PAIN: A Passive Web Performance Indicator for ISPs

PAIN: A Passive Web Performance Indicator for ISPs

Martino Trevisan, Idilio Drago, Marco Mellia
Politecnico di Torino
e-mail: {first.last}@polito.it

Abstract

Understanding the quality of web browsing enjoyed by users is key to optimize services and keep users' loyalty. This is crucial for both Content Providers and Internet Service Providers (ISPs). Quality is intrinsically subjective, and the complexity of today's websites challenges its measurement. Objective metrics like `OnLoad` time and `SpeedIndex` are notable attempts to quantify web performance. However, these metrics can only be computed by instrumenting the browser and, thus, are not available to ISPs.

PAIN (PAssive INdicator) is an automatic system to monitor the performance of websites from passive measurements. It is open source and available for download. It leverages only flow-level and DNS measurements which are still possible in the network despite the deployment of HTTPS. With unsupervised learning, PAIN automatically creates a model from the timeline of requests issued by browsers to render web pages, and uses it to measure website performance in real-time.

We compare PAIN to objective metrics based on in-browser instrumentation and find strong correlations between the approaches. PAIN correctly highlights worsening network conditions and provides visibility into websites performance. We let PAIN run on an operational ISP network, and find that it is able to pinpoint performance variations across time and groups of users.

Keywords: Passive Measurements; Web QoE; Machine Learning

1. Introduction

Objective metrics to indicate the Quality of Experience (QoE) are key to understand how users enjoy the web. Such metrics are of prime importance to all actors involved in the service delivery. From Content Providers, which need to monitor users' satisfaction to maintain or increase their user base, to Internet Service Providers (ISPs), which need to be aware of performance offered by the network and factors affecting web browsing experience [26]. The idea that unsatisfied users are more prone to switch providers is widely disseminated. More than that, there are many anecdotal evidences that a small deterioration of quality levels could result in losses of revenues to providers.[1]

Given the importance of QoE, Content Providers have developed a number of objective metrics to estimate users' QoE. On the contrary, there are hardly any objective metrics to estimate users' QoE at ISPs [7, 5, 26], even if ISPs are equally blamed for poor users' experience. Bad performance in the network and, in particular, in the last-mile is historically the first suspect when users' quality degrades. This has motivated major Content Providers to publicize rankings of ISP performance.[2] It is no exaggeration to say that ISPs are evaluated based on the experience of end-users while interacting with third-party services, with video and web browsing being the most important. In addition, ISPs need to measure the impact of possible network configuration changes on performance – e.g., to decide whether the deployment of web caches or new content delivery nodes is advantageous, or to tune configuration parameters of their networks.

Users' QoE is intrinsically subjective, thus hard to be assessed. Ideally, QoE should be estimated by means of metrics such as the Mean Opinion Score (MOS), which is quantified by asking users directly about their opinions on the service. Previous works [5, 7, 11] have proposed objective metrics that have been shown to be correlated with users' MOS, even if a model to predict MOS is still hard to get [8]. These metrics however either are computed at the server-side (i.e., available to Content Providers only) or require ground truth from in-browser instrumentation (i.e., not scalable for the monitoring of a large number of sites at ISPs). Passive solutions that provide visibility into web performance are rare, and generally complicated by the need to analyze payload to reconstruct web pages [26].

We introduce PAIN (PAssive INdicator), a completely unsupervised system to monitor website performance using passive traffic logs. The adoption of encryption (e.g., HTTPS) makes solutions that reconstruct web sessions from payload [7, 5, 26] no longer effective. PAIN instead relies only on L4-level statistics (e.g., Netflow), annotated

[1] https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales

[2] For an example, see `https://ispspeedindex.netflix.com/`

with the original server domain[3] information [6] to compute a synthetic indicator of the website performance.

We validate PAIN in a testbed, in which we browse websites while collecting also classic client-side objective metrics. We show that PAIN is able to spot changes in network conditions, reporting quality degradation when the site performance effectively degrades. PAIN metrics are strongly correlated with objective metrics obtained by means of client instrumentation, which are in turn known to be correlated to users' MOS [12, 15]. Finally, PAIN outperforms alternatives, either by avoiding expensive training or by working with encrypted traffic.

We demonstrate the practical application of PAIN in a case study. We deploy PAIN in an ISP network for one full year. First, we show how PAIN can help the ISP understand its users' experience, e.g., highlighting web browsing performance of users connected with different Internet access capacity. Then, we show how PAIN lets the ISP quantify variations in web browsing performance, e.g., pinpointing sudden performance variations of websites.

PAIN is open-source, and it is released as a module of the NetLytics Big Data platform [29]. It can be fed using Tstat [32], Squid [2] and Bro [25], to extract performance metrics directly from raw log files.

This paper extends our preliminary work [31]. In contrast to the workshop version, we have performed new experiments to better validate parameter choices, added new results comparing PAIN against supervised alternatives, and deployed PAIN in a large operational network in a case study. Finally, the presentation of the algorithms have been extended.

In the following, Section 2 details the problem and envisioned deployment scenario, while Section 3 summarizes related work. Section 4 describes PAIN design and algorithms. Section 5 introduces the employed datasets. Section 6 validates PAIN, while Section 7 describes our experience of running PAIN on an operational network. Finally, Section 8 concludes the paper.

2. The complexity of QoE estimation

2.1. Objective quality metrics

Given the intrinsic subjectiveness of QoE, measuring it is hardly possible without involving the users directly. Therefore, large-scale measurement campaigns are usually infeasible. Not a surprise, several approaches exist to estimate QoE with objective metrics calculated without human intervention.

In this paper we focus on users' experience while browsing the web. Two of the most popular objective metrics to estimate users' QoE in this scenario are:

(i) **OnLoad** time: The time browsers fire the `onLoad` event – i.e., when all elements of the page, including images, style sheets and scripts have been downloaded and processed;

(ii) **SpeedIndex**: Proposed by Google,[4] it represents the delay to render the visible portions of a page. It is computed by capturing the video of the page loading in the browser and tracking its visual progress.

Both metrics are considered a proxy to indicate users' QoE, with some authors [7] arguing that the `SpeedIndex`, along with other metrics not covered in this work, is more representative of users' QoE than the OnLoad time.

These metrics are computed by the web browser at client-side. Collecting them requires the access of users' devices. Content providers and websites usually instrument services to collect such metrics from web browsers and upload results to servers as pages are loaded.

2.2. Challenges for estimating QoE from network traffic

Objective metrics based on Deep Packet Inspection (DPI) [26] no longer work, due to the deployment of encrypted protocols. Methods to compute objective quality metrics must therefore be compatible with the data visible in the network.

ISPs can still rely on flow-level monitoring [17], which provides coarse-grained data about the activity collected at the network and transport layers. Moreover, ISPs usually control key Internet services, e.g., the DNS. PAIN exploits flow-level measurements and DNS information to build models for the traffic of given websites. In the remainder of the paper we assume that both flow level and DNS measurements are available at the ISP.

Obtaining objective quality metrics from such coarse-grained data is not trivial. The complexity of websites has dramatically increased over the years [19], and loading a web page requires reaching dozens of servers and fetching hundreds of objects.

Once users reach a website, browsers open multiple flows to different servers to fetch HTML objects, scripts and media content. We call the domain associated with the first contacted server the *Core Domain* and the remaining contacted domains *Support Domains*.

Figure 1 provides a simplified example: arrows represent the time in which flows to support domains start while the user is visiting the core domain *www.nytimes.com*. In this example, loading the web page requires the browser to open 16 flows to 12 different servers.

Figure 2 provides a more realistic example of the flow-level measurements obtained during visits to a website. It depicts *all* flows to support domains opened during a visit to *www.bbc.co.uk*. This visit has taken around 6 seconds to load all objects. The browser has contacted 94 (unique) support domains. Black lines in the picture represent notable browser events. The browser starts rendering the page at $0.7s$ and finishes parsing the HTML document at time $1.6s$, when the browser has downloaded mainly HTML objects and JavaScripts. Then, it starts to download other page objects (e.g., images and style sheets), firing the `onLoad` event only at $5.4s$. After this, the browser

[3] We use the term *domain* informally throughout the paper, meaning Fully Qualified Domain Name (FQDN).

[4] https://developers.google.com/speed/docs/insights/about

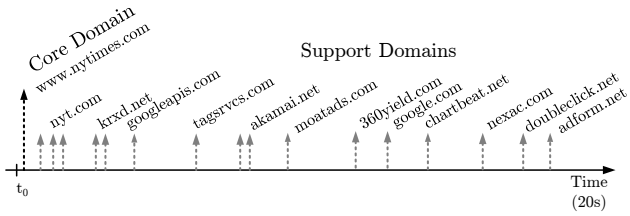


Figure 1: Sample of flows in a visit to *www.nytimes.com*. We use the time to contact support domains to monitor performance.

