumulative statistics that are compared, in case a problem signal is raised, to an appropriate threshold to verify if the metric has some ``anomalous'' behaviour. For instance, measures such as queueing delay experienced by Web traffic (e.g., due to competing traffic in the user LAN) can be inferred from simple RTT measurement: though generic, such a metric can provide insigthful hints about the users' perceived quality of experience (QoE) through perceptual mapping[9].

[9] Mapping between QoS metrics and QoE metrics can be done through standard perceptual framework: as QoS to QoE mapping is not necessarily bound to a single application, such as the Web browsing of this use case, we defer a more thorough description in Sec. 3.3.

2.5.2.2 Algorithm description

The proposed algorithm aims at identifying the segment that is responsible for the high Web page loading time.

The main idea is to try to perform as much of the troubleshooting process as possible exploiting the passive measurement [26], without requiring active measurements and by exploiting the advantages of having more than one device in the local client network.

First of all, when a user raises a problem signal, by clicking on the button provided in the Web browser, indicating that the Web page is taking a too long to load, the algorithm starts the process to locate the problem.

The first check is made on the local client, by simply checking the CPU usage of the local machine and the ratio $\frac{T_{idle}}{T_{tot}}$. If the ratio exceeds a given threshold then the algorithm concludes that the problem is in the local host.

Otherwise, if the local host does not present any problem, the algorithm performs a check on T_{http} (average value over all the values corresponding to the different objects of the loaded Web page), by verifying if the cumulative statistics applied to this metric has exceeded a given threshold. Note that this metric can be considered as a rough approximation of the time required for getting the first data packet from the remote Web server, thus in case it is normal it can be concluded that the problem is not in the network (neither local nor backbone) and neither is in the remote host.

Hence, the algorithm performs a check, at first, on the size of the Web page (verifying if the number of objects/bytes of the page exceeds a threshold) and then, in case the Web page dimension is not responsible for the problem, it checks T_{tcp} and T_{DNS} possibly concluding that the problem is generated by the long distance towards the remote Web server or in the DNS server, respectively.

Instead, in case T_{http} is normal, the algorithm automatically excludes the DNS and the page dimension cases and proceeds by requiring some cooperation from the other devices of the local network. In more details it ``asks'' the other devices to report any experienced problem.

At this point, there can be three distinct cases: all the other devices are experiencing some problems, none of the other devices is experiencing any problem, and just some of the other devices are experiencing some problems, that is:

In case all the different devices are experiencing some problems 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). Among the remaining cases (i.e., gateway, local network, middle boxes, and backbone network), the algorithm can assume that the problem is located, with high probability, close to the devices (otherwise probably not all the devices would experience problems) and thus it begins the diagnosis phase by checking the gateway and the local network. If the problem is neither in the gateway nor in the local network, it check the middle boxes. If the problem is not 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).

2.6 Mobile network performance issue cause analysis

2.6.1 Use case introduction

This scenario addresses the issue of identifying the root cause of problems related to connectivity and poor quality of experience on mobile devices. More specifically, we explore how measurements that are collected at various points of the infrastructure can be used to automatically identify the reason for poor mobile network performance (e.g., video buffering, disconnections, poor browsing experience).

Identifying the root cause of poor connectivity on a device is not a trivial task. For instance, when a mobile device is trying to load a webpage or a video then any of the involved parties along the path of the information could be the bottleneck:

- **Device issues:** The user's phone might not be able to correctly load and display the content for various reasons: inadequate CPU or memory, missing codecs, lack of caching mechanisms, poorly configured drivers, lack of hardware acceleration or even the wrong video quality was selected by the application.
- **Mobile ISP issues:** The user might be in an area with poor cellular reception (low SNR, with only limited physical modulations usable) or where no high-speed data connectivity is available (lack of HSPA support, limited physical bitrate available on the radio channel). In some cases (e.g., football games, concerts) there might be a very high demand at a given geographic area that may affect a number of local users. Furthermore, incorrect settings at the RNC can result in long delays to connect to the network or alter the latency significantly. Finally, the mobile operator might be running low in backbone capacity (e.g., when microwave links or old wired technologies are used) resulting in lack of capacity to carry data until the base station.
- **Fixed ISP:** If a device is connected via a fixed network (e.g., home via WiFi or ethernet) then traffic from other devices in the same local network (e.g., connected to the same router) might be creating the problems. Furthermore, the ADSL router or the DSLAM might be misconfigured or congested. Finally, the backbone of the ISP or the peering points with the service providers or the core network might be congested.
- **Core network issues:** The issue might be generated by a congested or badly configured core network (routing issues), lack of peering points between the provider and the video hosting service.
- **Service and CDN provider: :** This includes issues with the content distribution network or the servers that support the service (e.g., congested video servers).

Due to the fact that these measurement probes rely on different legal and physical entities (e.g., user devices, ISPs, service providers) cannot freely exchange the collected information. Apart from defining the measurement points and the algorithms that are required to identify mobile connectivity issues, a key challenge is the fact that it addresses issues related to data ownership, sharing and federation of the monitored information.

2.6.2 Description of the associated analysis algorithms

As described, the mPlane platform involves a number of measurement agents that reside within different entities (e.g., user devices, the ISP where the device is connected, service providers) in order to discover quality of experience problems and identify the root cause along the path.

2.6.2.1 Overview of Measurements at each entity

User's Devices:

Instrumented applications provide measurements from the device and application point of view. For instance, a video application monitors "buffering" events that affect the perceived quality of service. A mobile device OS (e.g., laptop/phone) monitors the signal strength with the associated cell or WiFi station. Furthermore, the users can manually report problems that include subjective opinion about the network connections. This enables the platform to connect ground truth about the actual quality of experience. Finally, the device probe contains a troubleshooting engine that is responsible to combine the measured or reported information to detect and handle any network-related quality of experience issues. Furthermore, the troubleshooting engine is responsible to collaborate with the probes within other legal entities (e.g., ISP, service provider).

More specifically, this probe:

- Allows the users to report "quality of service" issues related to their connectivity (by pressing, for instance, a "my internet is not working properly" button). The information is used to identify local issues within the users' device and to further forward this information to the involved ISPs and service providers. This also enables the providers to collect subjective ground-truth information about each application (e.g., what are accepted latencies for web content, what are acceptable speeds/qualities for video browsing. Finally, this information is then associated with the data that are gathered by the sensors and the network (e.g., are the users moving, are they at home connected to WiFi, are they in a specific geographic region?).

- Contains various measurement probes that passively and actively collect

  Measurements related to the application's performance. For example, if a video player is loaded information such as number of events related to "buffering", average bitrate, dropped frames etc. For web browsing, statistics about transferring, loading and rendering each component are collected. Finally, for a gaming application information such as latency and jitter are collected.

  Measurements related to wireless connectivity: type of connectivity (e.g., edge, 3G, LTE, 802.11gm, Ethernet, ZigBee, Bluetooth), signal strength, disconnection history, association history, etc.

  Network measurements: MAC layer and TCP/UDP statistics, bandwidth usage, etc.

  Device related statistics: CPU usage, memory usage, running apps, etc.

- Finally, a troubleshooting engine combines the measured data with the manual reports to create a mapping between the conditions that can lead to perceived quality degradation. Semi-supervised learning algorithms are used to automatically identify application issues with limited user interaction. More specifically, prior user-initiated reports are associated to specific network conditions. When similar conditions are met (by the same or any other user), an automatic report is generated without the user interaction. Notice that, in some cases the user might be probed for further feedback. Finally, a combination of static thresholds is used to also trigger an automated report (e.g., if number of buffering events exceeds a threshold).

Mobile ISP :

If the device is connected to a cellular network, the provider uses an probe to troubleshoot issues from their point of view of the network. For example, to identify congestion in certain parts of the network (e.g., a tower) or to pinpoint underperforming middle-boxes within the network. As with the mobile device, a troubleshooting engine is responsible to collect and handle the measured information and the troubleshooting requests. Similarly with the device probe, the mobile ISP contains a module that:

- Passively collect information related to each base station: number of users, utilization, QoS policies.
- Passively collect information related to the backbone connectivity: utilization, TCP/UDP performance (RTT, loss rates, etc.).
- Passively collect information related to the performance any middle-boxes: web proxies, deep packet inspectors, routers, etc.
- Maintain the topology of the object exchanged so the path that was taken through the network can be traced if required by an inquiry.
- As with the previous probes, the troubleshooting engine is also used to automatically identify errors or investigate issues reported by the users' mobile devices.

Wired ISP :

Similarly to a mobile ISP probe, the Wired ISP probes are responsible to identify issues within the ISPs network. For residential connections the home router is instrumented with a measurement probe to identify issues related to the quality of the local wireless and wired connectivity. Furthermore, probes deeper in the ISP (e.g., DSLAM, routers, peering points) also provide measurements. As with the mobile device, a troubleshooting engine is responsible to collect and handle the measured information and the troubleshooting requests.

- Instrumented gateway (router): Passively collect information related to:

  The connectivity of each associated wireless and wired client of the home/business/public network (e.g., signal strength, utilization, bitrate, packet loss).

  Information about the network status and load of the wireless and the wired interfaces (e.g., bandwidth utilization, loss rates, retransmissions, round trip times).

  Information related to the configuration of the router (QoS policies, allowed ports, access lists etc.).

- Similarly to the mobile ISP, the wired ISP installs probes across its own backbone to collect information related to the performance any middle-boxes, routers, peering etc.

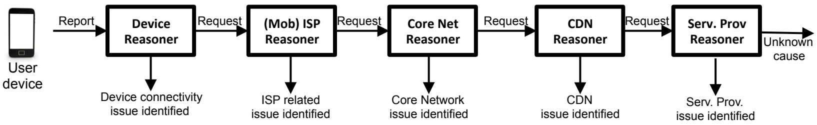


Figure 11: The iterative process of consulting with with reasoners across different entities.

- Finally, as with the previous probes, the troubleshooting engine is used to automatically identify errors or investigate device reported issues related to local Wi-Fi connectivity or within the ISPs network.

Core Network and Service Provider:

Core Network probes are used to identify issues with peering and backbone congestion or misconfiguration as with the mobile wired ISP bones middle boxes are used to take measurements and a troubleshooting engine is used to automatically identify errors and to investigate any device/ISP initiated requests.

2.6.2.2 Distributed troubleshooting and Data sharing

Due to the fact that data is generated within the network of various legal entities (e.g., the user, the ISP providers, core network providers) our troubleshooting detection algorithms follows a distributed approach in terms of data collection 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 user’s device, the mobile ISP, etc.).
- Each entity runs its own instance of a troubleshooting agent that can only access the internal data to identify any possible causes within the organization.
- 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.

For instance, Figure 11 shows an example of a user reported connectivity problem. In this example:

- The user reports an issue with connectivity or quality of experience (notice that the request to troubleshoot an issue can also be automatically generated).
- The application and device probes use the collected data to identify if the reason is within the device (e.g., poor signal strength, missing codecs, not enough memory, other applications are using the bandwidth). Only information that is collected and owned by the user’s device is used. If the reason is identified the issue is considered solved and the user and/or the

service provider are notified. If the reason is not identified then the local troubleshooting probe generates a request for further investigation is forwarded to the instance running at the ISP (mobile or fixed) that provides the connectivity. Only the required information such as the timestamp, the objects that caused the issue is shared to help the ISP identify the flow within its own network.

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

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

2.6.2.3 Troubleshooting Algorithms

At each site, the reasoner engine in our scenario offers three different capabilities i) *rule-based troubleshooting*, ii) *machine learning detection* and iii) *spatio-temporal correlation of multiple reports*. We now describe each one of these capabilities.

Rule-based troubleshooting:

When a request is made either by an external entity or internally (e.g., by the user or by a measurement) then the passively collected information in the database is used to pinpoint the exact cause. Standard rule-based mechanisms are used in this case. For instance the engine evaluates whether there was enough available bandwidth at the links that the object went through (e.g., only GPRS connection when trying to watch a video). An example is given in Figure 5.

The advantage of using rules lies into the fact that these are lightweight computations that can be used in mobile (battery operated) devices. The disadvantage of using this approach is how to set the threshold values. In some cases it is obvious (e.g, we can detect a problem when the CPU or the connectivity type is not enough to play a video) but in other cases it is harder (e.g., what is the number of retransmissions that generates poor video experience).

To make these correlations our system processes the data to identify threshold values with high confidence. If a significant correlation is found a rule-based model is used. Otherwise a more complex (and demanding) machine learning model is constructed.

Machine learning troubleshooting:

The main goals of the applied machine learning are ease/eliminate the human effort in manually building the complex rule based system. Usually such systems perform as good or even better than

Issue	Entity detected	Possible measurements
Configuration issues	User's device	Application API, OS API (e.g., missing codecs)
Inadequate CPU	User's device	Device CPU usage, video frames dropped
Inadequate memory	User's device	Low device memory, low refresh rates, high object allocation latency, swapping
Poor signal strength	User's device	Low RSSI, dropped packets, high mac-layer retransmissions
Frequent disconnections/mobility	User's device	Network state (connected non-connected), AP changes, rapid modulation and state changes
Poor wireless network speeds	User's device	Type of connectivity (GPRS, Edge, 3g, LTE)
High network usage	User's device	High traffic from other apps or the OS.
Network inaccessible	User's device	Network not configured (e.g., no IP) or no packets exchanged with gateway
Wi-Fi Congestion	Router	Signal strength, high contention windows, percentage of time medium was busy, high transfer rates
ADSL/cable/FTTH link issues	Router/DSLAM/ISP	High utilization, high transfer rates, high ping times, packet loss, TCP/IP metrics
Gateway Router issues	Router	High router CPU/Memory usage, large latency to forward a packet, queues, misconfiguration
Cellular base station congestion	Mobile ISP	High utilization, number of associated clients, lost frames, frequencies in use
Cellular base station issues	Mobile ISP	High latency, lost packets, RNC misconfiguration
Mobile ISP backbone	Mobile ISP	High utilization, high transfer rates, high ping times, packet loss, TCP/IP metrics, router statistics
Mobile ISP middle boxes	Mobile ISP	CPU/Memory, latency to process a request, logs
Mobile ISP peering	Mobile ISP	Peering utilization, latency, lost packets
Fixed ISP backbone	Fixed ISP	High utilization, high transfer rates, high ping times, packet loss, TCP/IP metrics, router statistics
Core Network Issues	Core Network	High utilization, high transfer rates, high ping times, packet loss, TCP/IP metrics, router statistics
Content Delivery Network	CDN	Selected server, load, CPU load, memory load, network utilization, latency to serve an object/query, disk/memory access time and utilization, cache hit ratios, number of users served
Service provider	Service provider	Selected server, load, CPU load, memory load, network utilization, latency to serve an object/query, disk/memory access time and utilization, number of users served

Table 5: Examples of detected errors using rules in a mobile scenario.

the human experts when complex environments are considered. Note, that these also could be used only as a automatic rule mining systems and the final decision could be made by a human expert.

The statistical machine learning can be used to detect the issues without explicitly specifying the rules for firing the alarm. These techniques could be divided into supervised (semi-supervised) learning or unsupervised learning. The basic idea of supervised learning lies in correlating the user reported issues with the measured variables. When a user reports a problem, previously observed correlation patterns can be investigated in an automatic way and most likely causes of problem reported to the user. Unsupervised learning, on the other hand, models a "normal" values of the observed variables without any user feedback and rises alarm if something unexpected is observed.

Different learning techniques can be used on different entities. The requirements for algorithms should be based on the entity type and data type that the entity collects. For example, on mobile devices algorithms are limited by slow CPU and limited storage resources. Therefore, simple stream processing algorithms should be considered. These do not store data on the disk and can process it on the arrival time. One example of such algorithm can be recursive linear regression with a sampling techniques to increase the performance. A streaming linear learning algorithm is use to associate possible issues to measurements. For instance as people click the "my internet is not working button" then the learning algorithm associates possible local conditions that typically result in poor experience. One can perform factor analysis of the built model to identify correlations between problems and the causing factors. This information is used to automatically generate future reports without requesting explicit user interaction. This is a key part of our system as it allows the ISP to collect ground-truth and crowd source quality of experience issues.

Inside the entities with higher computational power, energy and storage resources, more powerful and accurate models can be considered. This is also due to the fact that less and less explicit user feedback will reach entities further from the user. Such models include one class SVMs, time series causality detection, and outlier detection algorithms for time series.

Automatically identify issues as they happen: Similarly, machine learning is used to identify issues as they build up and warn the interested parties before they affect their network. Here, unsupervised learning algorithms can detect abnormal measurements and automatically ask neighbor entities to explain such statistical discrepancies.

Distributed troubleshooting: The topology information is used in every stage to identify the paths that each object took within the network. Therefore, if an error is not found within an entity then the troubleshooting engine is responsible to forward a troubleshooting request to the appropriate "next hop" that handled the data and handle any replies. Furthermore, as replies back the troubleshooting engine associates measurements to external problems.

For this purposes, we are implementing a system developed on top of storm to run streaming algorithms machine learning algorithms on top of the stream of incoming data.

Spatio-Temporal troubleshooting:

In the case of mobile devices, certain problems can occur in specific geographic regions. For examples, even within the same cell sector there might be areas with poor coverage. Furthermore, certain sectors, cell sites, RNCs, NodeDBs might be congested or misconfigured.

Our approach that allows external entities (e.g., user's smartphones) to report errors to the next entity along the data path (e.g., the mobile ISP) allows us to correlate reports from multiple devices in order to construct a spatio-temporal mapping of the reported issues for each data object or for group of objects (e.g., videos to the same CDN cluster).

To do this, the data produced by our monbilt probes are in geoJSON format where the most accurate information available is used (GPS, WiFi or GSM based localization). Furthermore, information about the mobile network deployment and the network topology is used to identify potential bottlenecks.

2.7 Anomaly detection and root cause analysis in large-scale networks

2.7.1 Use case introduction

CDNs are a vital part of the current Internet infrastructure. By deploying servers in multiple data centres across the Internet, content can be served to end-users with high availability and performance. However, CDNs pose challenges for ISPs, since changes in server allocation policies can cause sudden changes to the traffic carried by ISPs, impacting traffic engineering and possibly im-

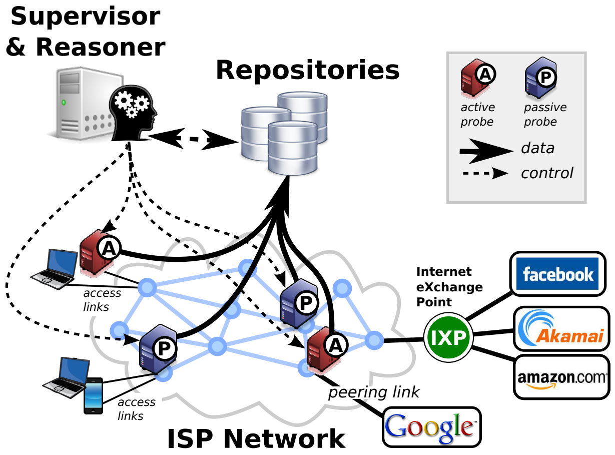


Figure 12: Scheme of a possible CDNs traffic monitoring system.

pairing end-user quality of experience. As such, ISPs need advanced tools to track the traffic served by CDNs.

In the context of mPlane, Fig. 12 shows a possible placement and combination of the different mPlane layers to monitor and detect anomalous behaviors in CDNs traffic. The ISP network can be seen as a composition of nodes interconnecting user clients to CDN servers. The role of mPlane in this specific scenario is to track the performance from users' access links up to CDNs peering link. The role of each of the mPlane layers to tackle this problem is the following:

The probes perform passive and/or active measurements, and represent the source of measurement data, which can be continuously produced (e.g., a passive probe monitoring links and exporting new data in a stream-like fashion), or can be generated on demand (e.g., by running commands such as `ping` to occasionally investigate a network path toward a server). Instrumenting this use case requires the installation a few high-end probes to monitor selected high-speed links possibly considering both the network edge (e.g., the traffic generated by an aggregate of end users) and the network core (e.g., the traffic traversing a peering link). Additionally, to cover ``the last mile'', a precious and critical resource for mobile carries, probes can be deployed to act as user devices, or end-users can instrument their devices to run some lightweight software (e.g., a browser plug-in) to automatically report measurements to the monitoring plane.

The repositories store, correlate, and pre-analyze the large amounts of data collected by the probes, both continuous data provided by the probes and static data such as routing tables topology descriptions, and other external data sources (please refer to the deliverable D3.2 for more details [59]).

The Supervisor handles the mechanics of the distributed measurement platform, as well as the final analysis, correlation, and compilation of the intermediate pre-processing results obtained from the repositories' analysis. The **Reasoner** is responsible for the automated iteration to filter or drill down to the root cause analysis of the detected anomalies.

2.7.2 Description of the associated analysis algorithms

2.7.2.1 Example of tracking Akamai CDN traffic policies

We describe the algorithms through a practical example on the detection of anomalies in CDNs' traffic: in particular, we address the problem of tracking Akamai CDN cache selection policies. Let us begin by detailing the components of the different mPlane layers associated to this analysis:

Probes: We refer to a data set collected from a passive probe installed at the network edge of a major European ISP to monitor an aggregation of about 50,000 ADSL customers. The probe is instrumented to run Tstat, an Open Source passive monitoring tool developed by Politecnico di Torino, and to collect per-connection logs, i.e., text files reporting a set of statistics on each monitored connection (see D2.1 and D5.1 deliverables for more details [59]). We use a one-week long data set collected starting from May 13th 2012, corresponding to 496 GB of data and containing 1.052 billion TCP connections.

Repository: The collected data are loaded into DBStream, a novel continuous analytics system developed by FTW. DBStream combines on-the-fly data processing of Data Stream Management Systems (DSMSs) with the storage and analytic capabilities of Data Base Management Systems (DBMSs) and typical ``big data'' analysis systems such as Hadoop. In contrast to DSMSs, data are stored persistently and are directly available for later visualization or further processing. As opposed to traditional data analytics systems, typically importing and transforming data in large batches (e.g., days or weeks), DBStream imports and processes data in small batches (in the order of minutes). Therefore, DBStream is like a DSMS in the way that data can be processed fast, but streams can be re-played from past data. The only limitation is the size of available storage. DBStream thus supports a native concept of time. At the same time DBStream provides a flexible interface for data loading and processing, based on the declarative SQL language used by all relational DBMSs.

Reasoner: in the considered scenario, the reasoner is not present and the iterative steps are run manually, using the basic analysis capabilities of DBStream to look for the root causes of the detected anomalies (in section 2.7.2.2 we give a description of the specific anomaly detection algorithm). To identify Akamai traffic, we rely on the MaxMind Organization Database which is capable to map an IP address to the name of its ``owner''. More specifically, we consider a connection to be related to the Akamai CDN if the destination IP address owner name contains the keyword `Akamai`. The MaxMind database has been loaded in DBStream to enhance the information provided in Tstat logs. In this context DBStream acts as a repository which performs join and filter operations among different data set.

