PAIN: A Passive Web Speed Indicator for ISPs

PAIN: A Passive Web Speed Indicator for ISPs

Martino Trevisan
Politecnico di Torino
martino.trevisan@polito.it

Idilio Drago
Politecnico di Torino
idilio.drago@polito.it

Marco Mellia
Politecnico di Torino
marco.mellia@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 Internet Service Providers (ISPs) to anticipate problems. Quality is subjective, and the complexity of today's pages challenges its measurement. `OnLoad` time and `SpeedIndex` are notable attempts to quantify web performance. However, these metrics are computed using browser instrumentation and, thus, are not available to ISPs.

PAIN (PAssive INdicator) is an automatic system to observe the performance of web pages at ISPs. It leverages passive flow-level and DNS measurements which are still available 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 analyze the web performance in real-time. We compare PAIN to indicators based on in-browser instrumentation and find strong correlations between the approaches. It reflects worsening network conditions and provides visibility into web performance for ISPs.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; **Network measurement**;

KEYWORDS

QoE Metrics; Passive Measurements; Machine Learning

1 INTRODUCTION

Metrics related to users' Quality of Experience (QoE) are key to understand how end-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 (ISP), which need to be aware of web performance to actuate when network factors affect web browsing experience [11].

In this paper we take the perspective of ISPs. Bad performance in the network, and in particular in the last-mile, is historically the first suspect in degradation events. This has motivated major Content Providers to publicize rankings of ISP performance.[1] It is no exaggeration to say that ISPs are evaluated based on the experience of end-users while interacting with third-party services. Moreover, ISPs need to understand the impacts of possible network configuration changes on web browsing performance – e.g., to decide whether the deployment of web caches or new content delivery nodes is advantageous, or to setup configuration parameters of the network. It is thus not a surprise that there exist in the market products for the measurement of QoE-related metrics targeting ISPs (e.g., Sandvine [11]). ISPs have a vital need to be aware of users' experience while using their networks, to anticipate complaints, take countermeasures, setup the network, and, ultimately, avoid the churning of customers.

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

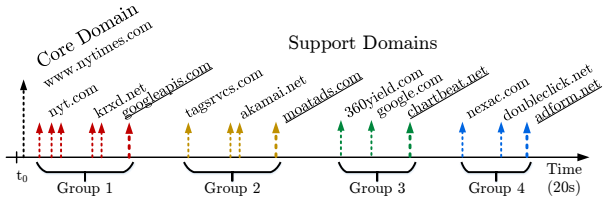


Figure 1: Flows in a *nytimes.com* visit. We use the time to contact support domains to monitor performance.

Users' QoE is is intrinsically subjective, thus hard to be fully assessed and quantified. Previous works [2, 5, 7] have proposed objective metrics that have been shown to be correlated with users' Mean Opinion Score (MOS), even if a model to predict MOS is still hard to get [4]. These metrics however either are computed at the server-side (i.e., available only to Content Providers) 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 to provide visibility into web performance are rare, and generally based on traffic payload to reconstruct web pages [11].

We introduce PAIN (PAssive INdicator), a system to monitor web page performance using passive traffic logs, as typically available in ISP networks. The deployment of encryption (e.g., HTTPS) makes solutions that reconstruct web sessions from payload [2, 5, 11] no longer effective. PAIN relies only on L4-level statistics (e.g., Netflow), annotated with DNS information [3] to compute a synthetic indicator of the web page rendering time.

The intuition behind PAIN is very simple. Once users reach a website, their browsers open many flows to different servers to fetch HTML objects, scripts and media content. We call the Fully Qualified Domain Name (FQDN) associated with the first contacted server the *Core Domain* and the remaining contacted FQDNs the *Support Domains*. An example is provided in Fig. 1, which illustrates with colored arrows the moment in which flows to support domains appear after a visit to the core domain *www.nytimes.com*. Given core domains of interest, PAIN automatically learns contacted support domains, as well as the typical order in which such flows appear in the network, creating *groups of support domains*. In the example, PAIN learns 4 groups from the observed network traffic. PAIN then considers the delay to observe flows of each group a performance indicator. It uses visits to the website from all users to (i) observe probable patterns; (ii) identify checkpoints that model the download process; and (iii) measure the delay to pass checkpoints, i.e., automatically building a benchmark.