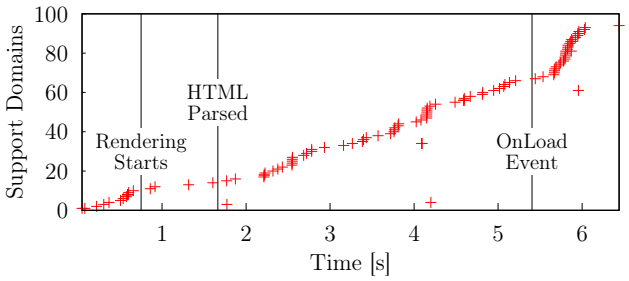


Figure 2: Support domain flows for a visit to *www.bbc.co.uk*. The browser contacted 94 support domains (y-axis) during 6 seconds (x-axis). Notable browser events are reported as vertical lines.

continues to download elements from other servers (and opening new flows). In this example, the page triggers 27 additional connections to domains hosting analytics, advertisements, etc.

The goal of PAIN is to calculate a performance indicator from this kind of traces, which are influenced by browser configurations, website designs, network configuration, etc. PAIN profits from support domains to estimate the performance of the website from flow timings. It is an unsupervised system that automatically learns typically contacted support domains after a core domain visit, and creates models describing the typical order in which support flows appear after the core domain visit. PAIN then considers the delay to observe support flows as performance indicators.

3. Related Work

Previous work focuses on estimating QoE-related metrics from passive network measurements. Authors of [11] show that indirect metrics can serve as indicators for the users' MOS. According to [13], packet losses are strongly correlated with users' session abandonment, thus suggesting that even some low-level network parameters may serve as indicators for users' QoE.

Considering web browsing QoE, past works have shown the difficulties for its estimation, proposing multiple objective metrics to this end. Egger *et al.* [14] show that user-perceived page load times may deviate from common technical metrics used to estimate page load times. Wang *et al.* [34] claim that in-browser computation and blocking Javascripts are significant factors affecting perceived QoE. Metrics such as the `onLoad` time or `SpeedIndex` have been shown to be correlated with QoE metrics, such as users' MOS [12, 15]. Authors of [7] propose `ByteIndex` and `ObjectIndex` – metrics based on the bytes delivered to the client to render a page. Authors of [9] propose the `Above-The-Fold` metric to overcome the limitations of the naive `onLoad` approach. This latter metric is used in combination with classical metrics to predict users' MOS in [12]. In the same direction as these previous works, PAIN provides a new metric to monitor the performance of websites that we will show to correlate well with established objective metrics, but without requiring any client-side instrumentation.

Past works targeting the ISP scenario require either DPI or ground truth from client browsers to train machine learning classifiers. Ibarrola *et al.* [18] build a network emulation system that estimates QoE-related metrics when varying network conditions, based on data collected from volunteers. Shaikh *et al.* [27] study the correlation between physical layer metrics and QoE. The authors however use a page formed by a single object on an in-lab testbed. A similar approach is used by Aggarwal *et al.* [4], where carefully instrumented mobile devices provide the ground truth to train models for predicting QoE parameters.

Other works rely on DPI of the HTTP transactions to gather knowledge about QoE-related metrics [16]. Balachandran *et al.* [5] create models to predict web QoE from passive measurements on cellular networks examining the sequence of HTTP requests. Similarly, some Sandvine's products [26] build dependency graph of web pages extracted from HTTP traffic traces to assess PLT, but they are limited to non-encrypted traffic. All these works are however outdated, since encryption is already the norm in the Internet [33].

Differently from past works, we follow an unsupervised approach, avoiding the need of a resource-consuming testbed to gather client-side metrics. Moreover, PAIN automatically builds models from flow-level traces, with no need to access traffic payloads, thus seamlessly operating with encrypted data carried over HTTPS.

4. The PAIN system

PAIN is an unsupervised system composed by two blocks (see Figure 3). The *Model Learning* module analyzes flow records exported by monitoring devices and creates a model for each core domain of interest, i.e., it discovers and clusters support domains associated to specific websites. It must be continuously updated to cope with changes in website structures. The *PAIN Index Computation* module extracts the actual performance index using the previously built models. All algorithms scale linearly with respect to the input size (i.e., number of flow records), and support scalable processing using big data approaches offered by Apache Spark.

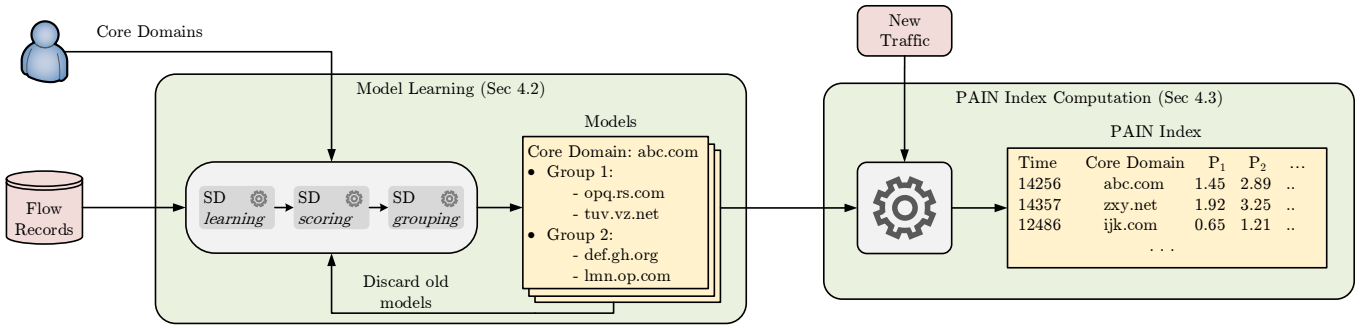


Figure 3: Architecture of PAIN. It learns and clusters support domains using flow records and a list of target core domains. The resulting groups are used to estimate performance.

4.1. Input data

PAIN expects two inputs: (i) Flow records from traffic, and (ii) the list of Core Domains of interest.

Flow records are annotated with time and domain information: Given a flow f, identified by client and server IP addresses, client and server port numbers and the transport-layer protocol, ts_f, te_f are the start and end timestamps, i.e., the time of the first and last packet of the flow. Each flow record must be enriched with information about the server domain d requested by the client.

Flow meters typically export information from the network and transport layers, missing the association between server IP addresses and domain names. To get the server domain, different methods can be used. For example, DNS logs can be employed to extract queries/responses and annotate records in a post-processing phase [6]. Equally, some flow meters export such information on-the-fly directly from the measurement point for popular protocols [17]. For instance, Deep Packet Inspection allows one to extract the Server Name Identification (SNI) from encrypted TLS flows, or the server `Host:` header from plaintext HTTP flows.

The list of Core Domains is a user-defined list containing the set of websites the ISP is interested in monitoring. Since PAIN operates with L4-level measurements and domains names, the analyst must specify only the domain names to be monitored, and not full URLs. This allows PAIN to deal with encrypted traffic. Clearly, PAIN cannot monitor the performance of a single visit to specific web pages. PAIN gains importance when monitoring popular websites accessed by large numbers of users in the network.

4.2. Model learning

The Learning Module observes the timings of flows as seen in the network traffic after a Core Domain. The first task is to learn which support domains are triggered by the core domain visit. PAIN learns that by focusing on the flows *commonly* occurring after core domains appearance in the network.

Given that downloaded HTTP objects while rendering pages vary from visit to visit (e.g., because of caching, persistent connections, modification in the content, personalized content etc.), PAIN analyzes the order in which *groups* of support domains *typically* appear. The rationale is that some support domains may be missing in a visit, while others may not be relevant for indicating the website performance (see Figure 2). PAIN uses groups of support domains to build models that are robust to such variations.

The combination of these building blocks lets PAIN model the typical behavior of the websites hosted in a core domain.

4.2.1. Support domains learning

PAIN learns support domains based on the methodology we introduced in [30]. Let C be the set of *core domains* of interest provided as input. PAIN training consists of learning the set of *support domains* S_c, for each core domain $c \in C$. One possible solution could be using active crawling, e.g., artificially visiting the pages hosted at c and collecting domains being contacted. Unfortunately, this does not work in practice because (i) the same service/website changes when accessed from different location, time, browser, device etc., (ii) c may require authentication, or the usage of a specific application, which complicates the crawling, and (iii) the approach poses scalability issues. PAIN leverages instead data collected from the network traffic itself to build and update the S_c.

The intuition is simple: When a client is observed opening a flow to the core domain c, the domains of flows that follow shall be considered within S_c. However, not all flows are truly linked to c, because the user may access multiple services at the same time (e.g., multiple browser tabs), or because the user terminal may contact unrelated services automatically (e.g., background software updates). In addition, a single support domain may be shared by multiple core domains, while a core domain itself may act as support domain for another services.[5]

[5]Recall Figure 1: `www.google.com` is support domain for `www.nytimes.com`. However, it is a core domain for Google's services.