Additionally, we track a single /25 subnet hosting Akamai caches. Servers in this network are reached by a direct peering agreement between the ISP and Akamai. Nodes in this subnet are very close, typically less than 5ms far away from customers in the monitored PoPs, and we refer to this subset of caches as ``Preferred'' in the following. The scope of the analysis is to investigate on the traffic of such preferred caches. In this context DBStream acts as a reasoner by performing some root cause analysis to investigate on the nature of the traffic captured.

Fig. 13 (left) details the evolution of the number of connections served by Akamai CDN nodes on Monday and Tuesday as seen from the vantage point. The preferred caches serve about 30% of

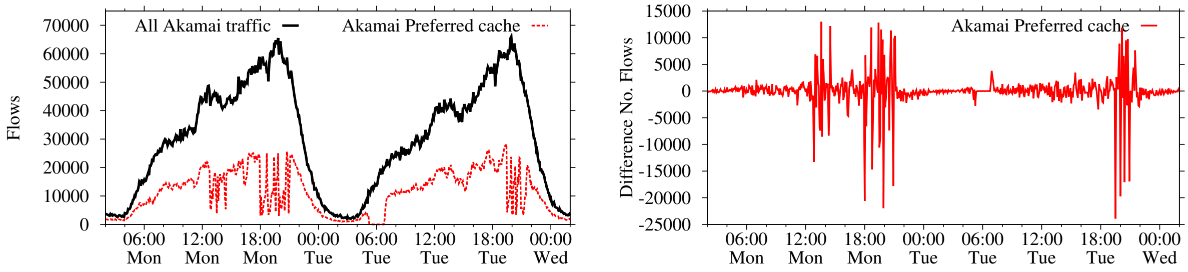


Figure 13: Evolution of number of connections served by Akamai CDN (left) and difference of number of connections served in consecutive 5 minutes time windows (right). UTC time is used.

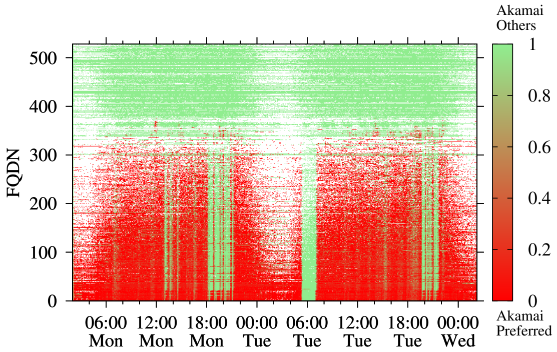


Figure 14: Evolution of the volume of requests per service name.

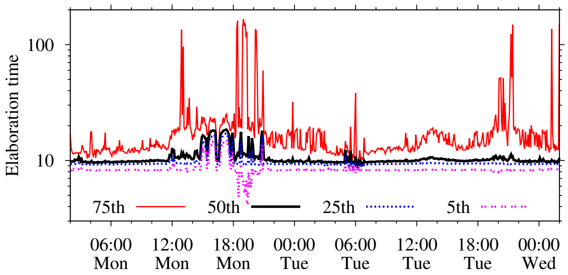


Figure 15: Evolution of server elaboration time percentiles for the preferred data center (41 seconds of query execution time).

traffic at peak time. Surprisingly, traffic served by the preferred caches presents occasional drops. These are effects of the CDN server selection policies shifting traffic back and forth among CDN nodes. To better capture the effect, Fig. 13 (right) reports the evolution of the difference of the number of connections served by the preferred cache in two consecutive time windows of 5 minutes.

We found this characteristic being present for the whole week but we did not observed any clear indication of periodicity. Moreover, results from the other vantage points are consistent, with the same effect being present at the same time periods. We use DBStream to further study the cause-effect relationship behind this phenomenon.

Single servers load: We start by checking if the sudden traffic shifts are due to some server failure in the preferred subnet. Three consideration hold for this analysis: (i) we found only 40 IP addresses being active and constantly used in the /25 subnet; (ii) there are a few (preferred) IP addresses handling up to 62% of requests but (iii) all servers presents almost absence of traffic in correspondence of the traffic shift. Thus, we can rule out the hypothesis of some specific node failing.

Per service analysis: CDN nodes host very different content, e.g., the same CDN server can serve both Facebook and iTunes/AppleStore objects. Tstat exposes this information by snooping the Full Qualified Domain Name (FQDN) of the requested content [13]. We thus check if the observed traffic shifts are due to the CDN moving some specific content, e.g., only *facebook.com*, or instead involves all contents, independently from their popularity.

DBStream is used to group rows by service names[10]. DBStream considers time bins of 5 minutes, and for each service name it computes the fraction of requests served by preferred and other caches. The obtained values are represented by the heatmap shown in Fig. 14. We selected the most popular services, and sorted them by the probability of being served by a preferred server. Results clearly show two groups: the bottom 300 services are normally served by some server among the preferred caches (red dots). The other 200 services are served exclusively by other Akamai CDN servers (green dots). Services not accessed anymore during off peak time are left white.

Also in this case, when some traffic shifts occurs, practically all services are migrated to other caches, as testified by the green vertical bars. Only a group of about 20 services is never migrated, except during a 2 hours from 5am to 7am on Tuesday, when traffic from the preferred data center goes practically to zero (see Fig. 13). Services on this group refer to Facebook static content being served by Akamai (e.g., *photos-N.ak.fbcdn.net*, *fbcdn-photos-N.akamaihd.net*). All other services are instead migrated to other CDN servers.

In general, results indicate that the traffic shifts are not related to some particular service, but are rather the effect of changes in the server allocation policies impacting all services.

Impact on performance: We conclude our analysis investigating the impact of the traffic shifts on the end-user offered performance. Fig. 15 (top) reports the evolution of the 5th, 25th, 50th, and 75th percentiles of the elaboration time[3] for the considered time period (y-axis is in log scale). In this case DBStream considers a time interval of 5 minutes to retrieve accurate percentile estimations. Results show that during the traffic shift occurring on Monday, some impairment of the elaboration time is visible. In particular, the 50-percentile grows from about 10 ms to about 20 ms before and during the shifts happening at 18:00. This would suggests that CDN server re-allocation could be triggered by some performance issues. However, this is not confirmed during Tuesday, when no practical impact on performance is visible in correspondence of traffic shift. Extending the analysis to the whole week (results not reported due to lack of space), we see that the same behavior observed on Monday is present also for the days before. Instead, since Tuesday the server response time does not present any oscillations anymore, while traffic shifts are still evident during the days.

In this example analysis we were not able to identify the causes behind ``the anomaly'' in the traffic of the preferred Akamai caches. Notice that this is not a negative result per-se. In fact, the example underline how the need of an automated reasoner capable of performing a root cause analysis. By extending the capabilities of the reasoner the analysis could be much more deep. For example, triggering active experiments (e.g., a traceroute) when the anomaly is in place would allow to further investigate the nature of the anomaly captured by the analysis delay observed via passive measurements.

Similarly, checking the presence of the same anomaly in the traffic observed by other vantage points could also improve the knowledge about the phenomenon. For example, if different probes do not show the same effect there might be problem along the path, i.e., there might be some problem in the ISP network.

[10] Some string pre-processing is applied to group together FQDN such as *a1.da1.facebook.akamai.net*, *a2.da1.fecebook.akamai.net*, etc.

2.7.2.2 Web services anomaly detection

With respect to the previous example, we tackle here the Anomaly Detection problem from a different angle, by focusing on the characterization of the traffic and the delivery behavior of a specific (yet extremely popular) web-service heavily relying on CDNs for being accessed. The purpose is to gain an understanding of the interplay among different hosting and delivering organizations, an by that to build an anomaly detection system capable of automatically detecting possible anomalies in the delivering mechanism. In fact, as we will show, the way Internet-scale web-services are delivered is very dynamic and complex to characterize, thus putting several challenges upon an automatic anomaly detection system.

As an example we perform an analysis of Facebook, the most popular and widely spread Online Social Network, as well as of the main CDNs and caches serving its content, i.e., Akamai, Facebook AS, and other important caches observed in our traces. Facebook represents an excellent case study for investigating the content provisioning complexity typical of Internet-scale web-services, as it is distributed by the most dynamic and widely deployed CDN so far. Indeed, due to the high number of daily users and the high volume of served traffic, Facebook has to deploy a sophisticated content delivery strategy. Indeed, in the traces collected at our vantage point (in the production network of a major European Mobile Operator) we observed more than 6000 server IPs serving Facebook contents, distributed across 20 Countries, and 250 different Autonomous Systems (AS), over 4 weeks. This confirms the wide-spread usage of several third-party content providers which adopt completely different working paradigms, as we show in the following.

2.7.2.2.1 Distribution of traffic across ASes To illustrate the role of the different Facebook content providers and their resource allocation policies, we plot in Fig. 16 the time series of the 5-minutes count of flows and exposed IP addresses (i.e., servers) per AS (for the top-5 ASes), for 4 consecutive days from July the 29th to Aug. the 1st, 2013. The figure shows that the flow share across the top-5 ASes[11] remains practically constant during the day. There is a clear daily pattern in the number of active IPs, and it is worth noting that Akamai systematically doubles the number of deployed servers during the peak hours (9pm - 10pm). Akamai and Facebook AS serve the largest share of Facebook flows. However, Akamai employs much more servers than Facebook AS, as we found it serves the largest flows corresponding to static contents, showing the role breakdown through the different ASes.

The figure also shows four ``anomalous'' events, identified as A, B, C and D, which break the normal traffic patterns and unveil the nature of the agreements between the ASes. Events A and B have similar characteristics. During the anomalies, even if the number of IPs steeply increases, the number of flows and volume of traffic served by Akamai abruptly decreases. NO and TeliaNet, which usually serve a negligible share of Facebook traffic, immediately deploy a number of backup machines (i.e., IP addresses) to take over Akamai traffic. To get a better view on this event, we plot in Fig. 17 a 12 hours zoom around the events C and D. During the event C, the Akamai glitch is again compensated by TeliaNet and by NO in terms of volume. However, unlike Telianet, there is no evident peak in the the number of flows served by NO, suggesting that the latter takes over the largest flows from Akamai. Event D differs from the previous ones since it does not involve Akamai and it is characterized by a swap between the number of flows served by NO and TeliaNet. The analysis of the four events suggests the existence of a chain of agreements between the ASes: Akamai gets support from NO, which in turn co-operates with TeliaNet. Therefore, whenever NO

[11] From hereafter, we refer to the Local Operator as LO, and the Neighbouring Operator as NO.

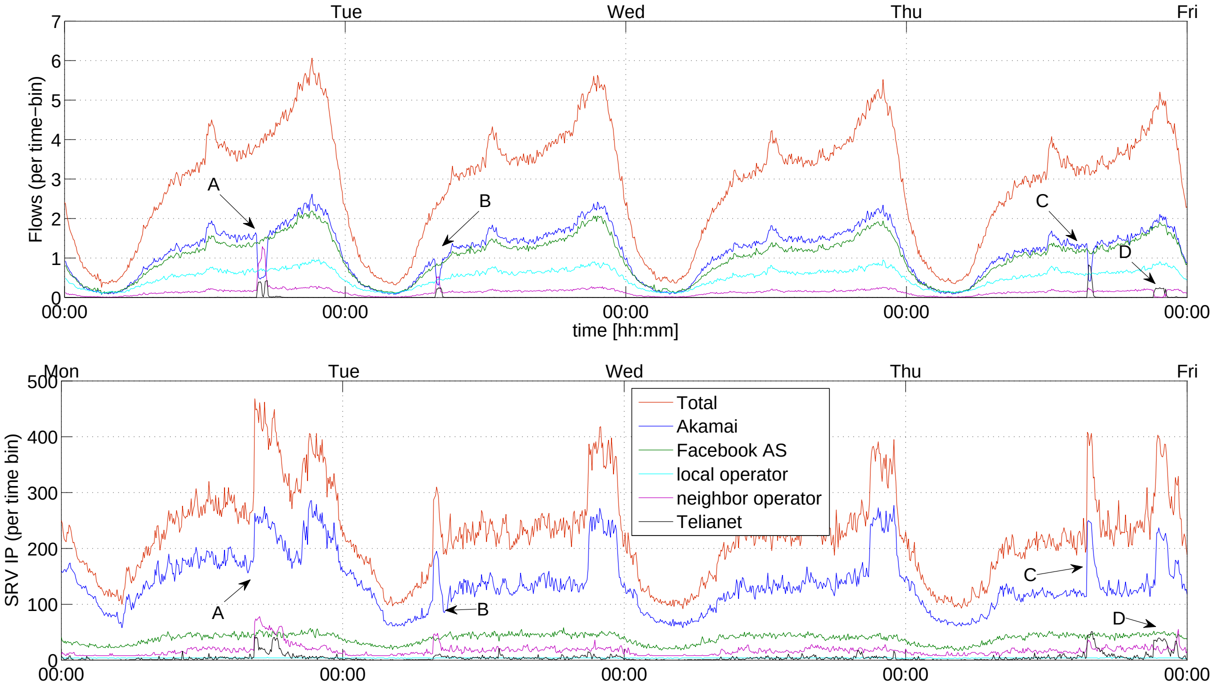


Figure 16: Count of flows (up) and server IPs (down) during 5 minutes, per AS. Actual flow numbers are rescaled for privacy reasons.

cannot cope with the traffic load handed over form Akamai, TeliaNet comes in support. Notably, we did not observe any anomaly in the total traffic, throughput, average RTT of the active IPs, nor in the number of erroneous HTTP responses, during the events A-D, suggesting that the load balancing policy worked effectively towards the objective of preserving the end-user perceived quality of experience. Still, these events might be critical for a network operator, as they entail significant traffic shifts and could have an impact on the Operator's network and traffic planning. In this specific case we verified via traceroutes that Akamai, TeliaNet, and NO are neighbours to LO. As reported in the Internet AS-level topology [2], the nature of the commercial agreement between LO and these three ASes is very different. Thus, we argue that such events might have economical implications for the LO, therefore they are worth to be reported.

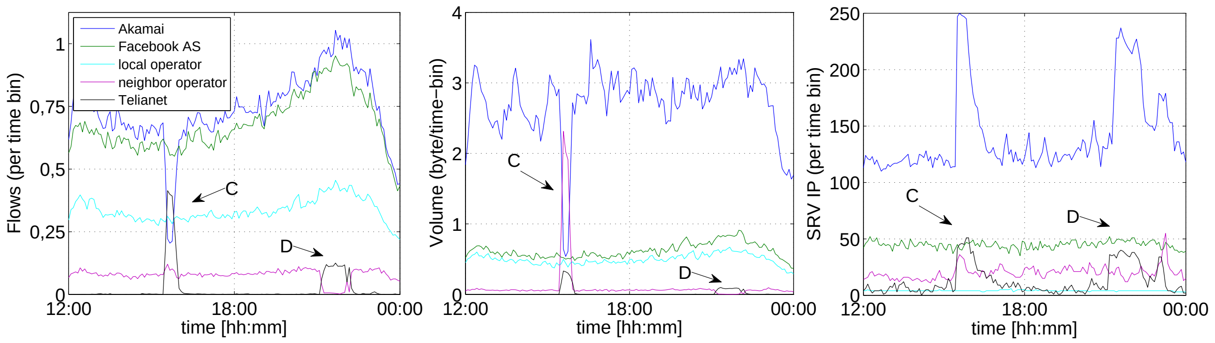


Figure 17: Total count of flows (rescaled) and server IPs over 5 min., per AS, for 12 hours.

2.7.2.2.2 Temporal Characteristics of Traffic Distributions To get further insights into the way different organizations serve Facebook traffic, we investigate how volume and flows distributes across the IP addresses exposed by the CDNs over time. In particular, for each individual server IP address we keep individual counters for the volume and the number of flows. Counters are cumulated at different time-scales from 1 to 60 minutes, to enable multi-scale analysis. At the end of a time interval we compute the distribution of the counters across the server IP addresses, and by that we obtain timeseries of distributions. By comparing the distributions referring to the different time intervals, we get an insight on how IP addresses of the different organizations complement each other and implement a load balancing policy. To quantify the degree of similarity between two distributions, we resort to a symmetrized and normalized version of the Kullback-Leibler divergence [29]. However, to fully understand the complex temporal structure of the CDN traffic, we need to visualize and quantify the degree of (dis)similarity of a large number of distributions over days and even weeks. For this purpose, we employ an ad-hoc graphical tool already proposed in [29], referred to as *Temporal Similarity Plot* (TSP). The TSP allows pointing out the presence of temporal patterns and (ir)regularities in distribution timeseries, by simple graphical inspection. The TSP represents, via a heat map, the reciprocal of the dissimilarity (i.e., the degree of similarity) between the traffic distributions in two time bins. Figure 18 gives an example of TSP for the distributions of all the Facebook flows across all the server IP addresses providing Facebook content, over $T = 28$ days. The blue colour represents low similarity (high divergence), while red corresponds to high similarity (low divergence). Because of the symmetry of the adopted discrepancy metric, the TSP is symmetric around the 45° diagonal.

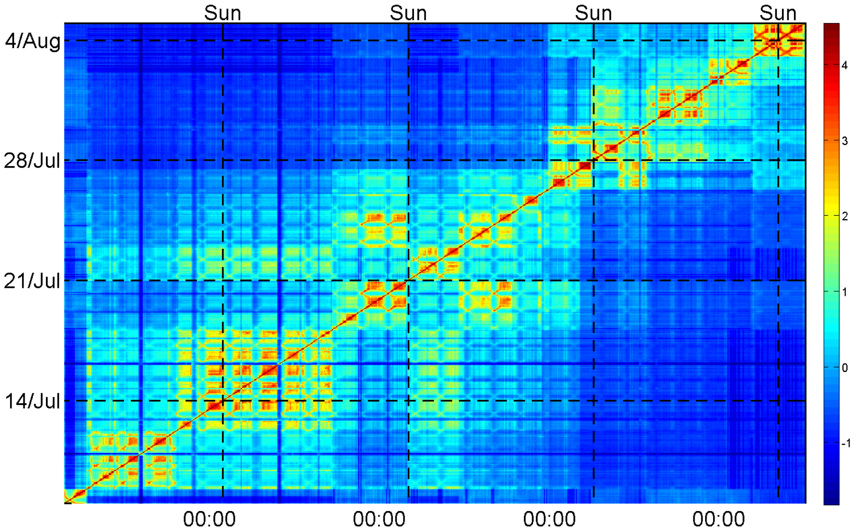


Figure 18: TSP of flow distributions at 1h time scale, over 28 days.

The TSP in Fig.18 refers to the distributions on a time-scale of 1 hour. Note the regular ``tile-wise'' texture within a period of 24 hours, due to the daily cycle. The lighter zones correspond to the time of the day, whereas the dark blue zones correspond to the night-time periods when the traffic load is low. The low similarity (blue areas) at night (2am-5am) is caused by the low number flows which induces larger statistical fluctuations in the compared empirical distributions. This pattern repeats almost identical for few days, forming multiple days macro-blocks around the main diagonal of size ranging from 2 up to 6 days. Besides the basic tile-texture, the analysis of the entire observation period reveals the presence of a more complex temporal strategy in the (re)usage of the IP address space. Indeed, it discloses a reusage of (almost) the same address range between 12-17th and 21-

22th, and between 18-20th and 23-24th of July. Finally, we observe a sharp discontinuity on July the 26-27th.

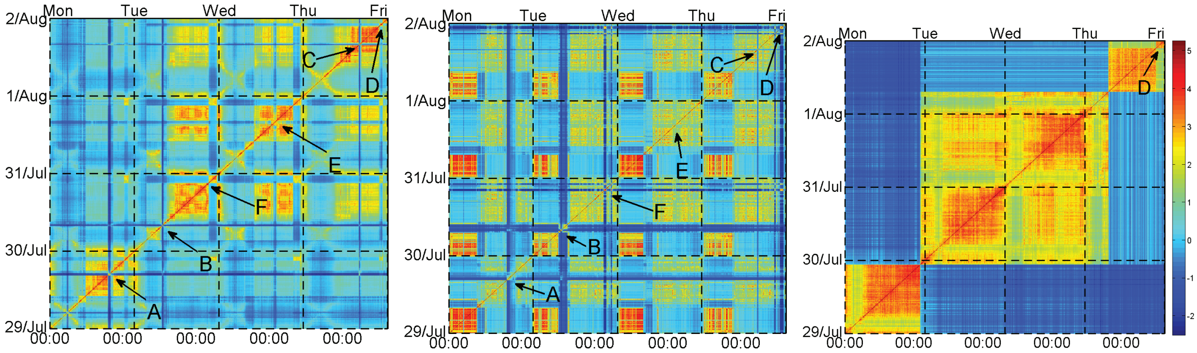
To get a better understanding of such a complex behaviour, it is worth analysing separately the two main sources of Facebook traffic volume-wise, that is Akamai and the Facebook AS. The visual comparison of Fig. 2.19(a) and Fig. 2.19(b) against Fig. 18 explains the complexity of the latter as the superposition of the very different allocation policies used by Akamai and Facebook AS. In particular, Akamai uses the same addresses/servers, for 4 to 7 days (see multi-days blocks around the main diagonal). When it changes the addresses used for serving Facebook traffic, the shift is not complete as we can observe the macro-blocks slowly fading out over time. This suggests a rotation policy of the address space of Akamai, on a time-scale longer than a month. However, we cannot prove this conjecture because of the limited duration of the analysed dataset. On the other hand, Facebook AS does not reveal such a clear temporal allocation policy. It alternates periods of high stability (e.g. between the 12-17th of July) with periods during which the allocation policy is extremely dynamic (e.g., form July the 26th onward). It is interesting noticing that Facebook AS is the responsible for the address space re-usage, between 12-17th and 21-22th, and between the 18-20th and the 23-24th of July, and for the abrupt change on July the 26-27th, both already identified in the total traffic (see Fig.18). Finally, NO always uses two distinct address sets during the night-time and the day-time (see Fig. 2.20(b)). Similar results have been obtained analysing the TSPs of the volume distributions across the IP addresses, for the top-5 ASes.

(a) Akamai

(b) Facebook AS

Figure 19: TSP of flow distributions at 1h time-scale.

The TSP also allows identifying anomalies in the traffic distributions. Indeed, a transient anomalous event appears in TSP as a blue cross centered on the main diagonal, and positioned around the time of the event. Figs. 2.20(b), 2.20(a), and 2.20(c) show the TSPs of the flow distributions between July the 29th and August the 1st at 5 minutes time-scale, for Akamai, NO, and Facebook AS respectively. As expected, the events A, B, and C are clearly visible in the TSPs of Akamai and NO, and are totally absent from the Facebook AS TSP. These events are also clearly visible in the TSP of Telianet (not reported for space limitations), and are in total accordance with the analysis of the total flows time-series in Fig. 16 and Fig. 17. Regarding the event D, it is observable in all the TSPs of Fig. 20, including the one of Facebook AS, even though it is completely invisible in the time-series of the total flows and volume of Facebook AS (see Fig. 17). Furthermore, Figs. 2.20(b) and 2.20(a) pinpoint the presence of two more anomalous events in the Akamai and NO traffic, namely the event E and F, that are completely invisible in the total flow and volume plots. In the following section, we provide guidelines and discussion on how to conceive a system capable of automatically detecting the kind of observed anomalous events.