We validate PAIN in a controlled environment, in which a testbed is instrumented to browse web pages and collect classic performance metrics. We compare measurements collected by the two independent approaches. We show that PAIN is able to spot changes in network conditions, reporting quality degradation when the page load time increases. PAIN metrics are strongly correlated with objective metrics computed based on client instrumentation.

2 THE PAIN SYSTEM

PAIN is an unsupervised system composed by two blocks (see Fig. 2). 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 periodically updated (e.g., monthly), to cope with changes in web-page structure. The *Metric Extraction* module extracts the actual performance metrics using the previously built models. All algorithms are $O(n)$ with respect to the input size (i.e., number of flow records) and are implemented in Apache Spark for scalable processing.

2.1 Input data

PAIN expects two input:

(i) Core domains: It is a user-defined input containing core domains the ISP is interested in monitoring, such as popular websites accessed by users of the network, or generic lists (e.g., Alexa). PAIN operates with up to L4-level measurements (e.g., source and destination IP addresses as well as TCP port numbers), and thus ISPs must specify only the domains to be monitored, and not full URLs. This allows PAIN to work with encrypted traffic, where domain names are still visible;

(ii) Flow records annotated with DNS information: they are captured in the network, by means of flow exporters. PAIN expects ordinary flow records (i.e., the usual 5-tuple), enriched with the domain names used by clients to contact servers. Different methodologies can be used to annotate flow records with that information, such as the methodology presented in [3], which leverages DNS traffic.

2.2 Model learning

The Learning Module models the arrival of flows opened by the browser to render web pages. The first task is to learn the support domains associated with each core domain. PAIN learns that by focusing on the flows *commonly* occurring after core domains. Then, given that flows to download objects while rendering a page vary from visit to visit (e.g., because of caching, persistent connections, modification in the content, etc.), PAIN analyzes the order in which groups of support domains *typically* appear.

2.2.1 Support domains learning: PAIN learns support domains based on the methodology we introduced in [13]. It observes all flows generated by a (possibly large) population of clients that visits the target core domain. We assume browsers access support domains immediately after loading the main HTML object hosted in the core domain. PAIN analyzes the training dataset on a per-client basis, processing flows in chronological order according to starting time. Then, every time a flow to a core domain is observed, PAIN opens an *observation window* with duration ΔT.

For each visit to core domain C, PAIN registers all domains contacted by the client during the observation window, forming a list of candidate support domains ($CAND^C$). If a support domain is seen multiple times within a window, only the first occurrence is registered.[2] Note that background traffic may be present in observation windows, whose flows introduce noise in the learning process. After the traffic of a number of clients is processed, PAIN evaluates the candidate domains to form the final set of support domains SD^C. It uses the frequency $f_{d,C}$ in which the domain d appears in observation windows of C. The assumption is that actual SD^C will consistently appear in multiple observation windows, whereas domains related to background traffic, being present by chance, can be filtered out. Candidate domains appearing less frequently than a threshold $MinFreq$ are discarded. The final SD^C is thus given by:

$$SD^C = \{(d, f_{d,C}) | d \in CAND^C \wedge f_{d,C} > MinFreq\} \quad (1)$$

2.2.2 Support domain scores: Intuitively, the time support flows appear in the network impacts the speed in which a web page is rendered. Page elements hosted in third-party sites (e.g., images and advertisements) are usually requested after other components of the page (e.g., scripts) are processed. As for `onLoad` time and `SpeedIndex` (see Sec. 4), PAIN leverages this intuition to calculate a score for $d_i \in SD^C$. The score is higher for those domains d_i appearing further away from the core domain C, in time. That is, PAIN identifies which support domains will serve as *checkpoints* based on their delays relative to the core domain flow.

However, SD^C is constructed from many observation windows, which are noise by definition. For instance, not all support domains appear in every observation window due to caching, persistent connections, etc. Equally, nothing prevents browsers or mobile terminals from opening flows to third-parties in a different order while rendering pages.

To determine the score for each $d_i \in SD^C$, PAIN computes a *dependency matrix* $\mathcal{M}^C$ of order $|SD^C|$. Each cell $\mathcal{M}^C_{i,j}$ represents the number of observations windows OW in which d_i has appeared *after* d_j in time. Note that $\mathcal{M}^C_{i,i} = 0$ and $\mathcal{M}^C_{i,j} = |OW|$ only if d_i appears always after d_j, and both d_i and d_j appear in all observation windows. The score of $d_i \in SD^C$ is calculated as $S^C_{d_i} = \sum_j \mathcal{M}^C_{i,j}$, i.e., d_i has high score if it appears often later than other domains.