Figure 4 reports a timeline of flows for a given client, and depicts the intuition behind PAIN learning. PAIN considers a flow f to be a learning sample if its domain $c \in C$. In Figure 4, tall arrows are identified as valid learning samples.

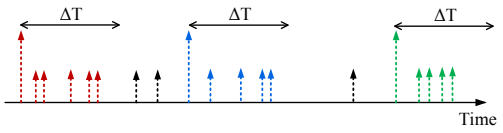


Figure 4: Support domain learning: a flow to a core domain triggers a new observation window, used to learn the support domain set S_c.

When a flow to a core domain is observed, PAIN opens an *Observation Window* (OW) of duration ΔT. Domains of *all* flows observed in ΔT become part of S'_c, the candidates for forming S_c. In Figure 4, they are represented by short arrows sharing colors with core domain flows. The longer ΔT, the more information is collected, with chances of polluting S'_c with *false* support domains. Eventual core domains observed during ΔT will be considered candidate support domains too. PAIN keeps open a single observation window per client during learning.

PAIN avoids polluting S_c by pruning candidate support domains in S'_c based on the frequency $F_{d,c}$ the domain d appears in observation windows of c, leveraging a large number of OWs to get rid of noise. Algorithm 1 describes the procedure for updating $F_{d,c}$ and maintaining the observation window as flows of a client are processed during learning.

Algorithm 1 Process flows of a client updating $F_{d,c}$ and maintaining the observation window.

Require:
- f ▷ The current flow
- $C = \{c_1, ...\}$ ▷ The core domains
- $S' = \{S'_{c_1}, ...\}$ ▷ Candidate support domains for domains in C
- $F = \{F_{d,c_1} ...\}$ ▷ Frequency of candidate support domains

1: $t = GetTime()$ ▷ Get current flow time
2: $d \leftarrow parse(f)$ ▷ Get the domain of f
3: $(t_c, c) \leftarrow ow$ ▷ Retrieve current ow if any
4: **if** $ow \neq \emptyset \wedge t - t_c \geq \Delta T$ **then**
5: $ow \leftarrow \emptyset$ ▷ Remove the ow if expired
6: **if** $ow \neq \emptyset$ **then** ▷ If an ow is open
7: $S'_c \leftarrow S'_c \cup \{d\}$ ▷ Insert d in S'_c
8: $F_{d,c} += 1$ ▷ Update $F_{d,c}$
9: **else**
10: **if** $d \in C$ **then**
11: $ow \leftarrow (t, d)$ ▷ Open a new ow if d is a core domain

Pruning of S'_c is then performed: Actual support domains should consistently appear in multiple observation windows, whereas domains related to background traffic, being present by chance, should be less frequent than support domains. PAIN gets the final set of support domains S_c based on $F_{d,c}$ and a threshold $MinFreq$ as follows:

$$S_c = \{d \,|\, d \in S'_c \wedge F_{d,c} > MinFreq\}. \quad (1)$$

$MinFreq$ is calculated by PAIN directly from the data, observing that $F_{d,c}$ should approach 1 for actual support domains and 0 for domains present in S'_c by chance. PAIN searches for the $MinFreq$ in the interval $[0-1]$ that minimizes the following error function:

$$Err(MinFreq) = \begin{cases} \sum_{d \in S'_c} |F_{d,c} - 1| & if \quad F_{d,c} \geq MinFreq \\ \sum_{d \in S'_c} |F_{d,c} - 0| & if \quad F_{d,c} < MinFreq \end{cases} \quad (2)$$

In our experiments $MinFreq$ results in the $[0.4, 0.5]$ range.

Traffic from all clients contributes to S_c, so that information is accumulated over time and in different conditions, i.e., identities, browsers, devices, configurations etc.

4.2.2. Support domain scores

Intuitively, the timeline of support flows reflects the speed at which a website is loaded (recall Figure 2). Page elements hosted by third-party sites (e.g., images and advertisements) are requested after other components of the page (e.g., scripts) are processed. PAIN leverages this behavior to calculate a score for $d \in S_c$. The score is higher for support domains appearing further away in time from the core domain c (e.g., right-most points in Figure 2).

However, support domains varies from visit to visit or even among pages hosted in the website. S_c is constructed from many observation windows and not all support domains appear in every observation window, e.g., due to caching and persistent connections. Equally, nothing prevents browsers or mobile apps from opening flows to third-parties in a different order while rendering pages.

To determine the score for each $d_i \in S_c$, PAIN computes a *dependency matrix* $\mathcal{M}_c$ of order $|S_c|$ for each core domain c. Each cell $\mathcal{M}_{c_{i,j}}$ represents the number of observations windows OW_c in which the support domain d_i has appeared *after* the support domain d_j in time. Note that $\mathcal{M}_{c_{i,i}} = 0$. Similarly, $\mathcal{M}_{c_{i,j}} = |OW_c|$ only if d_i appears always after d_j, and both d_i and d_j are in all observation windows for the core domain c. The score of d_i is calculated as:

$$score(c, d_i) = \sum_j \mathcal{M}_{c_{i,j}} \quad (3)$$

Note that $score(c, d_i)$ is high if d_i appears often later in time than other domains in the observations windows of c. Similarly, it is lower if d_i usually appear close in time to the core domain. Algorithm 2 reports a pseudocode for the score calculation function. It processes one core domain at a time. PAIN computes the dependency matrix $\mathcal{M}$ (lines 1-6), and, then, uses it to provide the scores (lines 7-8).[6]

4.2.3. Support domain grouping

After scoring, PAIN identifies groups of support domains. By clustering the support domains in some *few*

[6] In PAIN implementation, Algorithms 1 and 2 are both executed on-the-fly as new traffic comes into the system.

Algorithm 2 Compute the scores of support domains for the core domain c.

Require:
c ▷ Core domain to be processed
S_c ▷ Support domains of c
OW_c ▷ Observation windows for c

$\mathcal{M} \in \mathbb{R}^{|S_c| \times |S_c|}$ ▷ Define the dependency matrix
for ow in OW_c **do** ▷ For each observation window
  for d_i in ow **do** ▷ For each support domain in ow
    for d_j before d_i in ow **do** ▷ Supports before d_i in time
      if $d_i \in S_c \wedge d_j \in S_c$ **then**
        $\mathcal{M}_{i,j} += 1$ ▷ Increment $\mathcal{M}_{i,j}$ if supports are in S_c
for d_i in S_c **do** ▷ Compute score for each support domain
  $score(c, d_i) = \sum_j \mathcal{M}_{i,j}$ ▷ Sum the row of $\mathcal{M}$

groups, we filter out the noise caused by missing support domains, besides creating groups of domains that are strongly correlated to web performance.

More precisely, we sort $d_i \in S_c$ in increasing order of $score(c, d_i)$ and split the domains in n groups in G_c, where groups have at least $|G_{c_k}| = \left\lfloor \frac{|S_c|}{n} \right\rfloor$ support domains. G_{c_1} will contain those support domains that often appear the closest to the core domain flow, wheres G_{c_n} will have the support domains that often appear the furthest to the core domain. n is a parameter to be investigated.

The set G – i.e., groups of support domains for core domains C – is the output of the Model Learning module.

4.3. PAIN index computation

The index computation module analyzes live traffic to provide a performance index. Like in the training phase, PAIN analyzes the traffic flows on a per-client basis, chronologically sorted by time. When it encounters a flow to a core domain c, it opens an *observation window* ΔT long. PAIN considers all support domain flows generated by the client within the OW, and accounts them to the corresponding group.

We measure the time at which flows in each group are observed. A visit to a group is considered complete when the *last* flow in the group is observed. For each group G_{c_i} with $i \in 1, \ldots, n$, PAIN calculates the index P_i, equals to the time difference between the starting of the last flow in the group i and the starting time of the core domains c. Note that groups can be absent if none of its support domains is in OW. This can be typically caused by two phenomena: (i) the browser cache contains all the objects that are hosted on a particular domain and (ii) the browser already opened a persistent connection toward the target domains. In this case, we do not consider the sample.

We tested different criteria in place of *last* per group (e.g., average and median) and all lead to worse results. The intuition is that the website performance is mainly driven by the ability of the browser to obtain objects to render the pages, which correlates well with the time late flows are observed in the network. Using the last flow per group also highlights possible degradation of specific servers involved in serving the content.

The tuple $P = \{P_1, \ldots, P_n\}$ represents the performance index for a given visit to the core domain c. By considering all visits from all clients to c, PAIN builds statistics on the performance faced by clients. Due to the intrinsic noisiness of flow-level measurements, PAIN assumes relevance when multiple measures are aggregated to contrast different users, time periods or locations.

4.4. Design decisions, caveats and limitations

The decision of making PAIN a completely unsupervised system is motivated by our goal to monitor a vast range of websites. The system is expected to receive only the list of core domains of interest. It learns models directly from traffic, without requiring human intervention or any information collected at the client-side.