(a) Akamai

(b) Neighbor Operator

(c) Facebook AS

Figure 20: TSP of flow distributions on 5min. time-scale.

2.7.2.2.3 Design of an anomaly detector for web-services Network operators are particularly concerned with revealing macro-anomalies, i.e., those events that affect many of their customers accessing a web-service. This motivates us to consider a distribution-based approach where the network traffic can be represented by traffic distributions across the IP addresses of an AS. In this way, it is possible to profile the aggregated behaviour of the traffic served by a whole organization.

Therefore, we propose the adoption of a variation the AD scheme originally proposed in [29]. The algorithm relies on the analysis of distributions implements a "change-point detector"□ for distribution timeseries, that can reveal deviations in the temporal trajectory of the entire distribution from the "normal" behaviour observed in the past. The considered method is designed to cope with the temporal variability of both the sample size and distribution, leveraging the daily and weekly pseudo-seasonality of the real traffic process to dynamically identify the set of past observations (i.e., distributions) "most similar" to the current one. It does so in a way that is robust to the intrinsic irregularities in the pseudo-cycles - due for example to weekend-like patterns occurring in national festivities, or solar/legal time shifts - as it does not rely on any external information like calendar day nor absolute time. This set is then taken as the reference baseline to evaluate the consistency of the current distribution with respect to "the past"□: in other words, pseudo-seasonality is exploited to compensate for non-stationarity. That is, the traffic at the same hour of different days tend to be pretty similar, therefore they can be used to evaluate future samples at the same hour of future days (see for example Fig. 2.20(b)). Remarkably, the method is designed to exploit pseudo-seasonality if present, but does not requires it as it does not assume any particular temporal structure of the traffic distribution trajectory: in fact traffic features might exhibit different temporal structures. Being not designed upon any specific traffic characteristic, the method can be applied to traffic dimension with very different structural characteristics and at different timescales, regardless to the shape of the distribution and the adopted binning. For details about the reference identification algorithm we refer the reader to [28].

The former aspect assumes a paramount relevance in the context of CDNs' traffic AD, as CDNs host and serve the web-service contents in a highly dynamic way. Indeed, most of the AD schemes considers training once-and-for-ever and tests the current sample against the most recent ones. However, in our context a reference based only on the most recent samples (i.e., distributions) would not be able to follow, for example, the steep variation in the total volume as well as in traffic distribution in the morning and in the late evening, causing a series of false alarms, therefore leading to inaccurate conclusions. On the other hand, from initial trials with the proposed algorithm we found that it successfully adapts to the time of day variations, while remaining still able to detect the events A-F.

However, for some organizations such as Akamai and Facebook AS (see Figs. 2.19(a) and 2.19(b)), the pseudo-cyclical behaviour is broken by periodic traffic shifts to different address pools, without any apparent alignment with the working-days/weekend-days cycle. This poses a serious challenge to the reference identification algorithm as it implies a "working point"⍰ shift and calls for restarting the algorithm training. We are currently working to overcome this limitation investigating two possible directions. One possibility is to extend the training to a time window long enough to accommodate all the possible ``legitimate'' working states. This solution, however, has evident scalability problems. The alternative is to force the algorithm retraining as soon as the alarm burst length exceeds a given threshold. In this case the assumption is that the alarm burst is due to configuration shift rather than to a long lasting transient anomaly.

2.7.2.2.4 Required functionality of the AD approach We depict in figure 21 the breakdown of the required functionalities at each of the mPlane layer (i.e., WPs). We assume that the probes capture, parse the HTTP traffic, and filter the relevant web-service. Then, at the repository (WP4) DB-Stream performs the count of the flows/packets per server IP address for each aggregation timescale. Starting from the individual counters DB-Stream periodically calculates the relevant feature distributions. Finally, the actual AD algorithm, consisting of the reference identification algorithm and the actual AD test, runs at WP4. The output of the AD tool is treated as a warning form a ``reasoning'' function which takes both the detection results (from multiple traffic features), and context information, to decide about the nature and the root cause of each event. Indeed, the decision about the nature of a warning may completely change depending on the purpose of the detection. If the objective is optimizing end-user perceived quality, even dramatic changes such as the events A-D discussed in the former Sections are not relevant, as they do not result in any perceivable degradation of the average RTT and throughput (at least out of our analysis). On the other hand, if the purpose is cost optimization for an access operator, the same events might be extremely relevant. In fact, delivering traffic to the end-users from different CDNs may result in different cost figures, depending on the nature of the commercial agreements with the transit operators (e.g., peering vs. customer-provider agreement). Therefore, delegating to the reasoner the ultimate decision about the nature of an event allows decoupling the detection rule design from the final usage of the AD system. Still the output of the reasoner should be feedback to the detector as it has an effect on the reference update policy. Finally, note that the reasoning function is not only on charge of correlating available results but is expected to be able to trigger additional (possibly active) measurements (e.g., traceroutes) for the purpose of further investigating the root cause of an event.

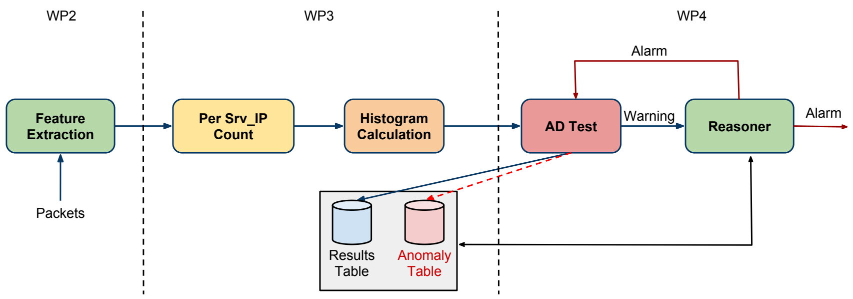


Figure 21: Scheme of the AD tool.

2.8 Verification and certification of service level agreements

2.8.1 Use case introduction

One of the case study is the verification of Service Level Agreement and one of the investigation carried out in these first month of the project has been an experimental analysis on correlation among quality of experience, quality of service and channel capacity, pointing out the enormous differences that can be present in ultra broadband accesses as in the case of FTTx accesses. The results presented in D5.1 about the measurements on GPON, show that concept of Service Level Agreement has to be defined for each of the OSI layers to take into account both the requirements of users and ISP. From such results we have defined a novel procedure to simultaneously measure channel capacity, throughput and goodput, and in this deliverable we show the algorithm that we propose for a suitable active probe.

2.8.2 Description of the associated analysis algorithms

The algorithm that FUB is developing executes as shown in the flowcharts of figure 22 and 23. The detailed the steps of the algorithm are the following:

1. Start the measure, the algorithm checks continuously the repository (every time it receives new data for a certain ID), it checks the data integrity, if the data is complete ect.
2. Estimates the Capacity with the TCP data that are from the server part of the probe (agent & server), the data are different from the ones that are used by the agent (probe), because the algorithm uses the medium value of all the mediums values in a period of time (that are sent to the repository by the agent and the server part of the probe)
3. It confront the estimated value with the estimated value that was reported by the probe, a difference with delta is allowed. If the values correspond go to point 7.
4. If the values do not correspond, then the algorithm increments a variable NrError (number of error for the specific ID) with 1, and tests this variable if it is under a certain threshold or not. If it is under the threshold go to point 6.
5. If not, the algorithm notifies the probe to stop the test and sends to the client an error message, after that it notifies an administrator for the error.

6. If NrError is less than the threshold, the algorithm will put a flag to the corresponding measure and will end the process of verification for that measure. (all measures with the flag will not be used for the calculation and the release of the certificate)

7. If the estimated capacities will corresponds, the algorithms continues, it will estimate the capacity of UDP in base of capacity reported by the probe (the server part), the error reported of UDP +(-) a small variable delta1

8. Then it checks if all the estimated capacities and measured capacities correspond or not (with small differences allowed delta and delta1). If they correspond go to point 12.

9. If they do not correspond go to point 4.

10. Verify if there is any error with the measures, strange results etc. If no do nothing let the procedure continue.

11. If yes, discard the test, delete data from repository, is useless data, prepare and send a report to the user, with the problem encountered and if the problem is made by the user, a possible fixme.

12. If they (estimated capacity TCP & UDP, measured capacity UDP, small difference is allowed, delta & delta1) correspond calculate the mean, a final one, of the capacity based on all estimation and all measurements so far.

13. Check for possible channel errors (noise) ect. Verify from the ID, the capacity of the channel of the zone.

14. Final confirmation of all estimation and measurement, with the final mean, with both data of the probe (Agent and Server). All estimation and calculation should be nearly the same (+[-] delta &/ delta1)

15. Prepare certificate, report all useful information.

16. Send Certificate/report to user.

17. END.

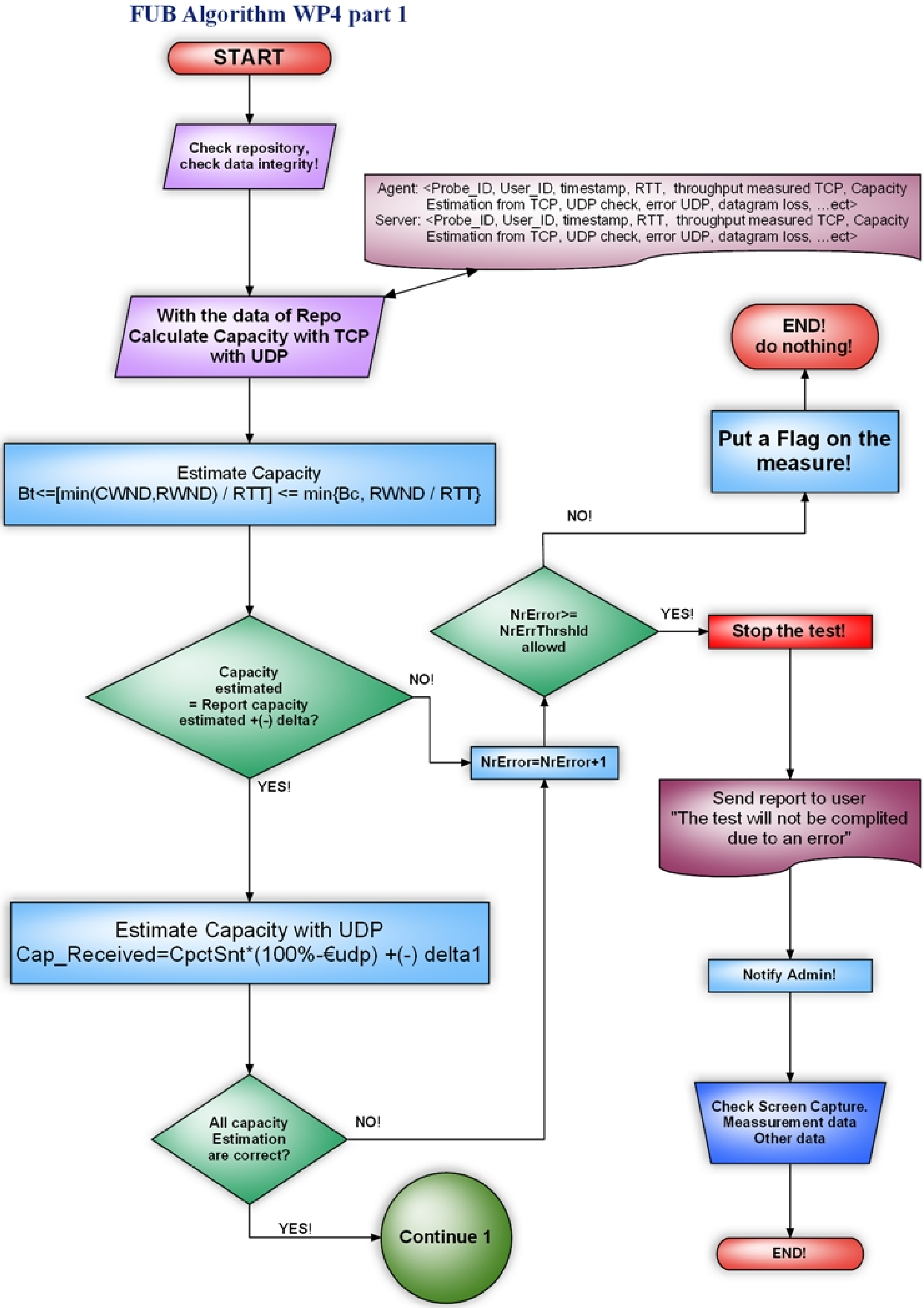

FUB Algorithm WP4 part 1
START
Check repository, check data integrity!
Agent: <Probe_ID, User_ID, timestamp, RTT, throughput measured TCP, Capacity Estimation from TCP, UDP check, error UDP, datagram loss, ...ect>
Server: <Probe_ID, User_ID, timestamp, RTT, throughput measured TCP, Capacity Estimation from TCP, UDP check, error UDP, datagram loss, ...ect>
With the data of Repo Calculate Capacity with TCP with UDP
END! do nothing!
Put a Flag on the measure!
Estimate Capacity Bt<=[min(CWND,RWND) / RTT] <= min{Bc, RWND / RTT}
NO!
NrError>= NrErrThrshld allowd
YES!
Stop the test!
Capacity estimated = Report capacity estimated +(-) delta?
NO!
NrError=NrError+1
YES!
Send report to user "The test will not be complited due to an error"
Estimate Capacity with UDP Cap_Received=CpctSnt*(100%-€udp) +(-) delta1
Notify Admin!
All capacity Estimation are correct?
NO!
Check Screen Capture. Meassurement data Other data
YES!
Continue 1
END!

Figure 22: First part of the flowchart of the algorithm.

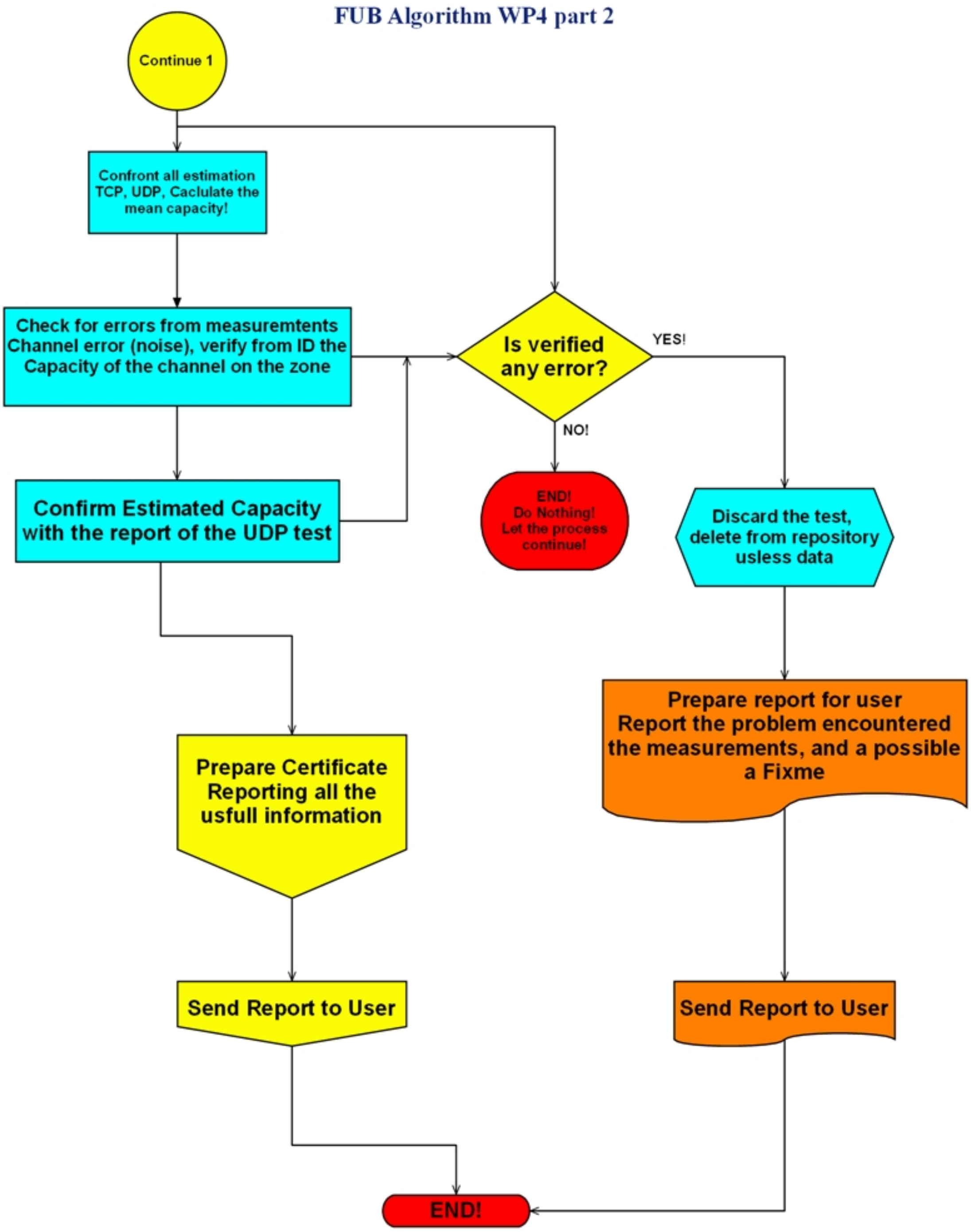

FUB Algorithm WP4 part 2
Continue 1
Confront all estimation TCP, UDP, Caclulate the mean capacity!
Check for errors from measuremtents Channel error (noise), verify from ID the Capacity of the channel on the zone
Confirm Estimated Capacity with the report of the UDP test
Is verified any error?
YES!
NO!
END! Do Nothing! Let the process continue!
Discard the test, delete from repository usless data
Prepare report for user Report the problem encountered the measurements, and a possible a Fixme
Prepare Certificate Reporting all the usfull information
Send Report to User
Send Report to User
END!

Figure 23: Second part of the flowchart of the algorithm.

3 Generic analysis algorithms

In this chapter we have collected a first small set of algorithms and approaches that are felt to have a more generic character and are therefore usable in several mPlane use cases and also possibly in other scenarios not currently considered in this project.

The five examples that will be further elaborated in the sequel address important topics. The first one is measurement prediction whereby some properties of some network paths are inferred (rather than measured directly) from the measurements of other paths. This technique has a clear scalability asset in large-scale systems where the number of paths can be fairly large. This inference problem is cast as a matrix completion problem and then solved in a distributed way, without having to rely on central nodes or central repositories. The generic character of the approach is due to its applicability to different sorts of metrics (symmetric or not, additive or not, represented by exact values or by classes).

The second family of generic algorithms deals with topology discovery and inference beyond traditional traceroute programs. The proposed new techniques already allow to detect middleboxes, TCP proxies and NATs, and other techniques are being explored to infer the weights of the links of the discovered topologies.

The third section proposes a set of algorithms to measure one-way delay variations (whose actual implementation in probes is the object of WP2, and is therefore reported in Deliverables therein), and the way to relate variations of one-way delay to performance of generic classes of applications. These information can be encoded and used by mPlane reasoners, e.g., to refine the drill down method, or to differentiate branching in specific use case (which is thus more fit to WP4 purposes, and is therefore reported here).

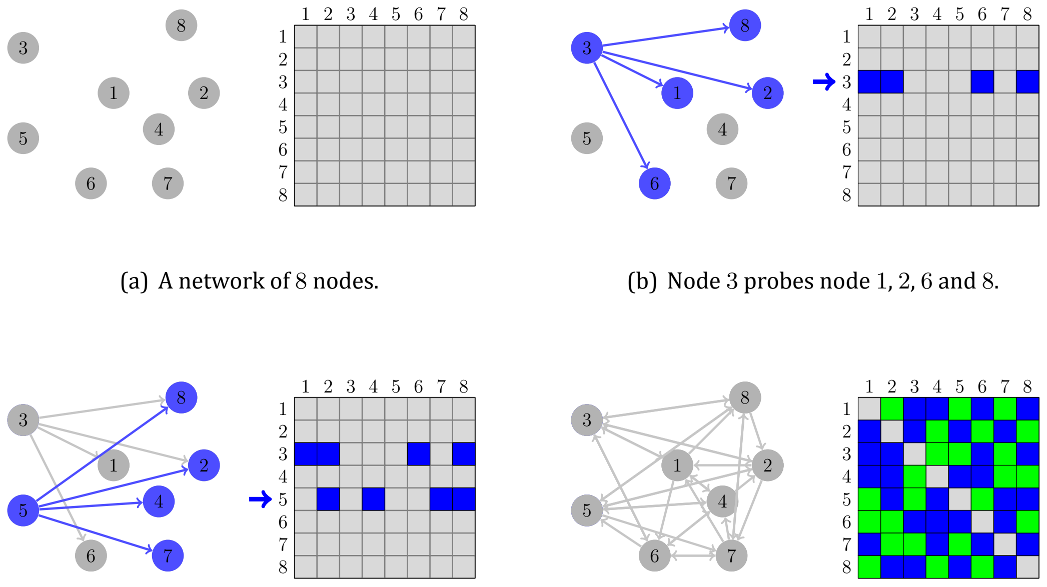
The fourth section addresses graph-based modeling, which provides a foundation for explaining phenomena and/or investigating problems involving one-to-one relationships/interactions (represented by edges) among entities (represented by vertices) allowing data analysis and mining to understand relations between these entities. The technique is applied to the analysis of the relationships between (attributes of) traffic directed to certain Autonomous Systems (AS)/address prefixes and content caches/servers. The technique is then extended to probabilistic hypergraphs and hierarchical directed acyclic graphs.

The fifth section focuses on the interest of Statistical Relational Learning (SRL) that combines probabilistic graphical models (probabilistic learning and inference) to model and reason about uncertainty with representation language to describe relational properties of the data and complex dependencies between them (logical learning and inference). The goal of SRL is of particular interest in the context of mPlane where the reasoner aims at learning hidden dependencies between multi-relational, heterogeneous and semi-structured but also noisy and uncertain data.

3.1 Prediction of unmeasured paths

3.1.1 Network Inference

The generic tool proposed in this section is useful to estimate the properties of all paths, such as round-trip time (RTT) and available bandwidth (ABW), in a large-scale system, without having to measure them all, which would require $O(n^2)$ measurement cost, where n is the number of nodes.



(a) A network of 8 nodes.

(b) Node 3 probes node 1, 2, 6 and 8.

(c) Node 5 probes node 2, 4, 7 and 8.

(d) A matrix with missing entries is formed.