Due to this design, PAIN does not directly provide MOS figures, as reporting the MOS would require involving users directly. However, Section 6 will show that PAIN indexes have strong positive correlations with objective metrics (e.g., `SpeedIndex`). These metrics, in turn, present strong positive correlations to users' MOS [12]. Even if these results do not prove PAIN is strongly correlated to the MOS, they are a strong evidence that PAIN is also positively correlated with the MOS [22].[7]

Other designs would be possible too, such as by using supervised algorithms. The system could train models from network traffic assuming client-side metrics are present. Such a supervised design would result in a system that requires ground truth data captured at client-side for each core domain of interest. The supervised approach would allow one to predict the actual values for objective metrics, e.g., estimating `OnLoad` and `SpeedIndex` from the traffic. We however argue that the absolute values of such metrics are far less useful than contrasting and monitoring the metrics across different users, conditions and time frames. PAIN is fully able to pinpoint *variations* in objective metrics (see Section 6.4) despite not being able to estimate their absolute values.

Moreover, the deployment of supervised alternatives requires a resource-consuming test-bed, in which training should be performed periodically for each monitored website. We have decided to follow the unsupervised approach, since it broaden the PAIN deployability and dramatically enhances training scalability. In Section 6.6 we consider a simple supervised approach and compare it to PAIN. We show that it brings limited benefits.

5. Datasets

In this section we describe our validation datasets. We employ both synthetic datasets generated using a testbed, and real world traces collected in an operational network. They are summarized in Table 1.

[7] Given $\rho_{X,Y}$ and $\rho_{Y,Z}$, it possible to demonstrate that $\rho_{X,Z} > 0$ if $\rho^2_{X,Y} + \rho^2_{Y,Z} > 1$. This is condition is largely satisfied in our case.

Table 1: Description of datasets.

Dataset	Size	Collected on	Collection environment
SynthTypical	11 GB	Testbed	10 websites, 4 (emulated) devices, 8 emulated typical access links
SynthDegraded	11.4 GB	Testbed	2 websites, 4 (emulated) devices, manually degraded access link conditions
RealWorld	495 GB	ISP network	> 100 K websites, 10,000 ADSL installations, 1 year

5.1. Synthetic traces

5.1.1. Testbed

Synthetic traces produced in a testbed allow us to compare PAIN to objective metrics directly collected in the browser. We instrument a PC with *WebPageTest* [3], a tool for web performance assessment. WebPageTest emulates networks based on DummyNet [10], a network emulation tool. Given a list of URLs, it automatically navigates through each page while saving detailed statistics. Many options are available, including the choice of client browser (Chrome and Firefox), device (PCs, tablets and smartphones) and network emulation (e.g., 3G, DSL and Cable). It thus provides the means to emulate users' browsing considering realistic clients and network conditions.

WebPageTest exports the HTTP Archive (HAR) [1] for each page visit. It contains information about the visit as well as statistics for each object: from HTTP-headers, to network-level statistics that describe the TCP connections opened to download objects, including the time in which the TCP connection starts, and the domain associated with it.

Additionally, WebPageTest computes many objective quality metrics. Here, we consider the `OnLoad` and the `SpeedIndex` (see Section 2).

5.1.2. Synthetic datasets

We build two datasets to validate PAIN, namely **SynthTypical** and **SynthDegraded**, with respectively *typical* and *degraded* network conditions.

The **SynthTypical** dataset is built by letting WebPageTest visit 10 popular domains in Italy (listed in Table 4). For each domain, WebPageTest visits the homepage and 9 internal pages for a total of 100 pages.

Since PAIN must work seamlessly regardless of client configurations, we consider 4 different browser and device combinations, which we summarize in Table 2. We consider both Firefox and Chrome running on PCs and we leverage Chrome's features to emulate its use on a smartphone and on a tablet.[8]

Table 2: Browsers and emulated devices in the testbed.

Browser	Device	Operating System
Mozilla Firefox	PC	Windows 10
Google Chrome	PC	Windows 10
Google Chrome	Nexus 7	Android
Google Chrome	iPad Mini	iOS

[8]We skip other browsers such as Edge or Safari, as they are not available in the Linux version of WebPageTest.

Table 3: Settings in the **SynthTypical** dataset. *Native* corresponds to a scenario with no traffic shaping.

Name	Down Link	Up Link	RTT
Native	1 Gbit/s	1 Gbit/s	native
FIOS	20 Mbit/s	5 Mbit/s	4 ms
Cable	5 Mbit/s	1 Mbit/s	28 ms
DSL	1.5 Mbit/s	1 Mbit/s	50 ms
LTE	12 Mbit/s	12 Mbit/s	70 ms
3G Fast	1.6 Mbit/s	768 Kbit/s	150 ms
3G	1.6 Mbit/s	768 Kbit/s	200 ms
3G Slow	780 Kbit/s	330 Kbit/s	200 ms

We consider 8 access network technologies summarized in Table 3. These are emulated by WebPageTest by imposing traffic shaping policies that mimic actual parameters of the technologies. The *Native* case has no shaping – i.e., the 1 Gbps Ethernet network connecting the testbed is used without changes. For other cases, DummyNet enforces typical bandwidth and Round Trip Time (RTT) faced by users of a given technology.

We visit each page twice for each setup: (i) with empty browser cache; and (ii) few seconds later for profiting from caching. The traffic is expected to vary strongly, since many objects are cached in the second case, complicating the identification of support domains. In total, WebPageTest recorded 6 400 visits while building this first dataset (all visits have been completed in about 48 h).

The second dataset, **SynthDegraded**, represents artificial conditions, in which we enforce link delay or bandwidth limits. We simulate scenarios in which website performance decreases due to worsening network conditions. We simulate 10 cases: (i) adding from 100 ms to 500 ms extra per-packet delay and (ii) imposing a limit from 2.5 Mbit/s to 312.5 kbit/s on uplink and downlink access bandwidth. Again, we visit each page twice (cold and warm cache) and with 4 browsers. For the sake of brevity, we performed these experiments for 2 websites only, namely *www.repubblica.it* and *www.subito.it*. WebPageTest has performed 8 000 visits for building this second dataset (completed in about 60 h).

5.2. ISP flow traces

This dataset includes flow summaries exported by Tstat [32] in a real deployment. Tstat is a passive monitor able to collect rich flow summaries. It exposes more than 100 metrics, including the typical ones exported by popular flow meters, such as server IP addresses contacted by clients, timestamps of the first packet in each flow and bytes counters per flow. Tstat associates flow records to

Table 4: Support domains for websites in **SynthTypical** dataset, and frequency they appear after `onLoad`.

Core domain	Support domains			After
	Min	Median	Max	OnLoad
www.corriere.it	30	57	137	2.2 %
www.ebay.it	2	50	223	40.5 %
www.gazzetta.it	25	58	138	6.5 %
www.ilmeteo.it	17	56	185	18.5 %
www.lastampa.it	14	34	81	8.7 %
www.meteo.it	27	52	91	6.6 %
www.mymovies.it	24	45	147	11.0 %
www.repubblica.it	27	53	216	23.0 %
www.subito.it	26	52	119	7.0 %
www.wordreference.com	2	14	68	6.0 %

Table 5: Similarity of support domain across different sub-pages and devices (**SynthTypical** dataset).

Core domain	Support domains similarity	
	Subpages	Devices
www.corriere.it	0.69	0.78
www.ebay.it	0.16	0.68
www.gazzetta.it	0.67	0.81
www.ilmeteo.it	0.90	0.68
www.lastampa.it	0.61	0.79
www.meteo.it	0.87	0.66
www.mymovies.it	0.69	0.59
www.repubblica.it	0.56	0.74
www.subito.it	0.82	0.58
www.wordreference.com	0.89	0.41

domain names requested by clients using the SNI information from TLS handshakes and by exploiting DNS traffic also observed in the network [6].

We have instrumented a Point of Presence (PoP) of a European ISP, where $\approx 10,000$ ADSL customers are aggregated. The ISP provides us the access link speed of each ADSL customer. Moreover, each customer is provided a fixed IP address and, thus, by inspecting the (anonymized) client IP addresses in our dataset, PAIN isolates flows per ADSL installation and use them as the per-client traces. Each trace includes information about traffic of all users' devices connected at home.

We here consider data from the whole year 2017. Considering only HTTP and HTTPS flows, we obtain 15 billion flows related to around $100,000$ websites. This trace represents a realistic scenario of a possible PAIN deployment. No ground truth about associations of support and core domains is available in the dataset.

6. Validation

6.1. Support domains at a glance

We first provide high-level statistics about support domains (see Table 4). We aim at complementing Figure 2, illustrating the challenges to extract knowledge from support domains and their complex relations with the page loading process. Table 4 lists the websites in **SynthTypical** dataset. The 3rd column reports the median number of support domains across all visits: They vary from less than 20 to more than 50. The number and order at which support domain flows are opened vary across visits (see 2nd and 4th columns of Table 4). More than that, support domains are often contacted after the `OnLoad` event has fired, e.g., due to browser pre-fetching or the presence of analytics scripts programmed to run after the page is loaded. We quantify the percentage of these cases in the 5th column of the table. Extreme is the case of *www.ebay.it*: More than 40% of connections are issued after the browser completed loading the page.

These results already hint for the importance of PAIN grouping step. For example, if one would naively take the delay of the last support flow as a performance indicator, the obtained metric would likely have very low correlation with objective quality metrics observed at the client-side, as such metrics represent events happening much earlier in time than late flows.

6.2. Support domains across sub-pages and devices

PAIN aims at monitoring whole websites, represented by a core domain. However, websites host many web pages, which may rely on completely different support domains. Moreover, it is unclear to what extent the set of support domains remain similar when websites are accessed from different devices. We now quantify these effects to verify whether PAIN could be applied in such diverse scenarios.

We perform an analysis aiming at quantifying the *variability* of support domains in different scenarios. First, we compare the list of support domains obtained when considering each sub-page of the websites in the **SynthTypical** dataset. For each website, we compute the Jaccard index similarity coefficient [21] for the sets of support domains contacted for each pair of sub-pages. Table 5 reports the median values obtained for each websites.

If the Jaccard index is equal to one, the sets of support domains are equal. We can see in the table that median values are indeed high for the evaluated websites. That is, sub-pages of these sites usually share most support domains, e.g., with *www.ilmeteo.it* reaching a 0.9 median similarity coefficient. Low values are observed for *www.ebay.it* (0.16), where manual inspection reveals that some sub-pages have a simpler structure, relying on a lower number of support domains. PAIN would fail to identify the groups of support domains when people visit these simpler pages. It will thus ignore these samples when calculating the performance metric.

We repeat the operation for the 4 emulated devices in the **SynthTypical** dataset (recall Table 2). Overall, varying the device used for accessing the web page does not affect the contacted support domains. The lowest value is observed for *www.wordreference.com*, where the median similarity coefficient is 0.41. As above, these results show that the set of contacted support domains is rather stable when varying devices, allowing PAIN to operate even if different devices are connected to the monitored network.

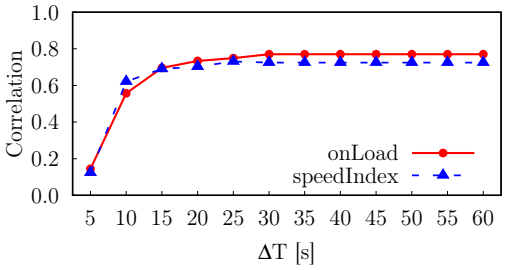
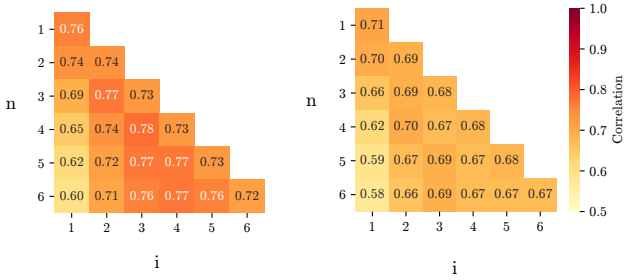


Figure 5: Spearman Correlation of P_3 with `onLoad` and `SpeedIndex` when varying ΔT (**SynthTypical** dataset).



(a) OnLoad (b) SpeedIndex

Figure 6: Correlation of PAIN index with `onLoad` and `SpeedIndex` when varying the number of groups n (**SynthTypical** dataset).

6.3. Tuning of parameters ΔT and n

We now tune the parameters ΔT and n. We rely on the **SynthTypical** dataset. We vary each parameter while comparing the PAIN index to the metrics exposed by our testbed, i.e., `onLoad` and `SpeedIndex`. Indeed, we want the PAIN index to be correlated with the objective metrics, since a high correlation with these metrics would suggest that PAIN is also correlated to the users' MOS. We quantify correlation using the Spearman's rank correlation coefficient between PAIN index and objective metrics [28]. A Spearman coefficient higher than 0.5 is usually considered a strong correlation indication.

We first observe the impact of the observation window choice (ΔT) in Figure 5. Only the correlations between objective metrics and the 3rd group of support flows (i.e., P_3) are shown to improve visualization. Notice in the figure that PAIN achieves high correlation coefficients when ΔT increases. When ΔT value is larger than $30s$, results do not improve further. In a nutshell, PAIN is not very sensitive to ΔT. Provided that support domains are grouped, and each group is used to extract P_i, PAIN index remains mostly unaffected, even if some support domains are not associated to the respective core domain because ΔT is expired. In the following, we set $\Delta T = 30s$.

We next perform a similar analysis for n, the number of groups. We report results for the **SynthTypical** dataset for `onLoad` and `SpeedIndex` separately in Figure 6a and Figure 6b, respectively. Each row j represents an experiment with a different $n \in [1, 6]$. The column i reports the correlation of P_i when using $n = j$. For example, the leftmost cell on the last row represents the correlation of P_1 with `onLoad` when using $n = 6$.

PAIN is not very sensitive to n either, i.e., results always show high correlation values. Some other interesting observations can be extracted from the figures too.

First, notice that for both `onLoad` and `SpeedIndex` the highest correlation is usually not achieved with the latest group for $n > 1$. This result confirms our intuition from Figure 2, that flows opened after the page has finished loading would decrease the correlation between late groups of flows and the objective metrics. Remind that we always take the arrival time of the last observed flow in each group as performance metric. Using a small value of n provides poor information. Considering multiple groups, on the other hand, makes PAIN more robust to outliers.

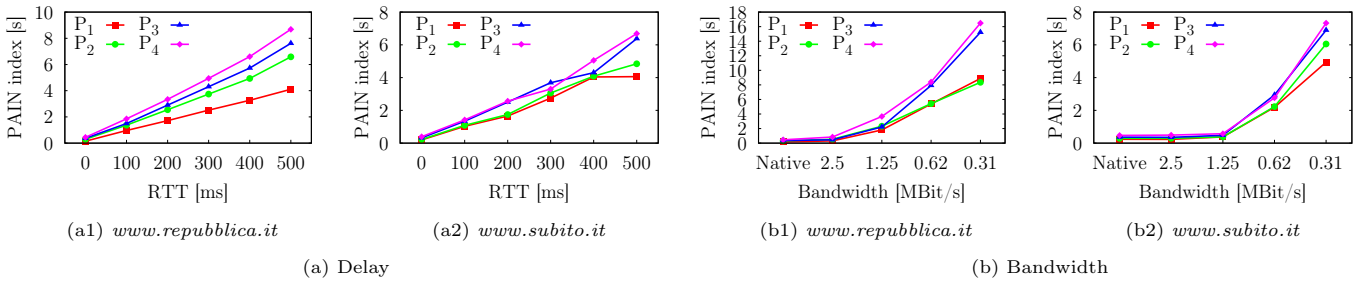
Second, comparing correlations with `onLoad` and `SpeedIndex` for a single n, notice how the best correlation is achieved with earlier groups (i.e., smaller i) for the `SpeedIndex`, in particular for large n. This is due to the fact that the `SpeedIndex` is computed based on the visual progress of the page, which is usually achieved earlier than the `onLoad` event. Different PAIN groups correlate better with each metric.

Finally, $n = 4$ seems to be a good trade-off for both metrics. P_2 and P_3 are the groups correlated the most with objective quality metrics when $n = 4$, and in general, P_i assumes more significance when i is close to n. We take P_3 with $n = 4$ for the remaining experiments. However, Figure 6 shows that small variations of n and P_i do not affect the results, and our experiments reinforce this claim.

6.4. Effects of network conditions

We check whether PAIN can reflect worsening on network conditions using the **SynthDegraded** dataset. Figure 7 illustrates PAIN index values when varying delay and bandwidth to reach the two websites in the dataset. Each point in the figure depicts the median value for the PAIN index over all tests with the given setup. Each point is the result of 80 runs, i.e., 10 pages $\times$ 4 browsers/OS $\times$ 2 repetitions, thus covering experiments for different browsers, sub-pages, etc. Even if omitted for improve visualization, results show low variability (computed as the interquartile range), always lower than 15% from the median.