Figure 24: A matrix completion view of network inference. In (d), the blue entries are measured path properties and the green ones are missing. Note that the diagonal entries are empty as they represent the performance of the path from each node to itself which carries no information.

To address this issue, a natural idea is network inference whereby only a subset of paths are actually measured while all others are predicted. Although less accurate compared to the measurment of all paths, this ``measure a few and predict many'' framework is much more scalable due to the significant reduction of measurement overheads.

We have carried out active research on network inference. In particular, the prediction of unmeasured paths is cast as a matrix completion problem where a partially observed matrix is to be completed [85, 50]. In this context, the matrix to be completed, X, is a performance matrix, with the ijth entry, x_{ij}, representing the performance of the path from node i to node j, measured by a chosen path property such as RTT and ABW. Each node probes a few other nodes, measures the performance of the paths between them. The measurements are put at the corresponding entries of X, and the missing entries are the performances of those unmeasured paths and need to be predicted. The process is illustrated in Figure 24 with an example of a network of 8 nodes. Comparing to previous approaches to network inference [27, 19, 24, 70], the matrix completion formulation relies on neither topology or routing information of the network nor geometric constraints. Instead, it exploits the spatial correlations across network measurements on different paths which have long been observed in various research.

A great benefit of the matrix completion formulation is that it can deal with qualitative performance measures such as binary performance classes and ordinal ratings. Conventionally, the performance of a network path is represented by the real value of some path property. While this quantitative representation has been commonly accepted by the networking community, it does not reflect the QoS experience perceived by end users which by definition is what network performance is concerned with. Thus, we have investigated more qualitative than quantitative performance representations based on *binary classes* (The performance is ``good'' or ``bad''.) and on *ordinal ratings* (The performance is quantized from 1 star to 5 stars) which have the following advantages.

1. Classes and ratings carry sufficient information that already fulfills the requirements of many applications.
2. Classes and ratings are rough measures that are cheaper to obtain than exact property values.
3. Classes and Ratings are quantized measures and can be encoded in a few bits, saving storage and transmission costs.
4. Classes and ratings are dimensionless or pure numbers with no unit. This feature unifies different path properties and eases their processing in applications.

Furthermore, we have developed a fully decentralized approach that solves the matrix completion problem with matrix factorization based on Stochastic Gradient Descent (SGD) [85, 50]. The so-called *DMFSGD* approach has some distinct features.

1. It requires neither explicit constructions of matrices nor special nodes such as landmarks and central servers where measurements are collected and processed. Instead, by letting network nodes exchange messages with each other, matrix factorization is collaboratively and iteratively achieved at all nodes, with each node equally retrieving a small number of measurements.
2. It is flexible to deal with various properties such as RTT and ABW and to deal with not only property values but also performance classes and ratings.
3. It is simple and computationally lightweight, containing only vector operations.

These features make DMFSGD suitable for dealing with practical problems, when deployed in real applications, such as measurement dynamics, where network measurements vary largely over time, and network churn, where nodes join and leave a network frequently.

3.1.2 Network inference algorithm: DMFSGD

3.1.2.1 Matrix Factorization (MF)

As mentioned earlier, the network inference problem illustrated in Figure 24 is solved by matrix factorization (MF) which exploits the low-rank nature of matrices of real-world data. Mathematically, a n by n matrix of rank r, where $r \ll n$, has only r non-zero singular values and it can be factorized as

$$X = UV^T, \tag{3.1}$$

where U and V are matrices of $n \times r$. In practice, due to data noise, X is often full-rank but with a rank r dominant component. That is, X has only r significant singular values and the others are negligible. In this case, a rank-r matrix $\hat{X}$ can be found that approximates X with high accuracy, i.e.,

$$X \approx \hat{X} = UV^T. \tag{3.2}$$

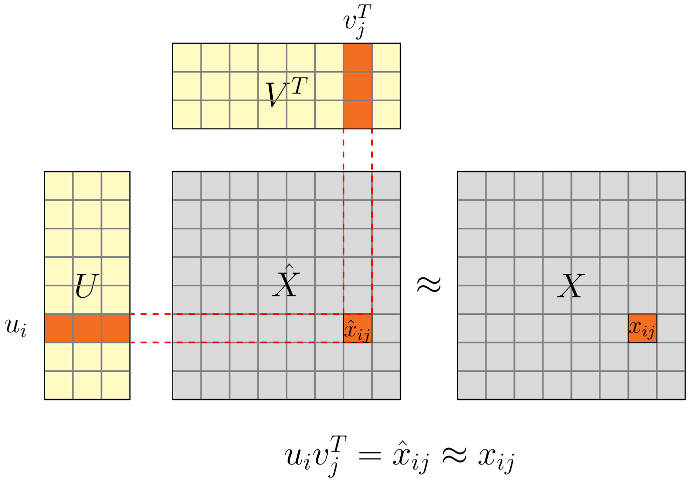


Figure 25: Matrix factorization.

MF can be used for solving the problem of matrix completion, which generally minimizes an objective function of the following form:

$$\min \sum_{ij \in \Omega} l(x_{ij}, u_i v_j^T), \tag{3.3}$$

where Ω is the set of observed entries, x_{ij} is the ijth entry of X, and u_i and v_j are the ith and jth row of U and of V respectively. l is a loss function that penalizes the difference between the two inputs. In words, we search for (U, V) so that $\hat{X} = UV^T$ best approximates X at the observed entries in Ω. The unknown entries in X are predicted by

$$\hat{x}_{ij} = u_i v_j^T, \text{ for } ij \notin \Omega. \tag{3.4}$$

Figure 25 illustrates MF for matrix completion.

3.1.2.2 DMFSGD

It is natural to require that MF be integrated in network applications with a decentralized architecture. To this end, the following measures are taken in the system design:

- (u_i, v_i)s, $i = 1, \ldots, n$ are distributively stored, i.e., (u_i, v_i) is stored at node i. (u_i, v_i) is called the *coordinate* of node i.
- Each node selectively probes a number of other nodes, called *neighbors*. Denote $\mathcal{N}_i$ the neighbor set of node i, $i = 1, \ldots, n$.
- Each node updates its coordinate by collaborating and exchanging messages with its neighbors.
- The system architecture integrates the prediction algorithm and the measurement methodology for the chosen metric.

These measures lead to the so-called *Decentralized Matrix Factorization by Stochastic Gradient Descent* approach (*DMFSGD*). In particular, the neighbors of each node can generally be randomly selected from the set of available nodes in the network. The system architecture needs to be modified for different metrics due to their different measurement methodologies. For example, the measurement of RTT is probed and computed by the sender, whereas that of ABW is probed by the sender but computed by the receiver. Figure 26 and 27 illustrate how DMFSGD works for RTT and for ABW by incorporating their different measurement methodologies. Figure 28 illustrates how a property of a network path is computed from the coordinates of the two end nodes.

3.1.3 DMFSGD implementation and results

We have implemented DMFSGD in Python and tested it on the platform of PlanetLab. The implementation requires the *numpy* library of Python for vector operations and the measurement tool of *ping* for acquiring RTT and of *pathchirp* for acquiring ABW.

We have successfully deployed DMFSGD for RTT on more than 500 Planetlab nodes. To evaluate the performance of DMFSGD, we used the criterion of *stress*, defined as

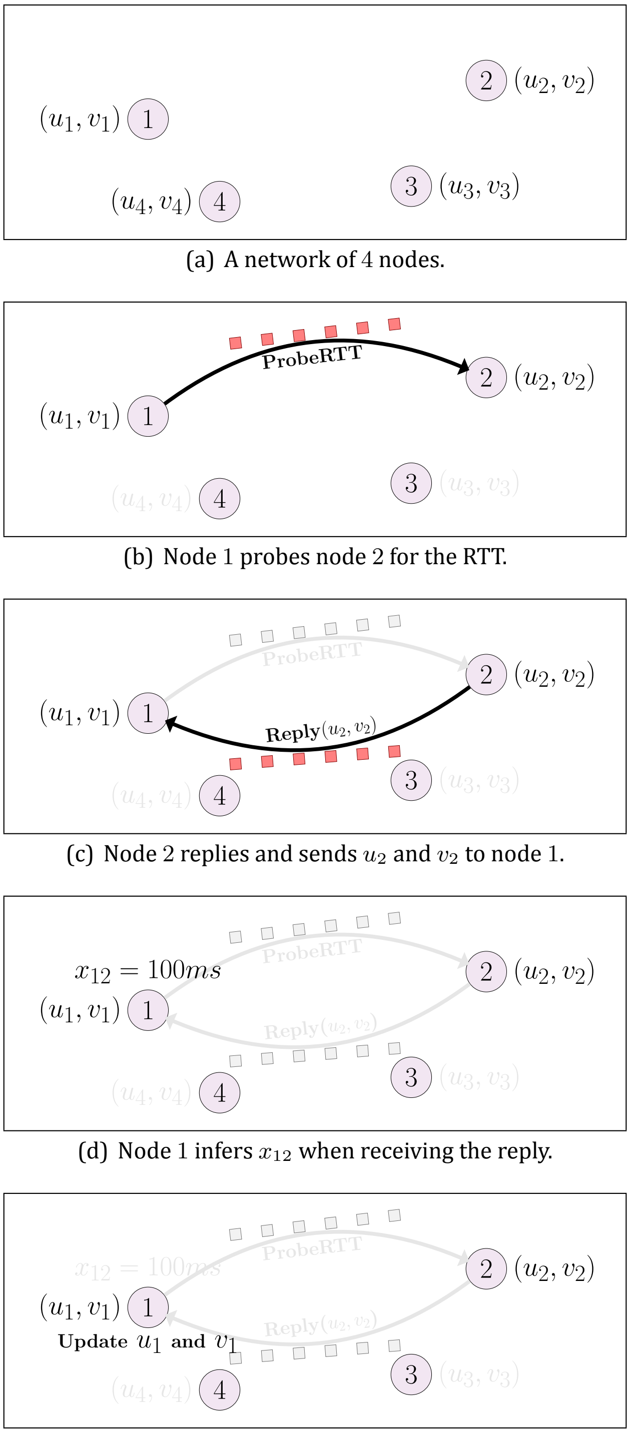
$$stress = \sqrt{\frac{\sum_{i,j=1}^{n}(x_{ij} - \hat{x}_{ij})^2}{\sum_{i,j=1}^{n} {x_{ij}}^2}}.$$

Figure 29 shows the evolution of stress during three days from October 18th, 2013 to October 20th, 2013. The u and v of each Planetlab node and the RTT measurements were collected every 30 minutes. We summarized each network path with the median RTT for this path and calculate the stresses on the predicted RTTs and the median RTTs. It is can be seen that DMFSGD performed stably and produced accurate predictions.

3.1.4 DMFSGD in the mPlane architecture

DMFSGD is a distributed algorithm that executes a high-level reasoning on measurements to predict path properties. It is therefore better understood as an mPlane WP4 algorithm that relies on lower-level measurements from some probes (e.g. RTT and/or available bandwidth measurements), which are considered as WP2 components. DMFSGD is an example of a spatio-temporal technique, because it combines measurements from different locations at different points in time.

One typical use-case application would be in a CDN service where the path characteristics between a client and the available CDN servers should be estimated (to reduce measurement load) in order to direct the client toward the most appropriate server. Another example would be the construction of an overlay network in which the edges are selected on the basis of their estimated properties. In P2P networks peers could also be selected based on some locality awareness inferred in this way.



(a) A network of 4 nodes.

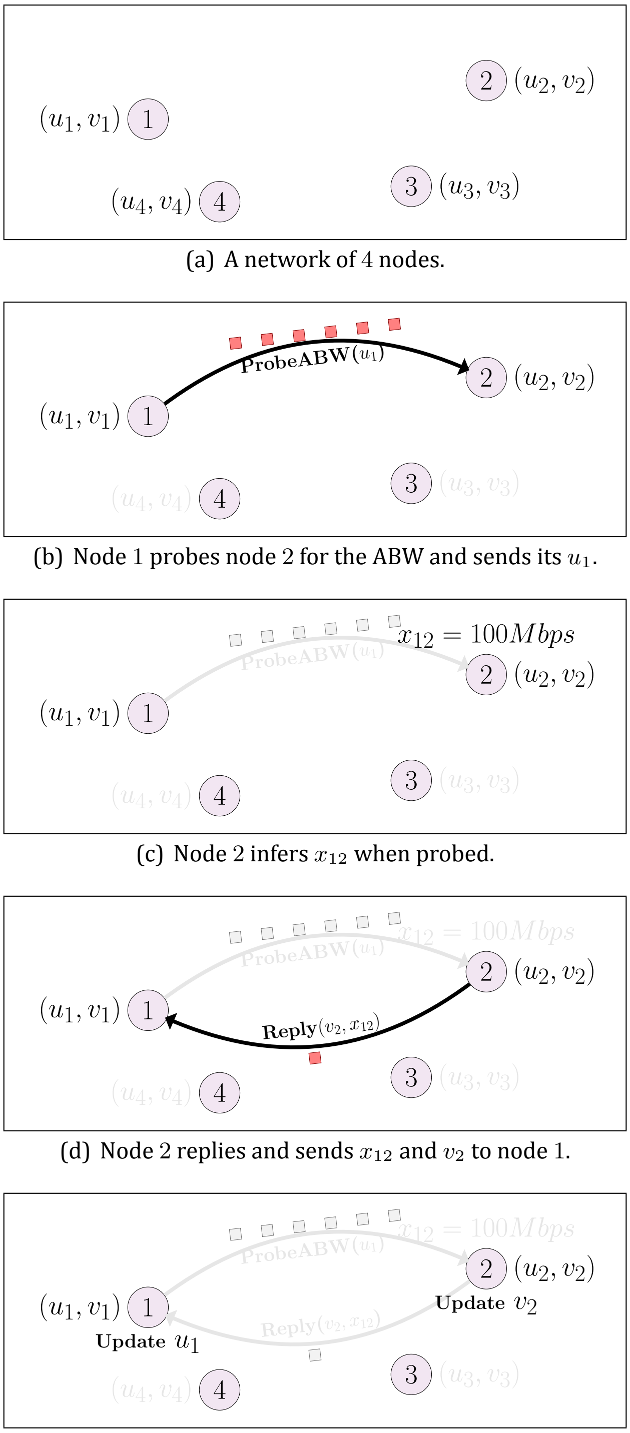
(b) Node 1 probes node 2 for the RTT.

(c) Node 2 replies and sends u_2 and v_2 to node 1.

(d) Node 1 infers x_{12} when receiving the reply.

(e) Node 1 updates u_1 and v_1.

Figure 26: An example that shows how DMFSGD works for RTT.



(a) A network of 4 nodes.

(b) Node 1 probes node 2 for the ABW and sends its u_1.

(c) Node 2 infers x_{12} when probed.

(d) Node 2 replies and sends x_{12} and v_2 to node 1.

(e) Node 1 updates u_1 and node 2 updates v_2.

Figure 27: An example that shows how DMFSGD works for ABW.

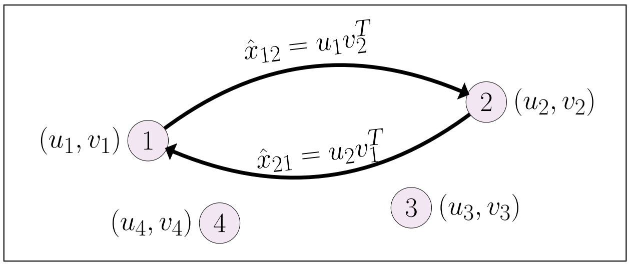


Figure 28: An example that shows how a node infers the performance, either RTT or ABW, of the paths connected to another node. Here, node 1 infers $\hat{x}_{12}$ and $\hat{x}_{21}$ by using its coordinate (u_1, v_1) and by retrieving the coordinate of node 2 (u_2, v_2).

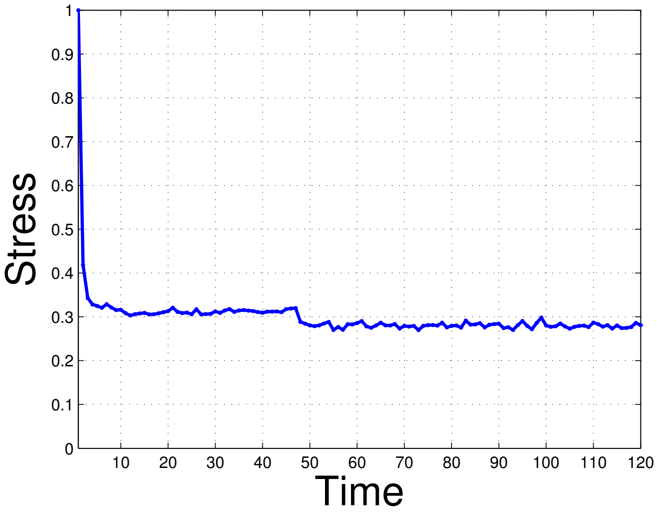


Figure 29: Evolution of the stress on RTT predictions among 500 PlanetLab nodes

3.2 Topology discovery

3.2.1 IGP Weight Inference

Routing simulations, as well as theoretical Internet graph understanding [37], 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 than ten years now, routing authors make use of weighted ISP topologies [52] collected with Rocketfuel [71]. 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. [76] and Coyle et al. [25]). 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 [46].

3.2.1.1 Measurement Mechanisms

We basically rely on the MERLIN platform [54] 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 [57]. 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 [44].

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* [55] 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 [55]. If filtering becomes an issue, we may envision to replace MERLIN by exploreNET [79], 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 [9] campaign from PlanetLab nodes 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.

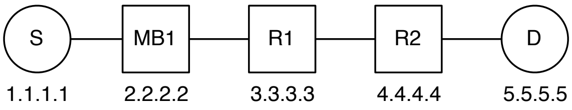
We use the scamper Paris Traceroute implementation [51]. In addition, we enable the MDA algorithm [10] (this corresponds to the `tracelb` option in scamper) to trace all per-flow load-balanced paths between each PlanetLab node and destination.

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.

A threshold must be found between the number of Paris Traceroute source and the accuracy of resulting weights. For being accurate, weights requires constraints on the sytem. The constraints can be revealed with the traffic injected on the topologies. We evaluate the potential accuracy of traffic injected using the *coverage metric*, i.e., the fraction of (source, destination) pairs of the TR-Graph (i.e., directed graph made by IGMP nodes and links augmented with Paris Traceroute unicast links) that are connected by a trace revealed by our campaign.

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. 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 ad-



(a) topology

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

(b) output

Figure 30: `tracebox` example

ditional 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.

3.2.2 Tracebox

`tracebox` is a `traceroute` [81] successor that enables network operators to detect which middleboxes modify packets on almost any path. `tracebox` allows one to easily generate complex probes to send to any destination. By using the quoted packet inside of ICMP replies, it allows to identify various types of packet modifications and can be used to pinpoint where a given modification takes place.

`tracebox` embeds LUA [42] bindings to allow a flexible description of the probes as well to ease the development of more complex middlebox detection scripts. `tracebox` aims at providing the user with a simple and flexible way of defining probes without requiring a lot of lines of code. `tracebox` indeed allows to use a single line to define a probe (see as example the argument `-p` of `tracebox` in Fig. 3.30(b)) similarly to Scapy [14]. `tracebox` provides a complete API to easily define IPv4/IPv6 as well as TCP, UDP, ICMP headers and options on top of a raw payload. Several LUA scripts are already available and allows one to detect various types of middleboxes from Application-level Gateways to HTTP proxies.

Sec. 3.2.2.1 and 3.2.2.2 give examples of algorithms or scripts that can be given as input to `tracebox`.

3.2.2.1 Proxy Detection

`tracebox` can also be used to detect TCP proxies. To be able to detect a TCP proxy, `tracebox` must be able to send TCP segments that are intercepted by the proxy and other packets that are forwarded beyond it. HTTP proxies are frequently used in cellular and enterprise networks [83]. Some of them are configured to transparently proxy all TCP connections on port 80. To test the ability of detecting proxies with `tracebox`, we used a script that sends a SYN probe to port 80 and, then, to port 21. If there is an HTTP proxy on the path, it should intercept the SYN probe on port 80 while ignoring the SYN on port 21. We next analyze the ICMP messages returned.

We deployed tracebox on the PlanetLab testbed and we identified two oddities. First, we found an HTTP proxy or more probably an IDS within a National Research Network (SUNET) as it only operated for a few destinations and that the path for port 80 was shorter than for port 21. Second, and more disturbing, tracebox identified that several destinations where behind a proxy whose configuration, inferred from the returned ICMP messages, resulted in a forwarding loop for probes that are not HTTP. We observed that the SYN probe on port 21, after reaching the supposed proxy, bounced from one router to the other in a loop as tracebox received ICMP replies from one router then another alternatively.

3.2.2.2 NAT Detection

NATs are probably the most widely deployed middleboxes. Detecting them by using tracebox would likely be useful for network operators. However, in addition to changing addresses and port numbers of the packets that they forward, NATs often also change back the returned ICMP message and the quoted packet. This implies that, when inspecting the received ICMP message, tracebox would not be able to detect the modification.

This does not prevent tracebox from detecting many NATs. Indeed, most NATs implement *Application-level Gateways* (ALGs) [72] for protocols such as FTP. Such an ALG modifies the payload of forwarded packets that contain the PORT command on the ftp-control connection. tracebox can detect these ALGs by noting that they do not translate the quoted packet in the returned ICMP messages. This detection is written as a simple script (shown in Fig 31) that interacts with tracebox. It builds and sends a SYN for the FTP port number and, then, waits for the SYN+ACK. The script makes sure that the SYN+ACK is not handled by the TCP stack of the host by configuring the local firewall (using the filter functionality, shown at line 7, of tracebox that configures iptables on Linux and ipfw on Mac OS X). It then sends a valid segment with the PORT command and the encoded IP address and port number as payload. tracebox then compares the transmitted packet with the quoted packet returned inside an ICMP message by an RFC1812-compliant router and stores the modification applied to the packet. If a change occurs and a callback function has been passed as argument, tracebox triggers the callback function. In Fig 31, the callback cb checks whether there has been a payload modification. If it is the case a message showing the approximate position of the ALG on the path is printed (see line 29).

3.2.2.3 Results

We deployed tracebox on PlanetLab, using 72 machines as vantage points (VPs). Each VP had a target list of 5,000 items build with the top 5,000 Alexa web sites. Each VP used a shuffled version of the target list. DNS resolution was not done before running tracebox. This means that, if each VP uses the same list of destination names, each VP potentially contacted a different IP address for a given web site due to the presence of load balancing or Content Distribution Networks. Our dataset was collected during one week starting on April 17, 2013.