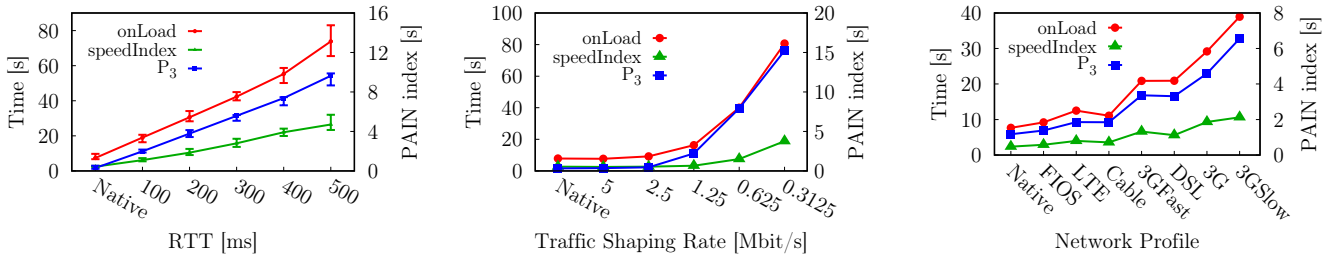
Consider Figure 7a, which refers to *www.repubblica.it* and *www.subito.it*, when RTT $\in [0, 500]$ ms. PAIN index increases alongside the delay, starting from around 0.5 s and up to almost 10 s when RTT is 500 ms for *www.repubblica.it*. That is, P_i reflects the network conditions and increases in case of degradation. Actual PAIN index values are sometimes inverted from their original order for extreme values of RTT (e.g., P_3 larger than P_4). This happens because the order at which a browser opens connections towards support domains is subject to variations. Similarly, in Figure 7a, P_1 has a slightly lower value



(a1) *www.repubblica.it* (a2) *www.subito.it* (b1) *www.repubblica.it* (b2) *www.subito.it*

(a) Delay (b) Bandwidth

Figure 7: Median value of PAIN index when varying delay and bandwidth (**SynthDegraded** dataset).



(a) Delay (b) Bandwidth (c) Profile

Figure 8: *www.repubblica.it* `onLoad`, `SpeedIndex` and P_3 for various setups (**SynthTypical** and **SynthDegraded** datasets).

for 500 ms than for 400 ms. Indeed, this confirms that P_1 and P_2 are not as good as P_3 and P_4 as indicators of the website conditions, reinforcing results of Figure 6.

Similar considerations hold for Figure 7b, which shows the impact of download link capacity. When the available bandwidth is reduced, PAIN index increases. Observe that a bandwidth of 1.25 Mbit/s already implicates performance degradation for *www.repubblica.it*, while still no penalty is suffered by *www.subito.it*.

In summary, results show that P_i reflects the network conditions, allowing ISPs to track degradation on the network that impacts website performance.

6.5. Comparison to objective metrics

We have seen in Figure 6 that PAIN index is correlated to objective quality metrics. We now detail that analysis, by directly comparing the values of P_3 to the `SpeedIndex` and `onLoad`. We set $n = 4$ and $\Delta T = 30s$. Figure 8 reports results obtained for a single website in different scenarios. Similar figures are obtained for other cases. Again, the figure reports median values over 80 runs. Figure 8a also reports error bars that span over 25^{th} and 75^{th} percentiles. We use this figure to illustrate the variability of our results, which we recall to be always limited to less than 15% of the median value. Similar results are obtained for the other two figures, but they are not reported as the error bars would overlap and compromise readability.

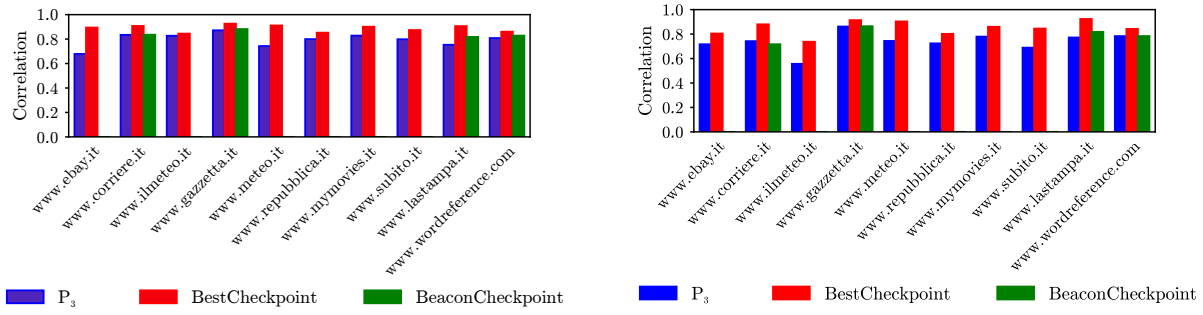
Each point in Figure 8 represents the median value for all visits with the given network condition. Since the metrics have different absolute values, we use the y-axis in the left-hand side to report `SpeedIndex` and `onLoad` times, and the y-axis in the right-hand side to report values of the PAIN index. Thus, the figure shows whether the metrics present similar rate of variation given changes in the network conditions.

Focusing on Figure 8a notice how the three metrics grow almost linearly with the RTT. The rate of variation in PAIN (see blue line) is similar for `SpeedIndex` (green) and `onLoad` (red) ones. When varying the bandwidth in the degraded scenario (Fig. 8b), the values of PAIN index change similarly to the rate observed for `onLoad` time, but faster than `SpeedIndex`. PAIN is more sensitive to deterioration on the available capacity. Yet, results show that the PAIN index is directly related to the website performance. Observe also that all three metrics are basically constant when the bandwidth is larger than 2.5 Mbit/s (see points in the left part of the figure). That is, the web page performance is not affected when a minimum bandwidth is available, and all three metrics reflect such behavior. Finally, Figure 8c reports the values for typical network scenarios. Again, we see similar patterns among the metrics, with the rate of variation of PAIN index in between the other metrics.

In summary, results reinforce that the metrics are correlated, and they vary according to the network conditions similarly. Absolute values are in different ranges, but they all reflect degradation in quality.

6.6. Comparison to alternative approaches

We validate PAIN against two possible alternatives:



(a) OnLoad

(b) SpeedIndex

Figure 9: Correlation of PAIN, **BestCheckpoint** and **BeaconCheckpoint** with objective metrics (**SynthTypical** dataset).

(i) **BestCheckpoint**: We use a *supervised* mechanism to extract a performance metric that tries to maximizes the correlation with objective metrics. Considering a training dataset and a core domain c, we extract the delay to observe each support domain $s \in S_c$ after all visits to c. Then, we compute the correlation coefficient between the delays for each $s \in S_c$ and the objective metrics (`SpeedIndex` and `onLoad`). We select the most correlated support domain to serve as landmark.

When evaluating new traffic, the delay to observe the landmark is considered as the performance metric for the core domain. Note that this supervised approach requires per-site objective metrics at training time.

(ii) **BeaconCheckpoint**: This approach has been proposed by authors of [20]. It consists in leveraging the analytics objects typically present in web pages to identify when page loading is complete. The intuition comes from the fact that analytics services wait for the browser to finish rendering the page before sending back statistics to the server. Here, we consider the *Google Analytics* script that uploads statistics to Google servers after the `onLoad` event is fired by the browser. After finding a flow to the core domain of interest, we search the HTTP requests to *Google Analytics* URL. Note that such an approach requires non-encrypted traffic and works only for sites embedding analytics scripts (e.g., only present in 4 websites in **SynthTypical**).

The delay between the core domain flow and *Google Analytics* request is reported as performance metric.

Figure 9 shows the correlation of PAIN, **BestCheckpoint** and **BeaconCheckpoint** with `SpeedIndex` and `OnLoad`. **BeaconCheckpoint** can be computed only for 4 websites. As we have seen before, PAIN correlation coefficients are positive and very high. Considering `onLoad` in Figure 9a, they range from 0.67 for *www.ebay.it* to 0.90 for *www.gazzetta.it*. Most values are close to 0.8 for both metrics. **BestCheckpoint** and **BeaconCheckpoint** are also positively correlated to the objective metrics. For example, for *www.gazzetta.it*, they achieve 0.92 and 0.88, respectively. **BestCheckpoint** is more strongly correlated to `onLoad` than PAIN. This is expected because of the supervised approach. Yet, absolute differences are small, showing that PAIN can achieve similar performance without the burdens of building ground truth for training the models.

Similar conclusions hold for `SpeedIndex` in Figure 9b. PAIN correlations coefficient span from 0.55 for *www.ilmeteo.it* to 0.86 for *www.gazzetta.it*, with other metrics in similar ranges.

Summarizing, PAIN index is strongly correlated with both objective metrics for different sites. PAIN achieves similar performance than other approaches, which are however hardly feasible in real deployments.

6.7. Learning duration and periodicity

Next we investigate the number of observation needed to learn support domains, and for how long the models remain valid. This information defines the duration and periodicity of PAIN learning. Since PAIN is unsupervised, it learns models directly from live traffic. Large learning periods should help creating robust models. On the other hand, sites may change over time invalidating the models.

We first evaluate how the size of the learning sample impacts PAIN. We perform experiments with the **RealWorld** dataset. Since we aim at checking how the models behave in large samples and long periods, we focus on the top-100 ranked sites in Italy by Alexa.

In Figure 10, we let PAIN learn support domains with an increasing number of observations per core domain. We then compare the selected support domains with the set obtained with the largest observation period – i.e., when all core domains have been observed at least $10,000$ times. The y-axis reports how similar the two sets are using the Jaccard similarity coefficient [21]. Clearly, the right-most point has value 1 (perfect similarity). Other points confirm that the larger the observation period is, the more stable the sets become. Indeed, after some thousand observations the similarity reaches almost a plateau, with the Jaccard coefficient at around 0.80 with $5,000$ observations. This figure suggests that some thousand observations are sufficient to learn stable sets of support domains.

A question still remains: How often should PAIN learning be performed? Performing learning sporadically may

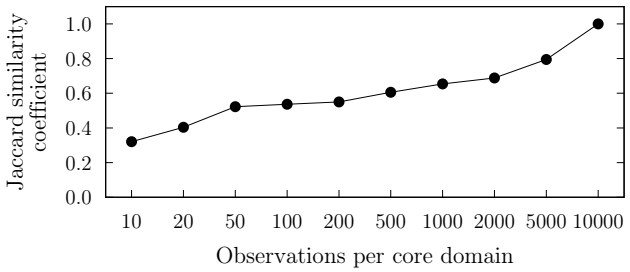


Figure 10: Support domains for increasing number of observations per core domain, compared to 10,000 observations (**RealWorld**).

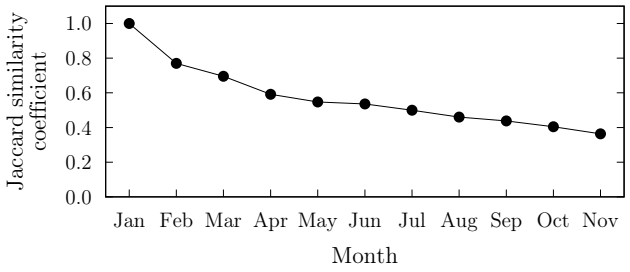


Figure 11: Support domains over the months of a year (**RealWorld**).

let models get outdated and reduce the metrics precision. We quantify this phenomenon in Figure 11. We let PAIN run on the **RealWorld** dataset using the previous subset of domains, learning support domains separately for each month. Then, we compare the learned sets at each month with those learned during the January 2017. Again, we use the Jaccard coefficient as similarity metric.

The figure shows that support domains learned on February have a 0.77 similarity coefficient with those learned on January. The similarity decreases to 0.69 on March, and finally to 0.36 on November. It is clear that even in short periods, e.g., a couple of months, the learned support domains diverge significantly. While PAIN grouping approach partly compensates for such variations, these results suggest that continuously updating support domains is advisable to retain PAIN performance.

In summary, PAIN benefits from a large number of observations to learn models of support domains for the websites. Few thousands of samples per core domain seem sufficient to bootstrap the system. On the other hand, learning must be continuous, with models being updated to avoid using outdated sets of support domains.

7. Case studies

We now report our experience when using PAIN in a real deployment. We exploit the **RealWorld** dataset, containing flow-level measurements of around 10,000 ADSL customers over one year. More concretely, we run PAIN to understand (i) whether web browsing performance changes for different ADSL installations; (ii) the impact of large server-side events on users' experience.

PAIN learns the models on the **RealWorld** dataset on a per-month basis. We focus on the top-100 Alexa rank for Italy. PAIN is set with $n = 4$, $\Delta T = 30s$.

7.1. Performance per ADSL capacity

For ISPs, it is important to understand the impact of access link capacity on web browsing. For example, ISPs are interested in knowing whether users with poor connectivity are significantly impaired while surfing the Web, e.g., to propose upgrades to such users. PAIN allows ISPs to estimate how objective metrics (i.e., `OnLoad` and `SpeedIndex`) vary across users, even if these metrics are not measurable with passive monitoring.

We know the download access link capacity of each ADSL installation in the **RealWorld** dataset. We thus divide users in three categories: (i) slow (< 4 MBit/s), (ii) medium (4–12 Mbit/s) and (iii) fast (> 12 Mbit/s). We then compute PAIN P_3 for users of each group.

Results for two news websites in Italy are reported Figure 12. For *www.lastampa.it* (Figure 12a), distributions are clearly not overlapping. PAIN index decreases significantly when the access capacity increases. Indeed, the median value moves from $9.6s$ for slow users to $4.3s$ for fast users. For *www.repubblica.it* (Figure 12b), differences across users are even more pronounced. PAIN index median value is $12.3s$ for slow users and $5.1s$ for fast users.

These results allow quantifying the role of access capacity on page load time in the real world, where previous experiments relied only on testbeds [24, 23].

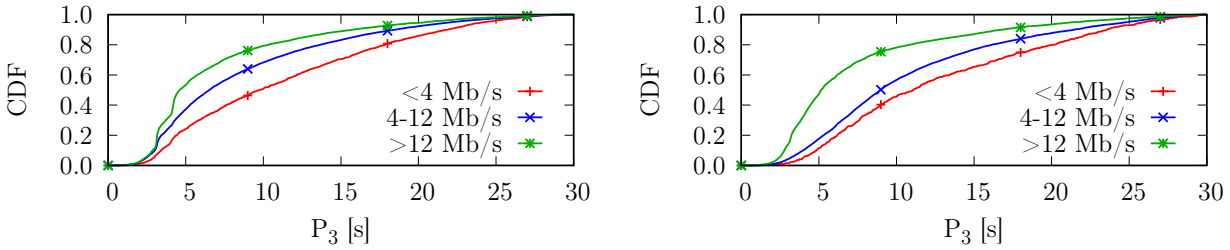
7.2. Impairments due to server-side events

ISPs can rely on PAIN index to monitor anomalies causing real impact on users' performance. To this end, we illustrate some noticeable episodes emerging from the **RealWorld** dataset. We let PAIN run on the entire dataset for the top-100 Alexa services. We then manually went through the obtained time series to find episodes worth of attention, such as abrupt changes in PAIN index. Prominent cases have been further investigated, to uncover possible reasons behind the sudden changes.

Figure 13 reports an episode related to *www.poste.it*, the website of the Italian national mail service. On January 18th 2017, the median PAIN index incurs sudden increase: The median value for P_3 grows from the $[4, 6]s$ range to the $[8, 10]s$ range, while median P_4 increases from $[8, 10]s$ to $[10, 12]s$ ranges (see y-axis the figure).

Investigating the root-cause for this change in behavior, we discovered that the website switched all services to HTTPS on that date. As such, the additional load imposed to both servers and clients is likely causing a performance impairment.

Figure 14 depicts a second prominent episode uncovered by PAIN, related to *www.repubblica.it*. Recall that this site hosts a major Italian news portal. The website



(a) *www.lastampa.it* (b) *www.repubblica.it*

Figure 12: Distribution of PAIN P_3 index according to the access-link capacity for all visits in **RealWorld** dataset

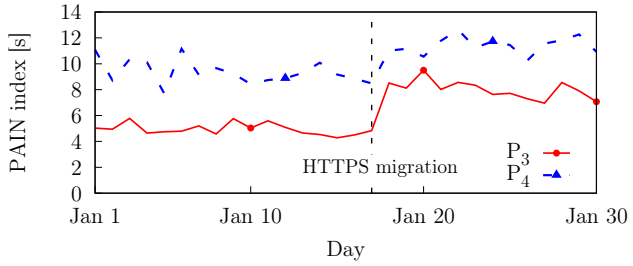


Figure 13: PAIN index for *www.poste.it* over 1 month (**RealWorld**).

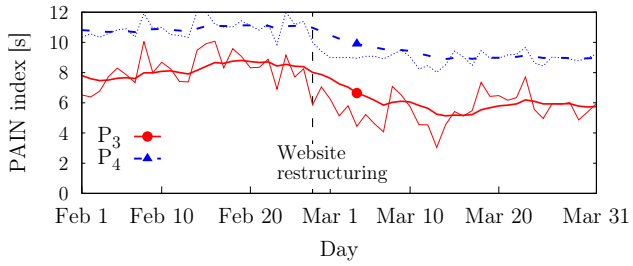


Figure 14: PAIN index trend for *www.repubblica.it* before and after website restructuring (**RealWorld**).

passed a major reorganization of layout and content on 27th February 2017. The portal claimed at the time that the reorganization would lead to performance improvements for its users.

PAIN is able to measure the website performance before and after the restructuring. Figure 14 depicts P_3 and P_4 evolution in time. The median values computed per day are reported with thin lines, with thick lines marking the exponentially weighted moving average (EWMA) of the values. The performance of the website has improved after the migration day. P_3 decreases from $\approx 8s$ to $\approx 6s$, while P_4 from $\approx 11s$ to $\approx 9s$.[9]

In summary, these case studies illustrate how PAIN can be used to spot changes in websites performance, due to intrinsic characteristics of the network or external events (e.g., websites modifications). PAIN can be used to trigger alerts in case of sudden changes in performance, driving ISPs to further investigate the problems that are relevant to users' experience.