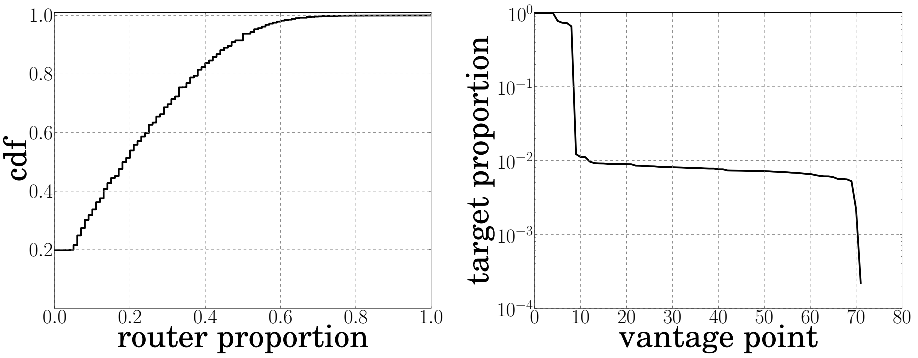
With this deployment, we validated and demonstrated the usefulness of tracebox based on three use cases. First, we looked at the proportion of RFC1812-compliant routers and their locations and showed that, in 80% of the cases, a path contains at least one RFC1812-compliant routers, as shown on Fig. 3.32(a). Second, we used tracebox to demonstrate that middleboxes, along a path, may

```
-- NAT FTP detection
-- To run with: tracebox -s <script> <ftp_server>
-- Build the initial SYN (dest is passed to tracebox)
syn = IP / tcp{dst=21}
-- Avoid the host's stack to reply with a reset
fp = filter(syn)
synack = tracebox(syn)
if not synack then
        print("Server did not reply...")
        fp:close()
        return
end
-- Check if SYN+ACK flags are present
if synack:tcp():getflags() ~= 18 then
        print("Server does not seems to be a FTP server")
        fp:close()
        return
end
-- Build the PORT probe
ip_port = syn:source():gsub("%.", ",")
data = IP / tcp{src=syn:tcp():getsource(), dst=21,
        seq=syn:tcp():getseq()+1,
        ack=synack:tcp():getseq()+1, flags=16} /
        raw('PORT '.. ip_port .. ',189,68\r\n')
-- Send probe and allow cb to be called for each reply
function cb(ttl, rip, pkt, reply, mods)
        if mods and mods:__tostring():find("Raw") then
                print("There is a NAT before " .. rip)
                return 1
        end
end
tracebox(data, {callback = "cb"})
fp:close()
```

Figure 31: Sample script to detect a NAT FTP that can be sent to tracebox

modify the TCP sequence number. Finally, the third use case for `tracebox` concerns middleboxes that modify the TCP MSS option. This TCP option is used in the SYN and SYN+ACK segments to specify the largest TCP segment that a host sending the option can process. In an Internet that respects the end-to-end principle, this option should never be modified. Fig. 3.32(b) provides, for each vantage point (the horizontal axis), the proportion of paths (the vertical axis, in log-scale) where the MSS option has been changed. We see that a few VPs encountered at least one MSS modification on nearly all paths while, for the vast majority of VPs, the modification is observed in only a couple of paths.

We further performed some tests with `tracebox` to verify whether the recently proposed Multipath TCP [33] option could be safely used over the Internet. This is similar to the unknown option test performed by Honda et al. [40]. However, on the contrary to Honda et al., `tracebox` allows one to



(a) Proportion of RFC1812-compliant routers on a path

(b) Vantage points proportion modifying MSS

Figure 32: `traceroute` results

probe a large number of destinations. To our surprise, when running the tests, `tracebox` identified about ten Multipath TCP servers based on the TCP option they returned. One of those server, `www.baidu.com`, belongs to the top 5 Alexa. All these servers where located inside China. A closer look at these options revealed that these servers (or their load balancers) simply echo a received unknown TCP option in the SYN+ACK. This is clearly an incorrect TCP implementation.

3.3 One way delay variation (ΔOWD) measurement

In order to infer users' Quality of Experience (QoE), two main approaches can be devised. One is to consider a specific application and define relevant subjective metrics (e.g., stuttering for VoD streaming, completion time for BitTorrent file sharing, etc.). Another one is to describe general, more objective, metrics that can be measured at the network or transport layer, that usually undergoes the name of Quality of Service (QoS), and use models for mapping these objective metrics to subjective quality.

Delay is among the most relevant objective metrics, as it correlates with the performance of almost any application. Indeed, even the performance of bulk data transfers, that are usually considered to be delay-insensitive and only driven by bandwidth, are affected by dealy as we will show in this section. As communications are bidirectional, it is generally possible (and easy) to measure bidirectional delay, usually known as Round Trip Time (RTT) or mono-directional delay, usually referred to as One Way Delay (OWD). Being able to reliable measure OWD delay is important (as congestion may happen only on a single direction), and the OWD metric decorrelates thus performance of both paths.

Yet, OWD is notoriously difficult to precisely measure, which is easy to see in the Internet where hosts are generally non-collaborative and non-synchronized. However, we point out that Internet delay is made up by several contributions, notably to serialize and propagate the signal, as well as delay due to queuing. While serialization and propagation delays are constant, the queuing delay component is variable, and as home gateways can buffer seconds worth of traffic in the current ADSL and cable settings, the queueing component of OWD it of extreme interest.

It follows that, while absolute OWD is hard to measure (as it contains an error due to clock off-

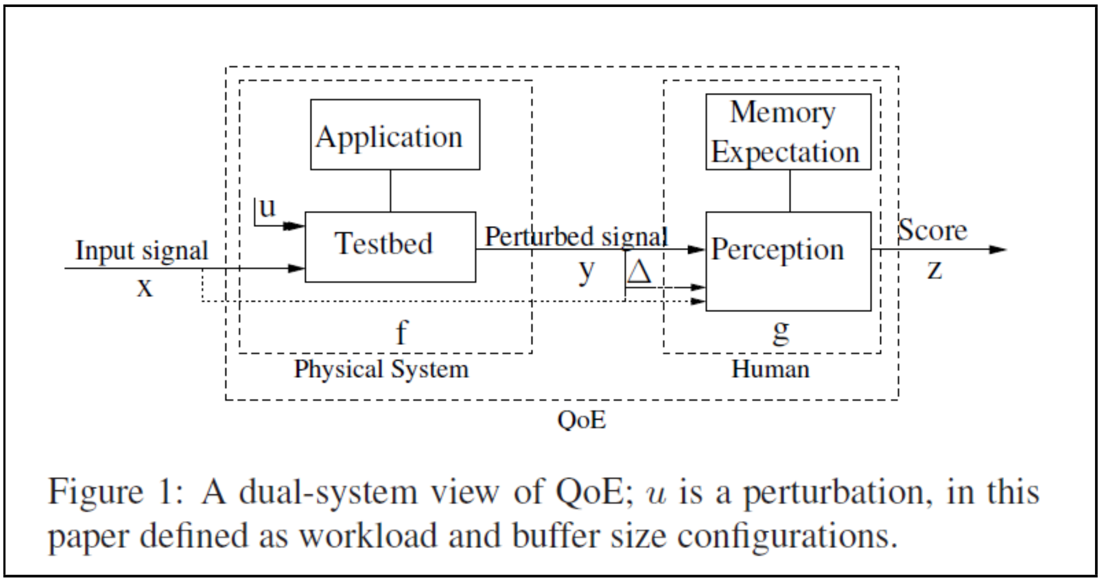


Figure 1: A dual-system view of QoE; u is a perturbation, in this paper defined as workload and buffer size configurations.

Figure 33: QoS to QoE mapping: model for gathering QoE MOS scores for varying traffic conditions (courtesy of [39])

set), variations of OWD are easier to measure (as the difference between two OWD samples on the same path cancels the error) and are telling at the same time, as they can be correlated to user QoE (Sec. 3.3.1). This is especially interesting for use cases targeted to troubleshooting of specific appllications (e.g., multimedia of Sec. 2.4 and Web of Sec. 2.5).

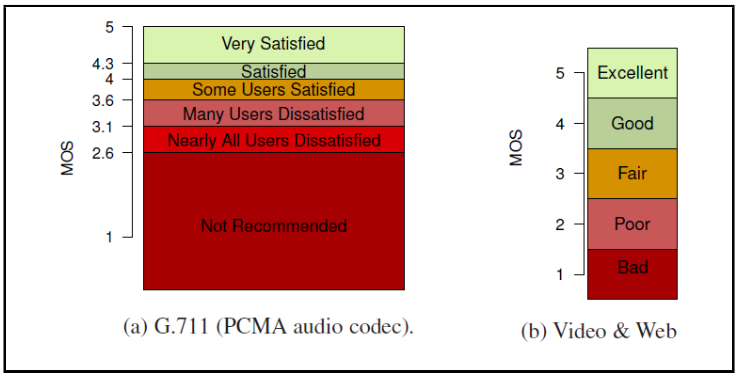
In what follows, we denote by ΔOWD the one way queuing delay measurement: we focus on measuring it with passive (Sec. 3.3.2) vs active (Sec. 3.3.3) methodologies, especially highlighting the implication of ΔOWD (as the methodologies themselves are the object of mPlane work package WP2 and are described therein). Ultimately, the comparison of passive and active methodologies to measure queuing delay, should assist the definition of mPlane reasoners to exploit the best methodology in any particular situation.

We finally show that ΔOWD has an impact also on bulk transfers, considering BitTorrent as an example (Sec. 3.3.4), which assist in refining and completing the mapping between QoS and QoE (of Sec. 3.3.1) to a larger set of applications.

3.3.1 QoS to QoE mapping via ΔOWD measurement

As mPlane aims at avoiding reinventing the wheel, it would be a shame to ignore exisiting valuable knowledge and reproduce experiments to perform QoS to QoE mapping. Correlation between delay and QoE is pointed out for specific applications such as Web [15, 74], multimedia [18] or P2P [77, 78], and by [39] that oversees all applications at once. In this subsection, inspired by a ``standing on the giant shoulder'' philosophy, we describe the mapping proposed in [39], extensively using material reported there (figures taken from [39] are properly highlighted to differentiate them from own figures). However, our purpose here is on providing brief coverage of the mapping and simple ``rules of thumbs'' to the mPlane Reasoner, so that we refer the reader to [39] for more details.

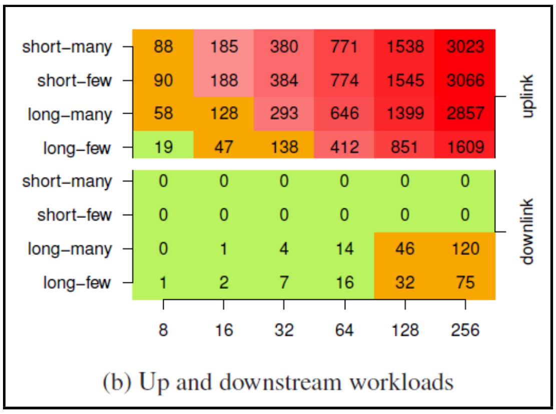
Authors in [39] adopt an experimental approach, where they perturb a system with controlled cross



(a) G.711 (PCMA audio codec).

(b) Video & Web

Figure 34: QoS to QoE mapping: Mean Opinion Score (MOS) scales to measure QoE (courtesy of [39])



		8	16	32	64	128	256
uplink	short–many	88	185	380	771	1538	3023
	short–few	90	188	384	774	1545	3066
	long–many	58	128	293	646	1399	2857
	long–few	19	47	138	412	851	1609
downlink	short–many	0	0	0	0	0	0
	short–few	0	0	0	0	0	0
	long–many	0	1	4	14	46	120
	long–few	1	2	7	16	32	75

(b) Up and downstream workloads

Figure 35: QoS to QoE mapping: mean queuing delay ΔOWD (in ms) for the access networks testbed with different buffer size (x-axis) and workload (y-axis) configurations, with highlighted MOS scale (courtesy of [39])

traffic in a testbed, and observe impact on QoS and QoE. The general model they refer to is reported in Fig. 33, where at the input they consider a large number of cross traffic scenarios (to mimick heterogenity of Internet traffic), and numerous settings as far as the buffer size is concerned (to mimick the heterogenity of home router devices). As output, the system yield a Mean Opinion Score (MOS) evaluation for different services (namely, voice, video and Web), whose scales are reported in Fig. 34.

Then, for each combination of cross-traffic and buffer size, authors performs tests for voice, video and Web services, and measure user MOS. The most important binding between our work in mPlane and the exisiting knowledge of QoE to QoE mapping of [39] is reported in Fig. 35. More precisely, as for each cross-traffic vs buffer-size pair, the figure reports the mean queuing delay ΔOWD (in ms) for the access networks testbed with different buffer size (x-axis) and workload (y-axis) configurations. The figures also highlights the MOS scale, where delays that significantly degrade the QoE of interactive applications (ITU-T Recommendation G.114) are colored in red.

Finally, detailed MOS ``heatmaps'' are provided in [39] for each of the voip, video and Web services (part of which we report in Fig. 36) that correlate ΔOWD queueing delay due to cross-traffic and

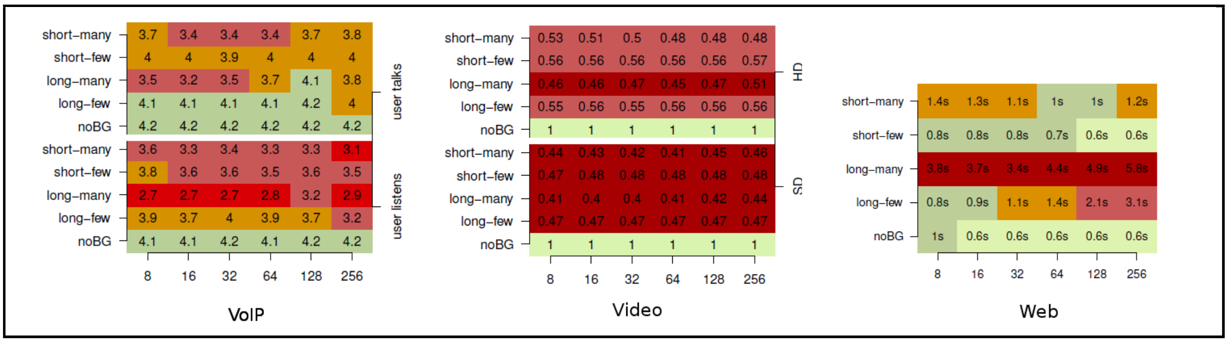


Figure 36: QoS to QoE mapping: Mean Opinion Score (MOS) scales to measure QoE (courtesy of [39])

buffer-size to the specific QoE of that service. As our work focuses on measuring ∆OWD in the Internet with active or passive methodologies, the above heatmaps consitute the binding from QoS to QoE.

3.3.2 QoS. Passive ∆OWD measurement (Tstat)

We have developed several passive methodologies that let us measure ∆OWD measurement of faraway or local hosts. In case of faraway hosts, this give us an opportunity to ``spy'' on the queuing delay of Internet hosts, gathering a relevant sample of the ∆OWD users suffer in their typical activities. In the case of local hosts, further knowledge of the rest of user traffic allow us to correlate ∆OWD sufferered by an application to the others application that may have caused it.

3.3.2.1 Faraway hosts

To study faraway hosts, we focus on BitTorrent, that is very popular and thus allows us to reach users across all countries and Internet AS, and futher allows us to compare the impact of transport level protocols as well. This section summarizes findings of [22] and [20], where we apply the methodology developed in [21].

Our internal BitTorrent peers exchanged with 24,678 external peers, about half of which transferred data using LEDBAT (the new low-delay BitTorrent transport protocol). Clearly, an host may participate into multiple swarms. Furthermore, as our experiments are repeated with different clients, we can thus encounter the same external peer multiple time over our experimental campaign. This is indeed the case, as we find 8914 unique IP addresses. Among these peers, in order to gather statistically meanignful results, we require a minimum amount of queuing delay samples q_i^B per peer: we only consider peers yielding at least 25 queuing delay samples (or 50 packets). Notice that, in case we encounter a peer multiple times in our experiments, we count it (at most) once per each experiments in which it appears (for each experiment, the peer is considered provided that it sends 50 packets). In the reminder of the analysis, we focus on this peer subset, consisting of 6,896 peers (1,931 unique IPs). First, we start by considering the distribution of the queuing delay percentiles over all peers. Intuitively, the 90, 95 and 99-th percentiles statistics are related to the queuing delay experienced by a peer during the most highly loaded 10%, 5% and 1% observation

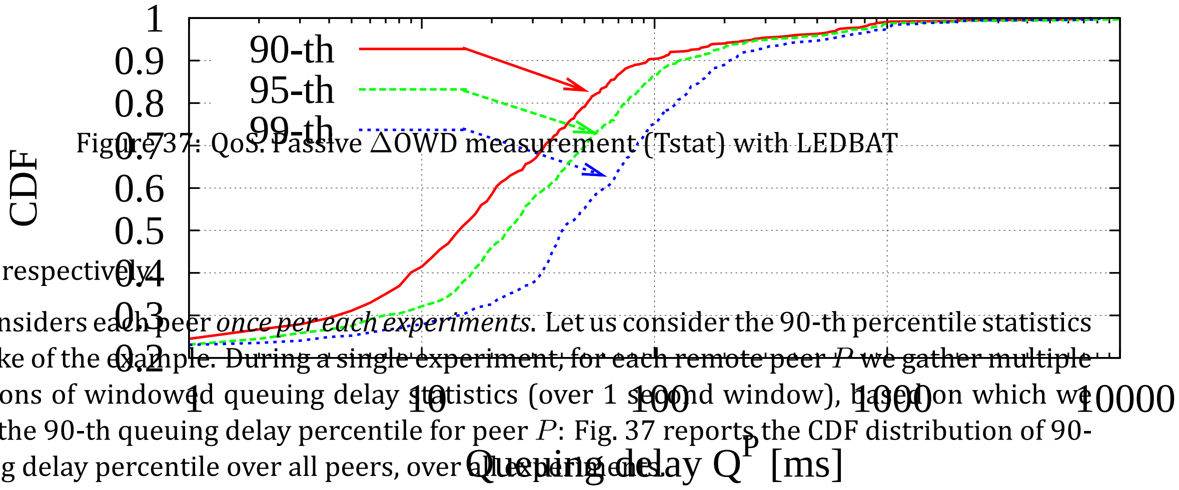


Figure 37: QoS: Passive ΔOWD measurement (Tstat) with LEDBAT

windows respectively.

Fig. 37 considers each peer *once per each experiments*. Let us consider the 90-th percentile statistics for the sake of the example. During a single experiment, for each remote peer P we gather multiple observations of windowed queuing delay statistics (over 1 second window), based on which we compute the 90-th queuing delay percentile for peer P: Fig. 37 reports the CDF distribution of 90-th queuing delay percentile over all peers, over all experiments.

First, notice that part of the samples report a very low queuing delay (i.e., below 1ms): these corresponds to cases where external leechers receive data from our probes, and we are receiving in turn only an acknowledgement stream. In this case, since queuing is in our upstream, the low-bandwidth acknowledgement stream is not incurring any access bottleneck and thus traverse an empty queue. Notice that, additionally, the remote peer is also not sending data to other peers either, as otherwise we would observe a rise in queuing delay.

Otherwise, we gather that 99% of the windows of the median peer have a queuing delay below 60 ms. Moreover, only 10% of the peers experience a 90-th (99-th) percentile above 100 ms (200 ms). In other words, for the 10% of peers that are most affected by bufferbloat, only 10% (1%) of their 1-second windows incurr more than 100 ms (200 ms) queuing delay. Finally, only 1% of peers have a 90-th (99-th) queuing delay percentile above 1 s (2 s).

Summarizing, from Fig. 37 we gather that (i) the bulk of peers experience queuing delays that are generally non-harming for interactive traffic: i.e., for 90% of peers, 90% of windows have a queuing delay smaller than 100 ms. At the same time (ii) some users may perceive the rest of their activities annoyed by the ongoing BitTorrent transfers, since about 1% of the peers have at least 10% of windows have a queuing delay exceeding 1 s.

From other experiments (reported in [20] but unreported here for the sake of simplicity) we understand that these high-delay windows mostly concern a small population of very active BitTorrent users, whose Internet activities can suffer from significant bufferbloat. Since a single TCP connection is able to fill the buffer, LEDBAT transfers to our probe are however not the cause of their bufferbloat, which is rather tied to TCP uploads to some other peers in the swarm.

Hence, we gather that mPlane reasoners can gather *passive measurement* of ΔOWD delay from LEDBAT BitTorrent traffic. At the same time, delay information is not only associated to LEDBAT BitTorrent traffic, as delay is possibly caused by (or suffered from) other traffic sources (e.g., multimedia, Web, etc.) sharing the link with BitTorrent. The delay information leveraged by these methods can then be correlated to application performance (e.g., multimedia, Web, etc.) or use for troubleshooting.

3.3.2.2 Local hosts

To study local hosts, we are no longer restricted to BitTorrent. This section summarizes findings gathered in [7] (technical report under submission), where we apply the methodology developed in [8] to analyze offline traces gathered in a single ISP network participating in the mPlane project (8 hr-long daily trace, gathered during 2009 when P2P was still popular).

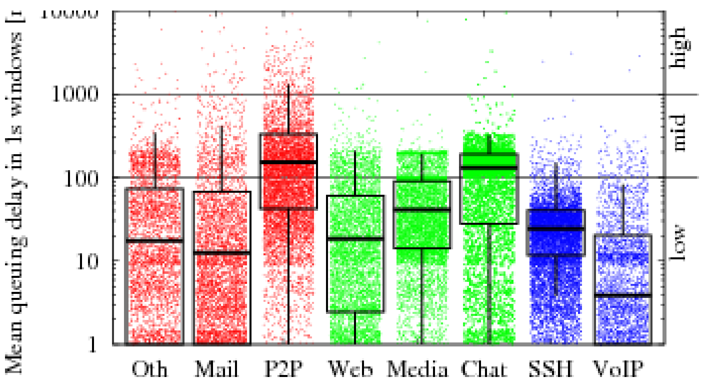
We argue that in order to give statistics that are useful from the user perspective, we need to batch consecutive samples (e.g., belonging to the same TCP burst) into windows whose duration relates with the timescale typical of user dynamics (as done in the previous section, as motivated in [20], we use windows of 1 second). For the network under observation, we consider each internal IP as a single[1] host. For each host, we collect delay samples for each active flow, corresponding to the average queuing delay over short time windows of 1 second duration, as estimated by valid data-ack pairs of each flow. Overall, our processing gathers about 10^7 individual per-flow samples.

Each delay sample carries an application label, obtained through Tstat Deep Packet Inspection (DPI) and behavioral classification capabilities. Though Tstat is capable of fine-grained classification of different applications [32], we cluster similar applications into few classes depending on the service they offer (namely, Mail, Web, Multimedia, P2P, SSH, VoIP, Chat and other uncategorized applications). To the best of our knowledge, mPlane is the first to report a detailed per-application view of the Internet queuing delay -- that depends on the traffic mix and user behavior of each household. We present our results in Fig. 38 where we depict a jittered density map of queuing delay samples (y-axis) for different application classes (x-axis), along with boxplots reporting the quartiles (and 5th, 95th percentiles). Applications are ordered, left to right, in increasing order of delay sensitivity. It can be seen that, for most applications the 75% of windows experience less than 100ms worth of queuing. The only exceptions are consituted by, rather unsurprisingly, P2P applications and, somehow more surprisingly, Chat applications, with *median* delays exceeding 100ms.

Let us dig further the implication of the above observations. Due to studies assessing the impact of delay on the QoE of several applications such as Web [15, 74], multimedia [39, 18] or P2P [77, 78] applications, we can easily map a QoS metric such the queuing delay, into an coarse indication of QoE for the end-user as reported in Sec. 3.3.1. Based on the above work, we set two thresholds at 100 ms and 1 second, such that:

- performance of interactive multimedia (e.g., VoIP, video-conference and live-streaming) or data-oriented applications (e.g., remote terminal or cloud editing of text documents) significantly degrades when the first threshold is crossed [39, 18];
- performance of mildly-interactive application (e.g., Web, chat, etc.) significantly degrades when the second threshold is crossed [15, 74];

[1] This is known to be simplistic as, due to the penetration of NAT devices, the same IP is shared by multiple hosts (50% of the cases [53]), that are possibly active at the same time (10% of the cases). Yet it has no impact for our methodology, since these potentially multiple hosts share the same access bottleneck link



(a)

	Delay		
App	*Low*	*Mid*	*High*
Other	89.7	10.2	0.1
Mail	93.2	6.2	0.6
P2P	58.4	38.7	2.9
Web	91.9	8.0	**0.1**
Media	86.8	**13.2**	0
Chat	45.7	**54.1**	**0.1**
SSH	98.6	**1.4**	0.0
VoIP	97.8	**2.2**	0.0

(b)

Figure 38: QoS. Passive ΔOWD measurement (Tstat) with TCP: Breakdown of queuing delay per application.

- additionally, while bulk transfers (e.g., P2P, long TCP connections) are elastic in nature, it has been shown that also TCP performance degrades [34] in presence of excessive buffering (i.e., control becomes unstable due to absence/delay of feedback information) and furthermore queuing delay affect control plane of P2P applications [77, 78] -- so that even these applications performance start to degrade when the second threshold is crossed.

Additionally, Fig. 38 tabulates the percentage of 1 sec windows for each application that fall into either of the three delay regions, where we use boldface values to highlight possible QoE degradation. It follows that, in practice, bufferbloat impact on QoE appears to be modest. A limited 0.1% of Web and Chat sessions may be impacted by significant delay, and 2.2% (1.4%) of VoIP (remote terminal) sessions may be impacted by moderate delay. P2P clearly stand out, raising the odds to induce high delays (2.9%) followed by SMTP (0.6%), though with likely minor impact for the end-users.

Hence, we gather that mPlane reasoners can gather *passive measurement* of ΔOWD delay from virtually any application, exp, that they can thus correlate to application performance (e.g., multimedia, Web, etc.) or use for troubleshooting.

3.3.3 QoS. Active ΔOWD measurement (TopHat)

Yet another option is to engage with faraway users with an active methodology. In this case, we are no longer restricted to a specific application (e.g., BitTorrent). As we are currently undergoing a throrough measurement campaign, in this section we only report preliminary results of a moderate-scale campaign (where we already gather nearly a billion samples). More details are available in a technical report (currently under submission [17]).

We focus on moderate number of hosts O(10^4) on the same ISPs, that we continuously probe at 0.5 Hz frequency from 2 separate PlanetLab nodes for a period of about 8 continuous hours. Our fast ping tool developed for TopHat during mPlane is able to handle about 25K ICMP probes per second, and we conservatively set a target set size of 75% this bound (i.e., 18,750 hosts, about one order of magnitude more than [74]). Overall, we receive replies to 47% of our sent packets, for a total of O(10^8) valid samples -- using only two PlanetLab servers, we already achieve a quite significant scale in terms of spatial reach and temporal frequency.

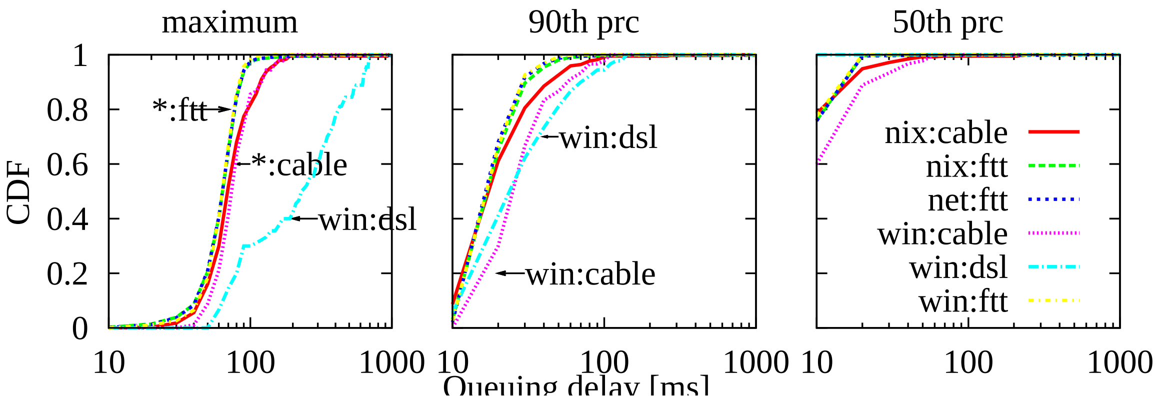


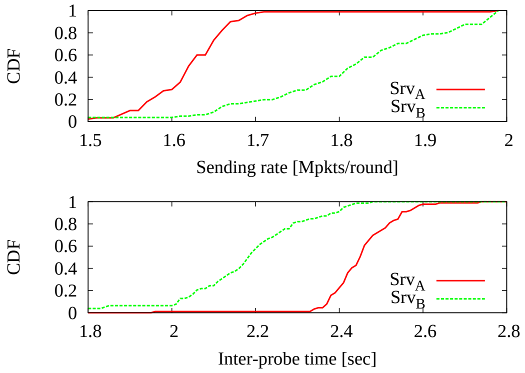
Figure 39: QoS. Active ΔOWD measurement (TopHat). Validation of the Internet measurement campaign: CDF of maximum (left), 90-th percentile (center) and median (right) delay, for hosts with reliable reverse DNS and OS fingerprint information.

We need to ensure that the methodology can yield to some significant insights in terms of the gathered statistics in the wild Internet. For validation purposes, we therfore infer (i) the access type (AT) of our target hosts by issuing reverse DNS queries, as well as (ii) the remote operating system (OS) through `nmap` fingerprinting. As for the access type, the ISP we are targeting offers DSL, FTTH and cable access: similarly to [20], we expect the breakdown of queuing delay along AT to yield an intuitive validation of the observed statistics. As for the OS, we do not argue queuing delay performance to be (strongly) tied to the OS (despite we still expect difference to arise due to, e.g., different default TCP congestion control flavors across OSs).

Specifically, we argue that in case the remote OS is reliably found to be a Windows OS, then we can very likely rule out the case of NATted access: more likely, we hit a private end-host or a server of a small-size business professional[2] Overall, we manage to infer both AT and OS information for 2546 hosts: as for the OS breakdown, the majority of hosts are reported to be some Linux variants (most of which are likely NAT devices), followed by known home gateway boxes (denoted with *net*), and 12 are Windows servers; as for the AT, the majority of hosts has fiber access (denoted with *ftt*), followed by cable and 6 DSL lines. Hence, this highly unbalanced validation subset does not allow to report any statistically meaningful per-AT or per-OS conclusions -- but nevertheless allows to validate our methodology.

We collect per-host percentiles during 5 minutes windows: Fig. 39 reports the Cumulative Distribution Function (CDF) of a few per-host queuing delay statistics, gathered over all hosts and measurement rounds (in other words, each host yields a sample for each CDF during a different 5 min round). In more details, top plot reports the maximum queuing delay CDF: as expected, from the picture clearly emerges that (a) fiber access suffers the lowest delays irrespectively from the OSs, (b) cable delays are only slightly higher, whereas (c) DSL end-hosts may seldom suffer from delays close to 1 sec. We further report the 50th (bottom) and 90th (middle) percentiles CDF. Notice further that (d) from practical purposes, the 90th percentile is lower than 100 ms under any combination of OS and AT -- including end-hosts behind DSL. Moreover, since the *win:cable* and *win:dsl* lines now clearly separate from the others, we infer that the methodology needs to be refined as

[2] In case the Windows OS runs an HTTP server, we empirically verify our intuition by manually browsing to the Webpage for a handful of cases. Residential users could instead be classified via Spamhaus/PBL.



Queuing delay	Srv_A		Srv_B	
Percentiles	Mean	Var	Mean	Var
50th	19.3	2.8	17.5	2.9
75th	27.1	8.0	25.1	8.2
90th	38.4	17.9	36.2	17.6

Figure 40: QoS. Active ΔOWD measurement (TopHat). Validation of the measurement campaign: queuing delay statistics and performance indicators for two different servers (same target set, same time).

it likely underestimates bufferbloat delay for NATted hosts -- although observation (d) suggests bufferbloat not to be relevant.

As a further validation, we contrast the queuing delay statistics relative to the the overall set of 18,750 hosts, probed during the same timeframe by two independent PlanetLab nodes. In principle, we expect bufferbloat measurement to be the same irrespectively of the measurement server. Yet, we need to rule out the fact that the mixture of applications running on other slices of the same PlanetLab nodes negatively affect the measurement.

To conduct rigorous analysis, we not only compare scalar queuing delay statistics (i.e., mean and variance, over all hosts and rounds, of the queuing delay percentiles, tabulated in Fig. 40) but also report CDF of the actual probe rate (in million packets per round) and effective inter-probe time (in seconds). Results confirm that, despite individual agents may run on PlanetLab nodes running a mix of different applications, (i) queuing delay statistics agree on a 2 ms accuracy and (ii) TopHat agents running on PlanetLab are able to send 1.5-2 millions ICMP probes per 5 minutes round, with an inter-probe gap for the same host of 2-2.5 seconds.

Hence, we gather that mPlane reasoners can gather active measurement of ΔOWD delay, achieveing significant spatial scale and high frequency: with minimal intrusiveness, the technique yields accurate and coherent results, that reasoners can thus relate to application performance (e.g., multimedia, Web, etc.) or use for troubleshooting.

3.3.4 QoE. Impact of ΔOWD on BitTorrent completion time

Finally, as far as applications for which a QoS to QoE mapping is not reported in Sec. 3.3, an experimental technique similar to the one performed in Sec. 3.3 can be envisioned. We consider for instance the BitTorrent file-sharing applications, with about 300,000,000 users worldwide, whose main Quality of Experience indicator is the torrent completion time.

In the BitTorrent case, completion time can be affected by several factors, including for instance the congestion control algorithm adopted for data exchange. It is indeed well known that the TCP family possibly generate large queueing delay [34], though the actual queuing delay is further correlated to the specific TCP flavor within the large TCP family. For instance, IETF endorses NewReno [6],

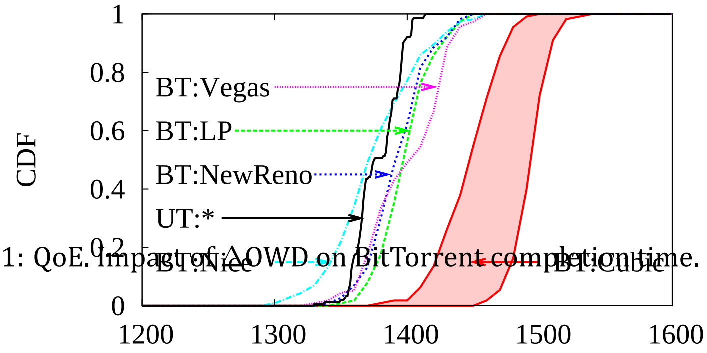


Figure 41: QoE. Impact of ΔOWD on BitTorrent completion time.

a high-priority loss-based congestion control algorithm for TCP. TCP Vegas [16] was proposed as a high-priority delay-based congestion control algorithm as an alternative to the traditional loss-based NewReno algorithm. Recent evolution of loss-based algorithms include TCP Cubic [69] and TCP Compound [75]. Cubic has become the default algorithm for TCP in Linux since kernel version 2.6.18 and Compound the default one in Windows. TCP LP [47] is a low-priority loss-based congestion control algorithm available in Linux, and TCP Nice [82] and uTP/LEDBAT [62] are two low-priority delay-based congestion control algorithms that react on Round Trip Time (RTT) and One Way Delay (OWD) variations respectively. Unlike Nice, uTP is implemented at the application layer over UDP framing explicitly limits the additional delay it adds to the bottleneck buffer (this limit is currently set to 100 ms to avoid harming interactive, delay-sensitive traffic [58]).

It follows that, depending on the specific TCP protocol in use, or alternative to TCP as in the case of uTP/LEDBAT, the (i) performance of BitTorrent may differ, as well as (ii) the queuing delay that BitTorrent traffic induces on competing applications at the bottleneck link. In this section, we show both aspects with an experimental approach on a private cluster where considering a flash crowd scenario[3] in which 75 leechers join a swarm initially populated by a single seed (that distributes a 100 MBytes file). We conduct several experiments in [78], under several scenarios, to both get the BitTorrent QoE (Sec. 3.3.4.1) as well as the implication of BitTorrent on the QoS of other applications (Sec. 3.3.4.1) for different TCP flavors. Here we limidtedly report the most important findings from the perspective of an mPlane reasoner.

3.3.4.1 Impact of TCP flavor on BitTorrent completion time

We first focus on the download completion time. We study the impact of the congestion control algorithms on the data plane efficiency and control plane timeliness by using the uTorrent 3.0 (UT for short) client and an instrumented version of the mainline BitTorrent 4.0.2 (BT for short) client [1]. For completeness, we consider different applications (BT, UT), congestion control algorithms (Cubic, uTP, NewReno, Vegas, Nice, LP), and scenarios (homogeneous, heterogeneous) in the mix.

All outgoing connections of a BT peer will use the same congestion control algorithm, because the operating system imposes a single algorithm at the *transport-layer* (CC-L4). However, uTP is implemented at the *application-layer* (CC-L7) and is only available for uTorrent. UT implements a dual-stack connection management, so that: (i) a uTP and a TCP connections are opened in parallel; (ii)

[3] Each machine of the cluster runs a single peer, seed or leecher.

in case a uTP connection is successfully established, the TCP one is dropped; (iii) connections are bidirectional, so that a uTP incoming connection can be used in the reverse direction. We notice that it is possible to disable uTP in UT with an application level setting (`bt.transp_disp` in the GUI). In the following, we consider the UT:Cubic variant as well, in order to gauge whether performance differences are rooted in the scenario, in congestion control choice, or in the application-layer implementation and settings (e.g., dual-stack connection management policies, fine tuning of timers, maximum number of simultaneous connections, etc.)

Fig. 41 reports the cumulative distribution function (CDF) of the completion time, only considering the heterogeneous case for the sake of illustration. We denote a set of peers in an experiment with the notation X:Y, with $X \in$ {BT, UT} and $Y \in$ {uTP, NewReno, Vegas, Nice, LP}. Additionally, we employ the notation X:$\star$ to indicate any congestion control algorithm used with application X. In the scenario of 41, half of the peers are using Cubic and half of them are using a single other congestion control algorithm among uTP, NewReno, Vegas, Nice and LP. For the sake of clarity, we show a CDF curve for each half of the peers using the same congestion control algorithm. In order to reduce the set of CDF curves and because the performance of the BT:Cubic peers is different from the other sets of peers, we show an envelope (with light gray shaded color) instead of independent curves. For instance, BT:Vegas is for half of peers using Vegas only for an heterogeneous scenario, and the other half using Cubic falls into the envelope BT:Cubic.

We observe in Fig. 41 that the completion time for BT:Cubic is greater than with any other combination of application and congestion control algorithms, and that the completion time for UT:uTP and UT:Cubic peers are indistinguishable[4], so we represent it with a single curve UT:$\star$. We also observe that the shortest completion time is for BT:Nice (unlike in the homogeneous scenario, not shown in this Deliverable).

In summary, results confirm that application implementation may have an important impact (i.e., UT connection management allowing uTP and Cubic traffic to mix), but also show that the congestion control algorithm has a significant impact. Contrasting our findings with related work, the difference of performance between BT:Nice and BT:Cubic is similar to the one noticed in [77], but our results on UT are different because the simplistic simulations settings of [77] do not take into account the complex connection management policy of UT.

3.3.4.2 Impact of BitTorrent TCP flavor on queuing delay of competing applications

We now turn our attention to the side effect that TCP flavors used by BitTorrent can have on competing applications: in this case, we report the queuing delay due to BitTorrent TCP (and BitTorrent UDP) traffic as relevant QoS metric that can be mapped to QoE of other applications following Sec. 3.3. Notice, that, additionally, BitTorrent-specific heatmaps can be extrapolated by correlating the observed queuing delay to the torrent completion time, as we will report later in this paragraph.

We use for the buffer occupancy (left plots) an x-axis in KBytes and the corresponding buffering time in seconds for new packets entering the queue. Again, for the sake of brevity Fig. 42 only reports the heterogeneous case, showing the CDF of the buffer occupancy.

We observe that the delay-based algorithms (UT:uTP, BT:Nice, BT:Vegas) leads to the shortest buffer occupancy, followed by the loss-based algorithms (BT:LP and BT:NewReno), and by BT:Cubic as

[4] This results from UT connection management, that allows on the one hand UT:Cubic peers to employ reverse uTP traffic, but also forces UT:uTP peers to send TCP Cubic traffic to UT:Cubic peers with whom they have not successfully established a uTP connection yet.

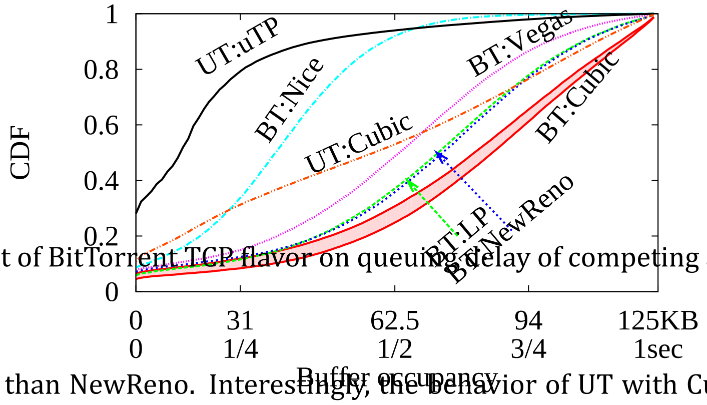


Figure 42: QoS. Impact of BitTorrent TCP flavor on queuing delay of competing applications

Cubic is more aggressive than NewReno. Interestingly, the behavior of UT with Cubic (UT:Cubic) is very different from the one of BT with Cubic (BT:Cubic), confirming the complex interactions between the application and the congestion control algorithm. Interestingly also, the BT:Nice buffer occupancy is very close to that of UT:uTP, and a few percent of peers have a higher buffer occupancy with UT:uTP than with BT:Nice (because the CDF crosses).

We also notice that, in 20% of the cases, queuing delay exceeds 250 ms even under the UT:uTP best case (that has the shortest queuing delay). It follows that, due to the presence of legacy peers employing TCP-only connections, interactive applications can still be negatively affected by BitTorrent traffic, even if the peer support LEDBAT. This happens because, as previosuly observed in Sec. 3.3.2.1, a single TCP connection can still generate bufferbloat, which means that congestion control is only a partial solution in an heterogeneous scenario, and would benefit of a more massive penetration of delay-based congestion control for any kind of bulk data transfer beyond BitTorrent (e.g., to the Cloud, Dropbox, etc.).

We finally extrapolate an heatmap equivalent to correlate BitTorrent QoES with the QoS delay measurement. This would allow, given delay measurement, to forecast the QoE performance of BitTorrent, as was done for other applications in Sec. 3.3, further completing the mapping between QoS and QoE. We therefore assess if (and how much) QoS and QoE are correlated in the BitTorrent case. Clearly, in case Actice Queue Management (AQM) techniques are in place, the picture becomes more complex, since as we show in [36, 35], a reprioritization phenomenon happens that change the congestion control performance. As such, in what follows we merely report the FIFO buffer management policy case.

We present in Tab. 6 a correlation based analysis, in order to compactly summarize our findings and assess whether differences in the buffer statistics explain the completion time difference. We compute the median and 90-th percentile of the T_i and B_i metrics, and we evaluate the Spearman's correlation coefficient as $\rho(X(T_i), Y(B_i))$. As an example, to compute the Spearman's correlation between the 90-th percentile of the queuing delay and the completion time for BT, we define a set of $(50th(T_i), 90 - th(B_i))$ pairs, where statistics are computed for any given swarm i ∈BT:⋆. The correlation among the pairs is then computed over the set of swarms.

Intuitively, the Spearman's correlation quantifies whether an order exists between two different metrics. In our case, it will show whether the order observed for the completion time is the same as for the buffer occupancy. In more details, while Pearson's correlation coefficient is directly evaluated over two metrics, and expresses the existence of a *linear* relationship between them, the Spear-

Table 6: QoS vs QoE: Assessing the relevance of a BitTorrent QoS to QoE heatmap via Spearman's Rank Correlation

		Buffer B			
	Time T	50-th	90-th	50-th	90-th
Homo	50-th	0.54	0.32	0.10	0.10
	90-th	0.36	0.14	-0.30	-0.30
Hetero	50-th	0.77	0.56	0.62	0.67
	90-th	0.76	0.58	0.60	0.69
		Multi applications { UT:⋆ ∪ BT:⋆ }		**Mono application** { BT:⋆ }	

man's correlation coefficient is instead evaluated over their *rank* and expres ses the existence of a monotonous (but not necessarily linear) relationship between the metrics. In a sense, this analysis allows to support (or confute) the relevance of a BitTorrent QoS to QoE heatmap similar to those shown in Sec. 3.3.

For the homogeneous scenario, we see that in the BT:⋆ case, the buffer occupancy and completion time are not correlated. This is coherent with findings in [66], whereby an additional delay equal for all peers have only minimal impact on system performance. Rather, completion time of slowest peers (90-th) is negatively correlated with delay. We only observe correlation when we consider both BT:⋆ and UT:⋆, hinting for an important impact of application settings: hence, not only congestion control, but application-specific tuning can have an important impact. This suggest that BitTorrent QoS to QoE heatmaps could thus be gathered implementation-wise, so to be able to capture instrisic differences at client level. The specific BitTorrent client is easy to get via passive methods as it is encoded in BitTorrent handshake messages (matter of fact, we have implemented this feature in Tstat), so that this information could be available at the reasoner.