[9] A one-tailed T-Test confirms that differences for values before and after the migration are statistically significant.

8. Conclusions

We presented PAIN, an automatic and unsupervised system to monitor website performance using flow-level measurements, and release it as open source. PAIN builds a behavioral model for the websites' traffic, leveraging flows automatically opened by browsers to retrieve images, scripts etc. The model is used for assessing performance.

We validated PAIN by showing that it can highlight sudden performance deterioration due to changes on network conditions. We showed that PAIN metrics are strongly correlated with well-known objective metrics used as indication of users' QoE, i.e., `onLoad` time and `SpeedIndex`. Moreover, we showed that PAIN performance is similar to supervised alternatives, which are however harder to be deployed in practical scenarios.

Finally, we deployed PAIN in an ISP network for one full year. PAIN allowed us to quantify website performance differences across customers with different access link capacities. Moreover, PAIN pinpointed sudden performance variations for websites that incurred restructuring.

Acknowledgements

The research leading to these results has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129 (BigDAMA) and the SmartData@PoliTO center for Big Data technologies.

References

[1] (2012). HAR 1.2 Spec. `https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html`.
[2] (n.d.). Squid-Cache. `http://www.squid-cache.org/`.
[3] (n.d.). WebPageTest. `https://www.webpagetest.org/`.

[4] Aggarwal, V., Halepovic, E., Pang, J., Venkataraman, S., and Yan, H. (2014). Prometheus: Toward Quality-of-experience Estimation for Mobile Apps from Passive Network Measurements. In *Proceedings of the HotMobile*, pages 18:1–18:6.

[5] Balachandran, A., Aggarwal, V., Halepovic, E., Pang, J., Seshan, S., Venkataraman, S., and Yan, H. (2014). Modeling Web Quality-of-experience on Cellular Networks. In *Proceedings of the MobiCom*, pages 213–224.

[6] Bermudez, I., Mellia, M., Munafò, M. M., Keralapura, R., and Nucci, A. (2012). DNS to the Rescue: Discerning Content and Services in a Tangled Web. In *Proceedings of the IMC*, pages 413–426.

[7] Bocchi, E., Cicco, L. D., and Rossi, D. (2016). Measuring the Quality of Experience of Web Users. In *Proceedings of the Internet-QoE*, pages 37–42.

[8] Bocchi, E., Cicco, L. D., Mellia, M., and Rossi, D. (2017). The Web, the Users, and the MOS: Influence of HTTP/2 on User Experience. In *Proceedings of the PAM*, pages 47–59.

[9] Brutlag, J., Abrams, Z., and Meenan, P. (2011). Above the fold time: Measuring web page performance visually. `https://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692`.

[10] Carbone, M. and Rizzo, L. (2010). Dummynet revisited. *SIGCOMM Comput. Commun. Rev.*, **40**(2), 12–20.

[11] Casas, P., Seufert, M., Wamser, F., Gardlo, B., Sackl, A., and Schatz, R. (2016). Next to You: Monitoring Quality of Experience in Cellular Networks From the End-Devices. *IEEE Trans. Netw. Service Manag.*, **13**(2), 181–196.

[12] Da Hora, D., Asrese, A. S., Christophides, V., Teixeira, R., and Rossi, D. (2018). Narrowing the gap between QoS metrics and Web QoE using Above-the-fold metrics. In *PAM 2018 - International Conference on Passive and Active Network Measurement*, pages 1–13, Berlin, Germany.

[13] Din, I., Saqib, N. A., and Baig, A. (2008). Passive analysis of web traffic characteristics for estimating quality of experience. In *2008 IEEE Globecom Workshops*, pages 1–5. IEEE.

[14] Egger, S., Reichl, P., Hoßfeld, T., and Schatz, R. (2012). "time is bandwidth"? narrowing the gap between subjective time perception and quality of experience. In *Communications (ICC), 2012 IEEE International Conference on*, pages 1325–1330. IEEE.

[15] Gao, Q., Dey, P., and Ahammad, P. (2017). Perceived performance of top retail webpages in the wild: Insights from large-scale crowdsourcing of above-the-fold qoe. In *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, pages 13–18. ACM.

[16] Gonzalez, R., Soriente, C., and Laoutaris, N. (2016). User profiling in the time of https. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 373–379, New York, NY, USA. ACM.

[17] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., and Pras, A. (2014). Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX. *Commun. Surveys Tuts.*, **16**(4), 2037–2064.

[18] Ibarrola, E., Taboada, I., Ortega, R., *et al.* (2009). Web qoe evaluation in multi-agent networks: Validation of itu-t g. 1030. In *Autonomic and Autonomous Systems, 2009. ICAS'09. Fifth International Conference on*, pages 289–294. IEEE.

[19] Ihm, S. and Pai, V. S. (2011a). Towards understanding modern web traffic. volume 39, pages 335–336, New York, NY, USA. ACM.

[20] Ihm, S. and Pai, V. S. (2011b). Towards understanding modern web traffic. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 295–312, New York, NY, USA. ACM.

[21] Jaccard, P. (1912). The distribution of the flora in the alpine zone. *New phytologist*, **11**(2), 37–50.

[22] Langford, E., Schwertman, N., and Owens, M. (2001). Is the property of being positively correlated transitive? *The American Statistician*, **55**(4), 322–325.

[23] Liu, Y. (2017). Demystifying mobile web browsing under multiple protocols. *arXiv preprint arXiv:1712.00237*.

[24] Meenan, P. (2013). How fast is your website? *Communications of the ACM*, **56**(4), 49–55.

[25] Paxson, V. (1999). Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, **31**(23-24), 2435–2463.

[26] Sandvine (2015). Measuring Web Browsing Quality of Experience. `https://www.sandvine.com/hubfs/downloads/archive/whitepaper-web-browsing-qoe.pdf`.

[27] Shaikh, J., Fiedler, M., and Collange, D. (2010). Quality of experience from user and network perspectives. *annals of telecommunications-annales des telecommunications*, **65**(1-2), 47–57.

[28] Spearman, C. (1904). The proof and measurement of association between two things. *The American journal of psychology*, **15**(1), 72–101.

[29] Trevisan, M. (2017). NetLytics. `https://github.com/marty90/netlytics`.

[30] Trevisan, M., Drago, I., Mellia, M., Song, H. H., and Baldi, M. (2016). WHAT: A Big Data Approach for Accounting of Modern Web Services. In *Proceedings of the BigData*, pages 2740–2745.

[31] Trevisan, M., Drago, I., and Mellia, M. (2017a). Pain: A passive web speed indicator for isps. In *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, Internet QoE '17, pages 7–12, New York, NY, USA. ACM.

[32] Trevisan, M., Finamore, A., Mellia, M., Munafo, M., and Rossi, D. (2017b). Traffic analysis with off-the-shelf hardware: Challenges and lessons learned. *IEEE Communications Magazine*, **55**(3), 163–169.

[33] Trevisan, M., Giordano, D., Drago, I., Mellia, M., and Munafò, M. (2018). Five years at the edge: Watching internet from the isp network. In *Proceedings of CoNEXT'18*, Heraklion, Greece.

[34] Wang, X. S., Balasubramanian, A., Krishnamurthy, A., and Wetherall, D. (2013). Demystifying page load performance with wprof. In *NSDI*, pages 473–485.

Authors

Martino Trevisan received his B.Sc. (2012) and his M.Sc. (2015) in Computer Science, both from Politecnico di Torino, Italy. He is currently a PhD student in Electrical, Electronics and Communications Engineering in the same university, where he joined the Telecommunication Networks Group (TNG). He has been collaborating in both Industry and European projects. His research interest areas include Network Measurements and Traffic Monitoring, while he is also particularly interested in leveraging Big Data and Machine Learning techniques in such fields.

Idilio Drago is an Assistant Professor at the Politecnico di Torino, Italy, in the Department of Electronics and Telecommunications. His research interests include Internet measurements, Big Data analysis, and network security. Drago has a PhD in computer science from the University of Twente. He was awarded an Applied Networking Research Prize in 2013 by the IETF/IRTF for his work on cloud storage traffic analysis.

Marco Mellia is Associate Professor at the Politecnico di Torino, Italy. His research interests are in the in the area of traffic monitoring and analysis, in cyber monitoring, and Big Data analytics. Marco Mellia has co-authored over 250 papers published in international journals and conferences. He won the IRTF ANR Prize at IETF-88, and best paper award at IEEE P2P'12, ACM CoNEXT'13, IEEE ICDCS'15. He is part of the editorial board of ACM/IEEE Transactions on Networking, IEEE Transactions on Network and Service Management, and ACM Computer Communication Review.