At the same time, Internet is an intrinsecally heterogeneous sceario, so that heterogeneous performance shown earlier are a better fit for mPlane purposes. In this case, we see that In the heterogeneous scenario instead, buffer occupancy and completion time are highly correlated, for both BT and UT (and even considering BT:⋆ alone). This extends the validity of previous simulation findings [77] to the real world, additionally showing that congestion control algorithms, neglected by previous studies, play an important role as well. In terms of BitTorrent QoS to QoE heatmaps, this implies that QoS metric should have a more important impact with respect to low-level implementation details, so that a single BitTorrent heatmap should be able to provide reasonable estimates of BitTorrent QoE from QoS measurements.

3.4 Hypergraph Mining

Graph-based modeling provides a foundation for phenomena and/or problems involving one-to-one relationships/interactions among entities allowing data analysis and mining to understand relations between these entities. However, graph modeling fails to capture group-level interactions between entities that are of different nature. Indeed, many of the relationships exhibited in various domains including information networks are not restricted to be one-to-one. Building a model that inherently handles many-to-many relationships/ group interactions would better rely on hypergraphs. In a graph an edge can be incident on exactly two vertices whereas each hyperedge in a hypergraph is an arbitrary subset of the vertex set and represents relations between its elements.

Thus, many hyperedges may be subsets of other hyperedges. Hypergraphs can model many-to-many relationships among entities attributes enabling in turn to handling problems such as similarity, clustering and construction of classifiers. Moreover, a probabilistic hypergraph presents not only grouping information, but also the probability that a vertex belongs to a hyperedge. In this way, the correlation information among vertices can be more accurately described. This section details the foundational principles and techniques underlying hypergraph data mining. The technique is applied the analysis of the relationships between (attributes of) traffic directed to certain Autonomous Systems (AS)/address prefixes (vertex) and content caches/servers (hyperedge). The technique is then extended to probabilistic hypergraphs, which represent the probability that vertices belong to hyperedges and hierarchical directed acyclic graphs which represent relationships between hyperedges.

3.4.1 Introduction

Graph-based modeling provides a foundation for explaining phenomena and/or investigating problems involving one-to-one relationships/interactions (represented by edges) among entities (represented by vertices) allowing data analysis and mining to understand relations between these entities.

Such modeling relies also on the distinction made between undirected and directed graphs but also between unweighted and weighted graphs.

- Unweighted graph: an unweighted graph G is defined by the tuple (V, E) where V set of vertices ($|V| = n$) and E is the set of edges (or arcs), ($|E| = m$). The elements e (or indexed $e_j, j = 1, cdots, m$) of the set E are pairs (u, v) where $u, v \in V$. An edge (v, v) is a self-loop.
- Weighted graph: a weighted graph G is defined by the tuple (V, E, w) where V set of vertices ($|V| = n$) and E is the set of edges (or arcs), ($|E| = m$) and w is a function which associated to each edge $e = (u, v)$ a weight. Weight can represent cost associated to the edge, the strength of the edge or a probability of interconnection (at the condition that the function $w : V \times V \rightarrow [0, 1] \in Re : e_j \rightarrow w_j \in [0, 1]$.
- Undirected graph: the edge pairs are unordered; the set E defines symmetric relation: $(u, v) \in E$ implies $(v, u) \in E$, in other terms the edge (u, v) and the edge (v, u) correspond to the same edge.
- Directed graph (or digraph): the edge pairs are ordered (the edges have an associated direction). A directed graph having no multiple edges or loops is called a simple directed graph. A directed graph having no symmetric pair of directed edges (i.e., no bidirected edges) is called an oriented graph. An acyclic digraph is a directed graph containing no directed cycles it is also referred to as a directed acyclic graph (DAG).

In the space of communication networks, "dyadic" deterministic graphs have been widely used to represent (or model) many "structures" from physical to routing topologies. Examples includes:

- Undirected unweighted graphs $G = (V, E)$ can model the inter-domain or AS-level network topology where finite vertex set V represents abstract nodes, e.g., the autonomous systems (AS), and the finite edge set E represents the interconnection between AS pairs $(u, v), u, v \in V$.

- Undirected weighted graph $G = (V, E, w)$ can model the router-level network topology where finite vertex set V represents routers or inter-connection points, and the finite edge set E represents nodes interconnection. This is particularly the case when the (intra-domain) routing topology is derived from link-state routing protocol exchanges

Using these models, more elaborated networking entities can be represented. Examples include:

- Topological path $p(u, v)$ from vertex u to v is defined as the finite vertex sequence $[x_0(= u), x_1, \ldots, x_{i-1}, x_i, \ldots, x_p(= v)]$ such that the vertex x_{i-1} is adjacent to x_i, $\forall(x_{i-1}, x_i)(i = 1, \ldots, p) \in E$. The length $\ell(u, v)$ of the path $p(u, v)$ is defined as the finite number of edges that the path $p(u, v)$ traverses from vertex u to v. Given a distance metric d, the distance $d(u, v)$ between two vertices $u, v \in V$ denotes the length of a shortest path $p(u, v)$ from vertex u to v.

- Routing path $r(u, v)$ from vertex u to v is defined by the loop-free topological path$[x_0(= u), x_1, \ldots, x_{i-1}, x_i, \ldots, x_p(= v)]$ such that the vertex x_{i-1} is adjacent to x_i, $\forall(x_{i-1}, x_i)(i = 1, \ldots, p) \in E$ as computed by the routing algorithm. In other terms, depending on the routing algorithm, topological and routing path may be different.

- Multicast distribution trees modeled as directed acyclic graphs or point-to-multipoint routing paths.

Distinction between topological path and routing path (output of the routing algorithm) leads to a routing topology which is a sub-graph G' of the graph G representing the network topology. These definitions lead also to the characterization of fundamental network topology properties such as the diameter $\Delta(G)$ of the graph G defined as the maximum distance between any two vertices $u, v \in V$, i.e., $\Delta(G) = max_{u,v}\{d(u, v)|u, v \in V\}$, the average path length defined as the average of the shortest paths length over all pairs of vertices, and the characteristic path length.

Despite their wide applicability for network modeling, dyadic graph modeling fails to capture group-level interactions / relationships between entities that are often of different nature. Indeed, many of the relationships exhibited in various domains including communication and information-centric networks are not restricted to be one-to-one.

In particular, these networks show i) multi-layer structures (an upper layer can make use of several lower layers and the latter can accommodate multiple upper layers or simply take the case of nodes interconnected by a multi-access broadcast segment), ii) multi-level/hierarchical structures (AS routing path over link-state routing paths provides an interesting example of complex relationships between routing levels or the set of sub-groups composing all network communities) and iii) (hidden) relationships between entities including similarity and affinity between vertex (and subsets of vertices).

3.4.2 Hypergraphs

The above observation leads to the introduction of a more general modeling technique where relationships would not be limited to be one-to-one but where such relationships would appear as a particular case.

3.4.2.1 Definition

A hypergraph is defined as a tuple $H = (V, E)$ where $V = v_1, v_2, \ldots, v_n$ is the vertex set, $|V| = n$ and $E = e_1, e_2, \ldots, e_m$ is the set of non-empty subsets of V, such that $\cup_j e_j = V$, called hyperedges (or hyperarcs). Each hyper-edge e_j is an arbitrary sub-set of the vertex set V. Thus, many hyperedges may be subsets of other hyperedges which represent the relations between two or more vertices. The degree of the vertex v is the sum of the corresponding row of the incidence matrix of H. This is equal to the number of hyper-edges that the vertex belongs to.

In the present section, and similarly to weighted dyadic graphs, we also consider weighted hypergraphs $H = (V, E, w)$ where each hyperedge e_j is assigned a positive weight $w(e_j)$.

3.4.2.2 Representations

Two nodes are adjacent in $H = (V, E)$ if there is a hyper-edge e_i that contains both of these nodes.

A hypergraph $H = (V, E)$ can be represented by an incidence $n \times m$ finite matrix $I(H) = V \times E$ such that $i(v_i, e_j) \in [0, 1]$ in which each of the N rows is associated with a vertex and each of the m columns is associated with a hyper-edge, where $i(v_i, e_j) = 1$ if $v_i \in e_j$ and $i(v_i, e_j) = 0$ if $v_i \notin e_j$.

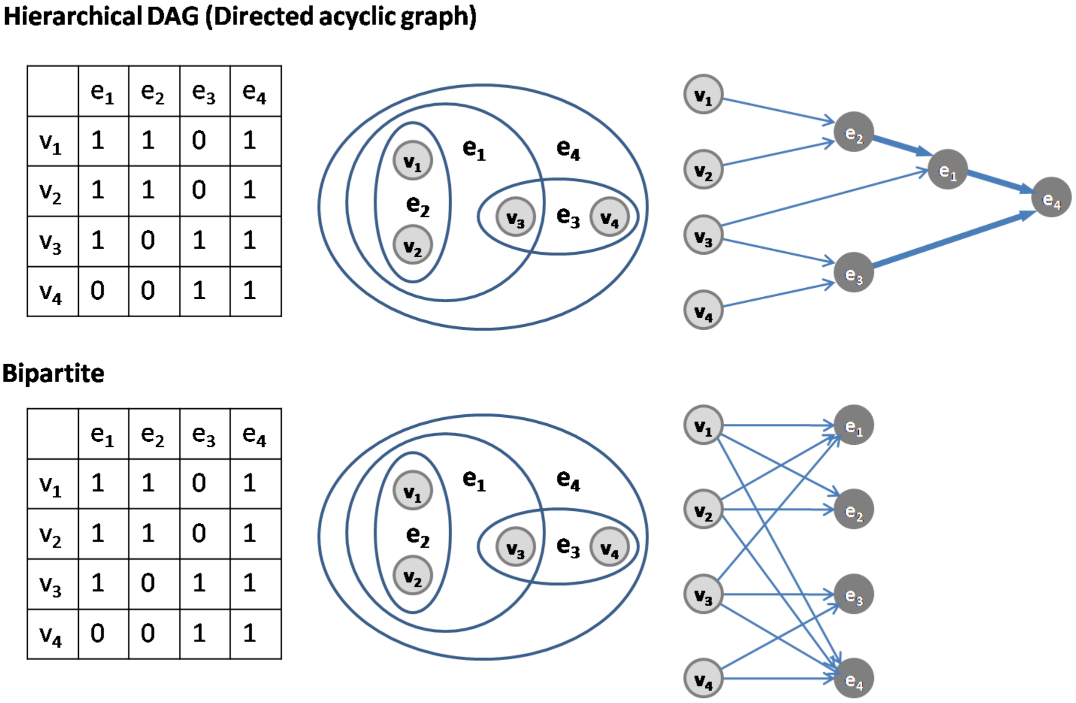
This representation is useful for numerical processing but lacks graphical model representation (visualization). For this purpose, Ensemble representation (which directly follows from the hypergraph definition provided here above), Hierarchical Directed Acyclic Graph (HDAG) and Bipartite Graphs are often used to represent hypergraphs. A bipartite graph, also called a bigraph, is defined by the tuple $G = (U, V, E)$ where set of graph vertices is decomposed into two disjoint sets U and V (U and V are each independent sets) such that no two graph vertices within the same set are adjacent and every edge $e \in E$ connects a vertex in U to one in V. In the bipartite representation, the vertex on one (left) side of the bipartition represent vertices in the hypergraph, while vertices on the other (right) side represent hyperedges. The edges in the bipartite graph represent the membership of vertices in hyperedges.

The following example provides the hypergraph represented by its incidence matrix $I(H) = |V|x|E|$, together with its representation by Ensemble, HDAG and Bipartite graph.

3.4.2.3 Applicability

Building a model that inherently handles many-to-many relationships/group interactions,leads to consider hypergraphs. Indeed, in a dyadic graph an edge can be incident on exactly two vertices whereas each hyperedge in a hypergraph is an arbitrary subset of the vertex set and represents relations between its elements. As many hyperedges may be subsets of other hyperedges, hypergraphs can model many-to-many inter-twinned relationships among entities enabling in turn to handling problems such as vertex similarity, clustering but also construction of classifiers (classification rules). More precisely, clustering proceeds by identifying groups of attributes with similar characteristics, which may be relationships among attributes based on observation patterns or identified using association rules. Determining classification rules involve learning the value of one attribute from the values of other attributes.

Hypergraphs has been proposed as a method of choice in the context of shared risks detection and identification. Let denote by C the set of components of the system, $C = c_1, \cdots, c_p$ such that $|C| =$



Hierarchical DAG (Directed acyclic graph)

	e1	e2	e3	e4
v1	1	1	0	1
v2	1	1	0	1
v3	1	0	1	1
v4	0	0	1	1

Bipartite

	e1	e2	e3	e4
v1	1	1	0	1
v2	1	1	0	1
v3	1	0	1	1
v4	0	0	1	1

Figure 43: Hypergraph Representation

p and S the set of shared risk groups, $S = s_1, \cdots, s_q$ such that $|S| = q$. We say that a component $c_j \in C$ belongs to the shared risk group s_i if this component c_j includes resources/supplies covered by s_i. The shared risk model is characterized by the following properties. Any component $c_i \in C$ belongs at least to one SRG, i.e., $|S| = q \geq p$. By extension, the component $c_i \in C$ belongs to the SRG set $S' = s_1, \ldots, s_{q'}$ with $q' \leq q$ if the component c_i crosses at least one of the resources of each of its members $s_1, \ldots, s_{q'}$. Any pair of components $c_i, c_j \in C$ belonging to the SRG s_k ($c_i, c_j \in s_k$) can individually belong to a set of other SRGs, i.e., $c_i \in s_p$, $c_j \in s_q$ such that $s_k \cap s_p = c_i$ and $s_k \cap s_q = c_j$. More generally, any component from a given subset of components taken individually may belong to a set of other SRGs

The nodal version of this model, where components are defined as modules of network programs, has a direct applicability in the context of programmable networks where software modules share common risks. The tasks consists in determining which nodes executing some module(s) share common risk of joint failures; for instance, when certain properties of the execution environment (node programs) may cause joint failure of these modules.

Observe that this situation differs from current networks operation (where operators configure pre-loaded software components) when distinction is made between node and network programming. In network function programming (the so-called "network as compiler" paradigm), the operator designs programs that are automatically verified, compiled and deployed at running time. This step leads to a significant change compared to current networks operations (since enabling the introduction and the replacement of new software-defined functions without requiring hardware change) but also to their control processes. Consequently, an error at the network program level may translate into multiple node program errors. In turn, ensuring network-level fault-tolerance requires extending the vertex-disjoint path problem (unformally, two nodes remain connected even if one node along one of the paths connecting these two nodes fails) with a detailed knowledge

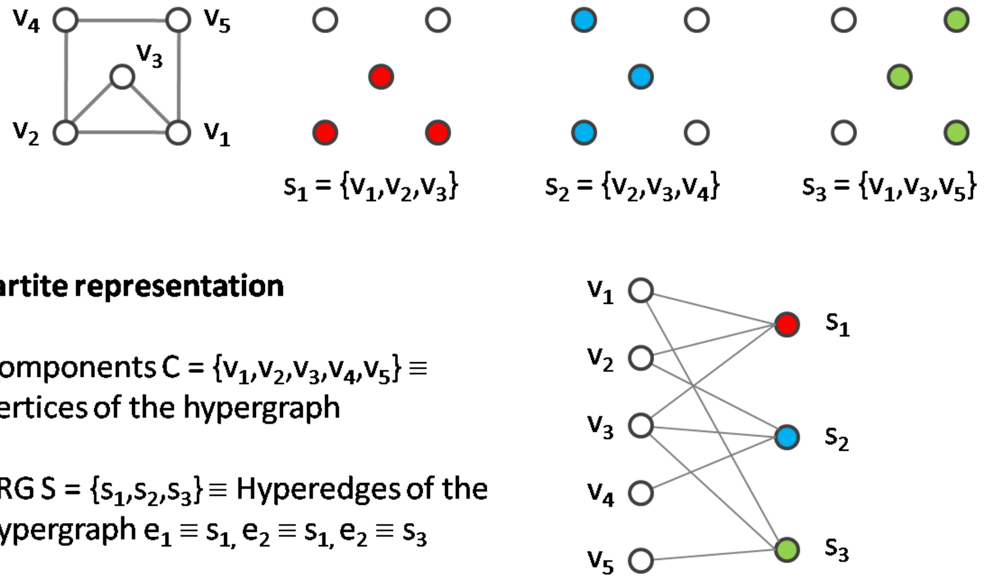

v4
v5
v3
v2
v1
s1 = {v1,v2,v3}
s2 = {v2,v3,v4}
s3 = {v1,v3,v5}
Bipartite representation
• Components C = {v1,v2,v3,v4,v5} ≡ vertices of the hypergraph
• SRG S = {s1,s2,s3} ≡ Hyperedges of the hypergraph e1 ≡ s1, e2 ≡ s1, e2 ≡ s3
v1
v2
v3
v4
v5
s1
s2
s3

Figure 44: Shared Risk Model - Nodal version

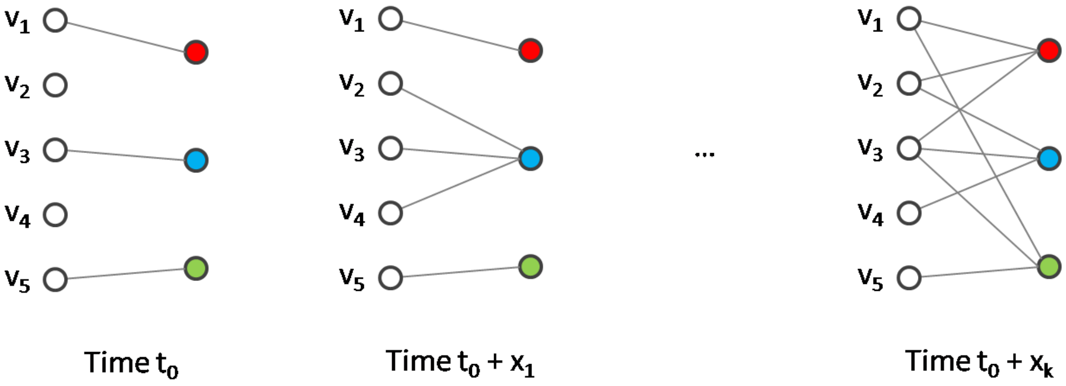


Figure 45: Shared Risk Model - Iterative Construction of the Hypergraph

about the software components executed along each of the path.

The iterative construction of the hypergraphs H can be performed by means of the bipartite graph $H = (U, V, E)$ where the vertex set U represents the set of nodes, the vertex set V represents the set of software components and the edge set E represent the inclusion into nodes of software modules. This allows also to consider systems where software modules can be grouped into larger programs together with the representation of their dependencies.

Fig.3 depicts the iterative construction of the hypergraph starting from different observations of software module failures. Starting from time t_0, from each event k occurring at time $t_0 + t_k$, a (set of) edge(s) is added relating nodes to software modules.

3.4.3 Probabilistic Hypergraphs

Conventional hypergraph structure assigns vertex v_i to hyperedge e_j with a binary decision, i.e., $i(v_i, e_j) = 1 or 0$. Consequently, all vertices in a hyperedge are handled equally; relative "similarity", "affinity", etc. between vertices is discarded by such decision (the incidence matrix is equivalent to a Boolean matrix). This leads to loss of some information, which may be harmful to some hypergraph based applications, in particular, when the relationship of assigning a given vertex to an hyperedge is based on a detection process which implies a certain level of uncertainty depending on the detection technique and the application itself. In certain applications the hyperedge set is known and the remaining task consists in assigning probabilities instead of taking a binary decision; in other applications the hyperedge set itself is unknown and the tasks consists in detecting the existence of hyperedges together with their constituency.

The incidence matrix $I(H)$ of a probabilistic hypergraph $H = (V, E)$ is defined as follows: $i(v_i, e_j) = P[i(v_i, e_j)] \in [0, 1], \forall v_i \in V and e_j in E$ such that $\sum_{i|v_i \in e_j} i(v_i, e_j) = 1$. This requires updating the values $i(v_i, e_j)$ as observation arrives. This model can be further extended if one assumes that the probability corresponds to the expectation that hypergraph vertices share a common property (or attribute). In this case, the probability corresponds to the likelihood of a shared attribute among vertices. This model being more elaborated it finds application when the task consists in determining relationships among data set attributes that can take value from a finite based on observation patterns or identified using association rules.

3.4.4 Application

In content-centric networks (a.k.a. information-oriented networks), intermediate nodes are capable to (temporarily) store content objects or sub-objects (when the content can be segmented). Depending on their network location, content servers and/or caches steer the spatio-temporal distribution of traffic leading to key challenges for transit networks. In particular, because these networks do not benefit from any information related to content location from their neighboring domains to fulfill traffic-oriented but also resource-oriented performance objectives (or both).

The key characteristic of such networks is to maintains multiple content objects that are reachable via single IP address (meaning are associated to the same locator) whereas multiple IP addresses (network locators) can host the same content object. Finding these M:N relationships and the spatial distribution between content objects are the main tasks underlying this application. Using this set of relationships, transit networks may specify their traffic engineering policies and tune their traffic engineered path computation so as to better comply to traffic-oriented but also resource-oriented performance objectives (or both).

On the other hand, we consider the situation where the edges of the transit networks connected to neighboring content networks can passively monitor (by means of monitoring points or agents) content requests and replies. The situation is depicted in Fig.4. Note that edges are assumed to share the information extracted out of content requests and replies.

The iterative construction of the hypergraph H by means of the bipartite graph $H = (U, V, E)$ where the vertex set U represents the set of content objects, the vertex set V represents the set of locators and the edge set E represent the probability to find a given content object at a given location. This iterative construction allows to find the M:N relationship between content objects and locators. The result of the execution is depicted in Fig.5. Remember that content requests are

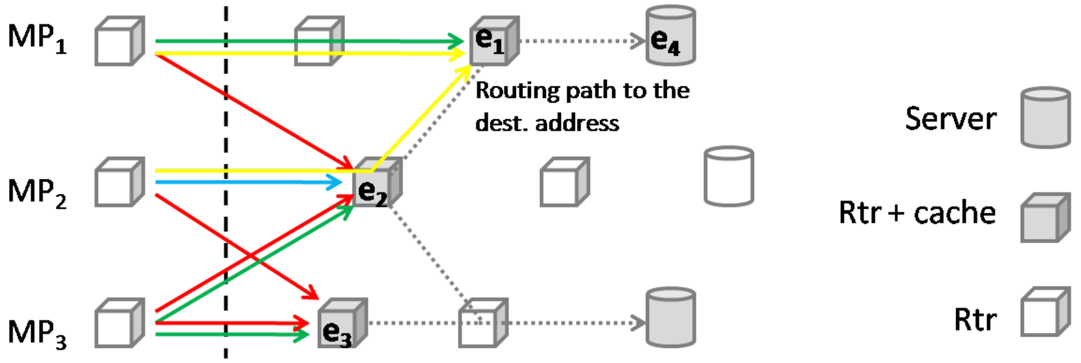


Figure 46: Transit networks and In-Network Caching - Baseline Topology

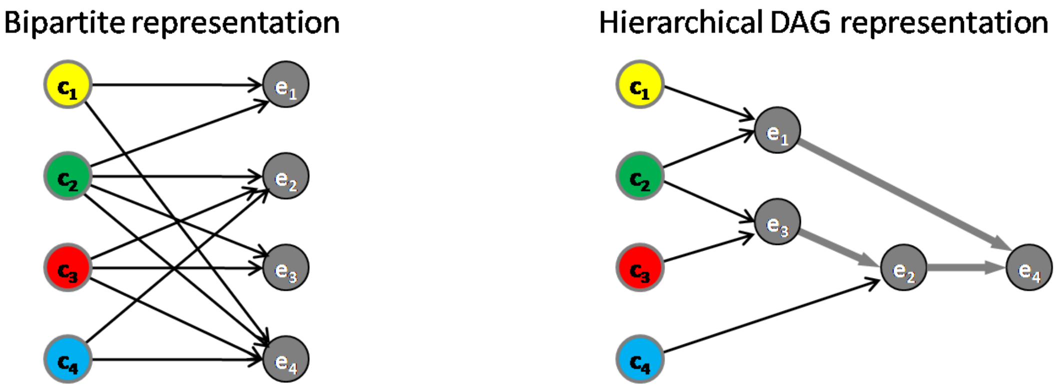


Figure 47: Transit networks and In-Network Caching - Iterative Construction of the Hypergraph

directed to servers but content replies have as source address the sender/transmitter of the content object. In case this condition is not verified (the transmitter puts the server address as source address) an additional procedure is required to spatially disambiguate two different locations using the same locator. Fig.5 also provides the representation in terms of HDAG where spatial relationship between content objects.

The setting of the probability to each edge of the bipartite graph representation of the hypergraph is part of the learning task which may be performed by means of frequentist inference or Bayesian inference. To briefly outline the main difference between these approaches, Bayesian inference estimates the conditional probability $P[hypothesis|data]$. In contrast, frequentist inference estimates the conditional probability $P[data|hypothesis]$. The critical point about Bayesian inference, is that it provides a principled way of combining new events (a.k.a evidences) with prior knowledge (a.k.a. beliefs), through the application of Bayesian rule (whereas the frequentist inference relies only on the evidence as a whole, with no reference to prior beliefs). The proposed procedure to construct hypergraphs fits perfecttly to the Bayesian rule which can be applied iteratively: after observing some events, the resulting posterior probability can be treated as a prior probability, and a new posterior probability computed from new events.

Another approach consists in estimating the class probability between content requests/replies where each class correspond to a certain object type (attributes). This approach requires further investigation in so-called imbalanced scenarios in which the number of instances from each class is (perhaps largely) unequal. In such scenarios, mis-classification costs tend to be asymmetric:

incorrectly classifying rare events is usually more costly than making mistakes in the other direction (which may lead to imperfection when establishing traffic engineering rules).

3.5 Statistical Relational Learning

This section aims at providing elements of answer to the following question on data analysis and mining: which class of learning technique can be considered and/or extended when the input data is obtained online by its individual monitoring agents. Answering this question imposes to first examine the fundamental relationship between the learning technique and the input data properties on which it performs. On the one hand, (most of) the commonly envisaged statistical learning techniques assume that i) propositional data are identically and independently distributed ("i.i.d. assumption"), implying that an element in the sequence is independent of the random variables that came before it and ii) random samples of homogeneous data objects result from single relation. These common assumptions have to be contrasted with the intrinsic properties of real world data sets, in particular, those characterizing communication networks. These environments are characterized by data that are not identically distributed (heterogeneous) and not independent (complex multi-relational structures). On the other hand, most relational learning techniques neither assume noise nor uncertainty in data whilst real world data are often characterized by distributions that show presence of uncertainty and noise.

3.5.1 Introduction

Many distributed learning techniques ranging from reinforcement learning to Bayesian learning have been investigated over last two decades. However, none of them effectively accounts for the intrinsic properties of the data characterizing the environments to which they apply. Filling this gap is the main purpose of Statistical Relational Learning (SRL) [49]. This technique combines relational logic learning to model complex relational structures and inter-dependency properties in data with probabilistic graphical models (such as Bayesian networks or Markov networks) to model the uncertainty on the data. The resulting process can perform robust and accurate learning about complex relational/ inter-dependent data. The goal of SRL is of particular interest in the context of mPlane where the reasoner aims at learning hidden dependencies between multi-relational, heterogeneous and semi-structured but also noisy and uncertain data. The ultimate objective of SRL is indeed to learn from non-independent and identically distributed data "as easily as" from independent and identically distributed data. This learning technique is particularly adapted to information extraction; it provides better predictive accuracy and understanding of domains. However, this technique induces a harder learning task and higher complexity. SRL is nowadays applied to social networks analysis, hypertext and web-graph mining, etc.; it is thus reasonable to also consider its potentials in the context of the mPlane reasoner since i) the models learned from both intrinsic (propositional) and relational information perform better than those learned from intrinsic information alone, ii) probabilistic relational models offer significant advantages over deterministic relational models (including better predictive accuracy and better understanding of relational structure in heterogeneous data set), and iii) SRL can learn accurate models of (inter-)dependent data instances.

3.5.2 Statistical Relational Learning

SRL combines probabilistic graphical models (probabilistic learning and inference) to model and reason about uncertainty with representation language to describe relational properties of the data and complex dependencies between them (logical learning and inference).

Graphical models provide a principled approach to deal with uncertainty and relational data by means of the probability theory. These models represent dependency structure between random variables by joint distributions. Two types of graphical models are commonly considered: Markov(ian) networks and Bayesian networks. On the one hand, Markov networks are described by undirected graphs where edges do not carry arrows (no acyclic constraint) and have no directional significance are useful for expressing symmetric relationships (soft constraints) between random variables. On the other hand, Bayesian networks are represented by Directed Acyclic Graphs (DAG) where edges have a particular directionality indicated by the arrows (acyclic constraint) are useful for expressing causal relationships between random variables.

In addition to the distinction between undirected and directed graphical models, the differentiation between main representation syntaxes, i.e., first-order logic vs. frame-based representation provides a complete categorization of the different classes of SRL models. One distinguishes as part of the directed models between rule-based models Bayesian Logic Programs (BLP) [45] and frame-based models Probabilistic Relational Models (PRM) [31] and as part of undirected models between frame-based models Relational Markov Networks (RMN) [65] and rule-based models Markov Logic Networks (MLN) [60]. In the present section, we extend the latter, i.e., the MLN model. Selection of this learning model stems from the following reasons: it suits control processes whose execution is causality-independent, it is more flexible when the data are made available sequentially (as it is the case in communication networks) and it enables exploiting data sparseness by grounding "lazily".

3.5.2.1 Incremental Markov Logic Networks (iMLN)

The fundamental issue stems from the following: these models have been designed independently on the input data arrival process, i.e., the processing algorithm perform on complete data set (in "batch mode"). However, when performing online learning, data arrive following different temporal patterns (in "sequential mode") and the model is to be updated as data arrive. As stated in the previous section, this is also one of the main reasons for selecting the Markov Logic Network model as it supports for sequential data.

For this purpose, we extend the Markov Logic Network (MLN) model which represents a probability distribution over possible worlds to cover incremental updates from arrival of input data. A MLN is formally defined as a set of pairs of formulas F_i in first order logic and their corresponding weights w_i denoted (F_i, w_i). It is important to emphasize that a MLN becomes a Markov network only with respect to a specific grounding and interpretation. Indeed, atomic formulas (vertices of the Markov Network) do not have a truth value unless they are grounded and given an interpretation. Thus, one requires that each vertex represents a ground atom (i.e., atomic formula whose argument terms are ground terms). Together with a set of constants in the domain of discourse, a MLN defines a (ground) Markov network with i) one binary vertex for each grounding of each predicate in the MLN (the value associated to the vertex is 1 if the ground atom is true and 0, otherwise) and ii) one potential function f_i for each grounding of each formula F_i in the MLN with the corresponding weight w_i. Each state of the resulting ground Markov network (with a log-linear representation of the potential function f_i) presents a possible world x (i.e., assignment of truth values to all possible vertices or ground atoms) whose probability $P(X = x)$ is given by:

$$P(X = x) = \frac{1}{Z} exp(\sum_i w_i n_i(x)) \tag{3.5}$$

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

Assuming we have at our disposition a given set of formulas $F_1, F_2, F_3, ..., F_n$, the learning task consists in finding the respective weights $w_1, w_2, ..., w_n$ by applying the following steps:

1. Obtaining the weights w_i from the pseudo-likelihood (PL) approximation of the joint probability distribution of a world x based on its Markov blanket. The Markov blanket of a vertex is the minimal set of vertices that must be observed to make this vertex independent of all other vertices. In a directed model, the Markov blanket includes parents, children and co-parents. The PL approximation of $X = x$ is given by:

   $$PL(X = x) = \prod_i P(x_i | neighbors(x_i)) \tag{3.6}$$

   The use of the pseudo-likelihood approximation does not require performing inference at each step and avoids the use of the partition function Z. It is indeed impractical to perform exact inference on large Markov models because of the computations on the partition function Z.

2. Inferring the most likely state of the world y given the evidence x. For this purpose, given the evidences x, it suffices to compute the following:

   $$\arg\max_y \sum_i w_i n_i(x, y) \tag{3.7}$$

   Computation of equation (3) makes use of the MC-SAT algorithm [41]. This algorithm combines Gibbs sampling, which proceeds by sampling each variable in turn given its Markov blanket, with a weighted SAT solver such as the MaxWalkSAT solver [38]. The latter defines a local search algorithm for the weighted satisfiability problem, i.e., find a truth assignment that maximizes the sum of weights of satisfied clauses.

Next, in order to predict the occurrence of certain patterns or events (predictive inference problem), it suffices to compute, using the evidences x, the equation (3) by means of the MC-SAT algorithm. Using this procedure, the corresponding MLN model is able to find for instance the probability that the formula F_i holds knowing that the formulas F_j and F_k do.

3.5.3 Application

SRL is particularly well suited for information extraction, as it provides better predictive accuracy and understanding of domains; however, this technique induces a harder learning task and higher complexity. The gain has to be significant in order to justify this harder learning tasks. Several applications are under investigation including fault diagnosis/root cause analysis and anomaly detection (e.g., detect anomalies/invalidity in control traffic, routing information patterns).

As stated earlier in this section, SRL enables to model the relations between topological and spatio-temporal properties together with collective inference compared to the approaches that process these properties independently. We aim at showing whether this learning technique would be able to determine among the large possible set of (sometimes hidden) relationships between heterogeneous data which of them are susceptible to cause serious operational disruption to relieve from the problem of "operating exclusively in the dark", more precisely, prevent that certain disruptions last longer than they should if additional information would be made available.

Concerning anomaly detection in control traffic the following problems can be considered as providing interesting cases to determine applicability of SRL:

- the detection of hidden relationships between policy rules that cannot be detected by local inspection of routing policies (e.g., for the detection of valid routing information inducing unintended unstable states in path-vector routing) or the detection of hidden relationships between routing paths that cannot be detected by local only inspection of the local routing information base; the learning task consists in performing collective classification task, where the class label of the links may be unknown. This application of SRL extends over relational classification in presence of autocorrelation (note: in relational classification, the task consists in predicting the class label of an object given its attributes).

- the detection of the security risk associated with the selection, e.g., of a given prefix originator; the learning task combines grouping objects that have similar characteristics based on their own attributes and the attributes of their links (link-based clustering) and determining whether a relation exists between two objects from the attributes of the objects and their link (a.k.a. link prediction).

4 Conclusions

This deliverable describes algorithms to 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 addition, we have proposed a series of more generic algorithms that may be useful for the use-cases currently considered and also in a larger context.

Overall the proposed analysis algorithms cover a large range of techniques that enable to

- understand whether a path has enough resources to sustain the rendering of the specific application
- estimate the future popularity trends of services and contents for network optimization
- sort and present only interesting web content to end-users
- assess and diagnose 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
- predict performance of unmeasured end-to-end paths
- discover network topology
- relate variations of one-way delays to performance of applications
- analyse the relationships between traffic directed to certain destinations and content caches/servers
- detect anomalies/invalidity in control traffic, routing information patterns

These algorithms can be grouped into the following categories:

- 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).

References

[1] http://www-sop.inria.fr/members/Arnaud.Legout/Projects/p2p_cd.html.

[2] Internet AS-level Topology Archive at. http://irl.cs.ucla.edu/topology/. [acc. 05-08-2013].

[3] *Revealing Middlebox Interference with Tracebox*, 10/2013 2013.

[4] adblock. EasyList, Adblock Plus. http://easylist.adblockplus.org/. [Online; July 2013].

[5] M. Ahmed, S. Spagna, F. Huici, and S. Niccolini. A peek into the future: Predicting the evolution of popularity in user generated content. In *ACM WSDM*, Feb. 2013.

[6] M. Allman, V. Paxson, and E. Blanton. RFC5681: TCP Congestion Control, 2009.

[7] A. Araldo and D. Rossi. Bufferbloat: passive inference and root cause analysis. Technical report, Telecom ParisTech, 2013. keyword=traffic.

[8] A. Araldo and D. Rossi. Dissecting bufferbloat: Measurement and per-application breakdown of queueing delay. In *ACM CoNEXT, Student Workshop*, Santa Barbara, CA, US, december 2013. keyword=traffic.

[9] 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/USENIX Internet Measurement Conference (IMC)*, October 2006.

[10] B. Augustin, R. Teixeira, and T. Friedman. Measuring load-balanced paths in the Internet. In *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, November 2007.

[11] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *ACM SIGMETRICS*, Madison, US-WI, 1998.

[12] W. Bellante, R. Vilardi, and D. Rossi. On netflix catalog dynamics and caching performance. In *IEEE CAMAD*, Berlin, Germany, september 2013. keyword=traffic.

[13] I. Bermudez, M. Mellia, M. Munafò, R. Keralapura, and A. Nucci. DNS to the Rescue: Discerning Content and Services in a Tangled Web. In *Proceedings of the 2011 ACM SIGCOMM on Internet Measurement Conference*, IMC '12, New York, NY, USA, November 2012. ACM.

[14] P. Biondi. Scapy. See http://www.secdev.org/projects/scapy/.

[15] Z. S. Bischof, J. S. Otto, and F. E. Bustamante. Up, down and around the stack: ISP characterization from network intensive applications. In *ACM SIGCOMM Workshop on Measurements Up the STack (W-MUST'12)*, 2012.

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

[17] P. Casoria, D. Rossi, J. A. Marc-Olivier, B. Timur, Friedman, and A. Pescape. Distributed active measurement of internet queuing delays. Technical report, Telecom ParisTech, 2013. keyword=traffic.

[18] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying skype user satisfaction. 36(4):399--410, 2006.

[19] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An algebraic approach to practical and scalable overlay network monitoring. *SIGCOMM Comput. Commun. Rev.*, 34(4):55--66, Aug. 2004.

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

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

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

[23] H.-K. Choi and J. O. Limb. A behavioral model of web traffic. In *IEEE ICNP*, Toronto, CA, 1999.

[24] D. B. Chua, E. D. Kolaczyk, and M. Crovella. Network kriging. *IEEE Journal of Selected Areas in Communications*, 24(12):2263--2272, Dec. 2006.

[25] A. Coyle, M. Kraetzl, O. Maennel, and M. Roughan. On the predictive power of shortest-path weight inference. In *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, November 2008.

[26] H. Cui and E. Biersack. Distributed troubleshooting of web sessions using clustering. In A. Pescapè, L. Salgarelli, and X. A. Dimitropoulos, editors, *TMA*, volume 7189 of *Lecture Notes in Computer Science*, pages 125--128. Springer, 2012.

[27] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of ACM SIGCOMM*, Portland, OR, USA, Aug. 2004.

[28] A. D'Alconzo, A. Coluccia, and F. Ricciato. A Distribution-Based Approach to Anomaly Detection for 3G Mobile Networks. In *IEEE Globecom '09*, Nov. 2009.

[29] A. D'Alconzo, A. Coluccia, and P. Romirer-Maierhofer. Distribution-based anomaly detection in 3g mobile networks: from theory to practice. *International Journal of Network Management*, 20(5):245--269, 2010.

[30] digg. Digg. `http://www.digg.com`. [Online; July 2013].

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

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

[33] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP extensions for multipath operation with multiple addresses. RFC 6824, Internet Engineering Task Force, January 2013.

[34] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Communications of the ACM*, 55(1):57--65, 2012.

[35] Y. Gong, D. Rossi, and E. Leonardi. Modeling the interdependency of low-priority congestion control and active queue management. In *The 25th International Teletraffic Congress (ITC25)*, september 2013. keyword=ledbat.

[36] Y. Gong, D. Rossi, C. Testa, S. Valenti, and D. Taht. Fighting the bufferbloat: on the coexistence of aqm and low priority congestion control. In *IEEE INFOCOM Workshop on Traffic Measurement and Analysis (TMA'13)*, Turin, Italy, April 14-19 2013. keyword=ledbat,bufferbloat,mplane.

[37] 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.

[38] Y. H.Kautz, B.Selman. A general stochastic approach to solving problems with hard and soft constraints. pages 573--⍰586. American Mathematical Society (AMS), 1997.

[39] O. Holfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford. BufferBloat: how relevant? a QoE perspective on buffer sizing. Technical report, 2012.

[40] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP. In *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, November 2011.

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

[42] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. LUA, an extensible extension language. *Software: Pactice & Experience*, 26(6):635--652, June 1996.

[43] S. Ihm and V. S. Pai. Towards understanding modern web traffic. ACM IMC, Berlin, DE, 2011.

[44] K. Keys. Internet-scale IP alias resolution techniques. *ACM SIGCOMM Computer Communication Review*, 40(1):50--55, January 2010.

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

[46] 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.

[47] A. Kuzmanovic and E. Knightly. TCP-LP: low-priority service via end-point congestion control. *IEEE/ACM Transactions on Networking*, 14(4):752, 2006.

[48] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? ACM WWW, Raleigh, US-NC, 2010.

[49] E. L.Getoor, D.Jensen. Proceedings of aaai 2000 workshop on learning statistical models from relational data. AAAI Press, 2000.

[50] Y. Liao, W. Du, P. Geurts, and G. Leduc. Decentralized prediction of end-to-end network performance classes. In *Proc. of CoNEXT*, Tokyo, Japan, Dec. 2011.

[51] M. Luckie. Scamper: a scalable and extensible packet prober for active measurement of the Internet. In *Proc. ACM/USENIX Internet Measurement Conference (IMC)*, November 2010.

[52] 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.

[53] G. Maier, F. Schneider, and A. Feldmann. Nat usage in residential broadband networks. In *PAM*, 2011.

[54] 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, Special Issue on Measurement of Internet Topologies*, 29(6):1776--1787, October 2011.

[55] 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.

[56] Y. Matsubara, Y. Sakurai, B. A. Prakash, L. Li, and C. Faloutsos. Rise and fall patterns of information diffusion: model and implications. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 6--14. ACM, 2012.

[57] 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.

[58] S. Morris. μTorrent release 1.9 alpha 13485. `http://forum.utorrent.com/viewtopic.php?pid=379206#p379206`, Dec 2008.

[59] mPlane consortium. Public Deliverables. `http://www.ict-mplane.eu/public/public-deliverables`.

[60] P. M.Richardson. Markov logic networks. volume 62 of *Machine Learning*, pages 107--□136, February 2006.

[61] D. Neal. Digg is dogged by conservative pressure groups. `http://goo.gl/M9Plt`. [Online; August 2010].

[62] A. Norberg. BitTorrent Enhancement Proposals on μTorrent transport protocol. `http://www.bittorrent.org/beps/bep\_0029.html`, 2009.

[63] opengraph. Facebook opengraph. `https://developers.facebook.com/docs/opengraph/`.

[64] pinterest. Pinterest. `http://www.pinterest.com`. [Online; July 2013].

[65] D. P.Taskar, P. Abbeel. Discriminative probabilistic models for relational data. In *Proceedings of 18th Conference on Uncertainty in Artificial Intelligence (UAI 2002)*, pages 485--494, 2002.

[66] A. Rao, A. Legout, and W. Dabbous. Can realistic bittorrent experiments be performed on clusters? In *IEEE P2P'10*, Delft, Netherlands, Aug 2010.

[67] reddit. Reddit. `http://www.reddit.com`. [Online; July 2013].

[68] D. Rossi, Y. Nicolas, D. Wolff, and A. Finamore. I tube, youtube, p2ptube: assessing isp benefits of peer-assisted caching of youtube content. In *IEEE P2P'XIII*, Trento, Italy, September 2013. keyword=traffic.

[69] I. S. Ha, Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *ACM SIGOPS Operating System Review*, New York, NY, Jul 2008.

[70] H. H. Song, L. Qiu, and Y. Zhang. NetQuest: A flexible framework for large-scale network measurement. In *Proc. of ACM SIGMETRICS*, 2006.

[71] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.

[72] P. Srisuresh and M. Holdrege. IP network address translator (NAT) terminology and considerations. RFC 2663, Internet Engineering Task Force, August 1999.

[73] B. Staehle, A. Binzenhoefer, D. Schlosser, and B. Boder. Quantifying the influence of network conditions on the service quality experienced by a thin client user. *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pages 1--15, Apr 2008.

[74] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: a view from the gateway. In *ACM SIGCOMM*, 2011.

[75] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *IEEE INFOCOM'06*, Barcelona, Spain, Apr 2006.

[76] 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.

[77] C. Testa and D. Rossi. The impact of utp on bittorrent completion time. In *IEEE Peer to Peer (P2P'11)*, Kyoto, Japan, September 2011. keyword=p2p,ledbat.

[78] C. Testa, D. Rossi, A. Rao, and A. Legout. Data plane throughput vs control plane delay: Experimental study of bittorrent performance. In *IEEE P2P'XIII*, Trento, Italy, september 2013. keyword=ledbat,bittorrent.

[79] M. E. Tozal and K. Sarac. Subnet level network topology mapping. In *Proc. IEEE International Performance Computing and Communications Conference (IPCCC)*, November 2011.

[80] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today's content caching: Why it matters and how to model it. *SIGCOMM Comput. Commun. Rev.*, 43(5), 2013.

[81] V. Jacobson et al. traceroute. man page, UNIX, 1989. See source code: `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`.

[82] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *USENIX OSDI'02*, Boston, MA, Dec 2002.

[83] 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.

[84] I. Witten, E. Frank, and M. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann series in data management systems. Elsevier Science & Technology, 2011.

[85] Y. Liao, W. Du, P. Geurts, and G. Leduc. DMFSGD: A decentralized matrix factorization algorithm for network distance prediction. *IEEE/ACM Transactions on Networking, to appear.*

[86] J. Yang and J. Leskovec. Patterns of temporal variation in online media. In *WSDM '11*, 2011.