2.2.3 Checkpoints: After ranking $d_i \in SD^C$ using $S^C_{d_i}$, PAIN identifies the checkpoints. We have tested several options, such as considering specific support domains as "landmarks" to calculate performance metrics. However, due to noise in observation windows (e.g., missing support domains), this results in a complicated choice. We propose a very simple rule that considers *groups* of support domains, which gives reliable outcomes based on our tests. The intuition is that, by clustering the support domains in some *few* groups, we filter out the noise caused by missing support domains.

More precisely, we sort $d_i \in SD^C$ based on scores $S^C_{d_i}$ and split the domains in n groups $G^C = \{G^C_1, \ldots, G^C_n\}$, where groups have at least $|G^C_k| = \left\lfloor \frac{|SD^C|}{n} \right\rfloor$ support domains. G^C_1 will contain the support domains that often appear the closest to the core domain

[2] This step avoids inflating the number of flows due to browsers pre-provisioning multiple TCP connections to the same server.

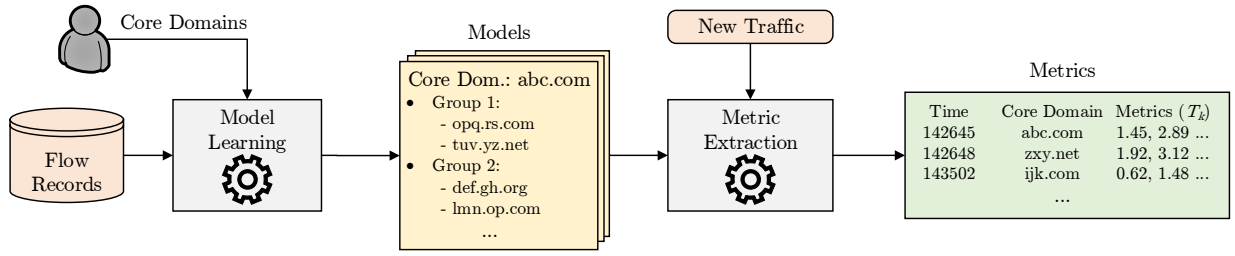


Figure 2: Architecture of PAIN. It learns and clusters support domains using flow records and a list of target core domains. The resulting groups are used to extract indicators of the website performance.

flow, wheres G_n^C will have the support domains that often appear the furthest to the core domain.

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

2.3 Performance metric extraction

The metric extraction module analyzes new traffic to provide the performance metrics. Intuitively, we measure the time at which flows in each group are observed. Visits to a group are considered completed when the last flow in the group is observed. This last visit is a *checkpoint*, and the relative visit time to checkpoints are recorded as performance metrics.

Like in the training phase, PAIN analyzes the traffic flows on a per-client basis, chronologically by starting time. When it encounters a flow to a core domain C, it open an *evaluation window* (EW) ΔT long. PAIN considers then all flows generated by the same client within the EW.

PAIN processes EW and extracts the *maximum* arrival time of support domains within each group G_k^C. The k-th checkpoint time T_k is computed as

$$T_k = \max_{s \in G_k^C} t_s - t_c \tag{2}$$

where t_x is the time of flow to domain x, s and c being the support and core domains, respectively.[3]

We tested different criteria in place of maximum (e.g., average and mean) and all lead to worse results; the intuition is that the web page performance is mainly driven by the ability of the browser to obtain all objects to render the page, which in turn depends on the time the last flow is observed on the network. The use of maximum also highlights possible degradation or failures of specific servers involved in serving the content.

The tuple $T^C = \{T_1, \ldots, T_n\}$ represents then performance measurements for a given visit to the core domain C. By considering all visits from all clients to C, PAIN builds a statistics on the performance faced by clients. Indeed, due to the intrinsic noisiness of flow level measurements, PAIN output assumes relevance when multiple measures are aggregated to contrast different users, time periods or locations.

[3]Note that groups can be absent if none of its support domains is in EW.

2.4 Design decisions and caveats

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

Other designs would be possible too, such as by using supervised algorithms. The system could learn the model from the network traffic guided by client-side metrics, e.g., by varying the number of checkpoints and the size of observation windows so to approximate client-side metrics in the best possible way. Such a supervised design would result in a system that adapts to different websites, thus leading to fine-grained groups of support domains for each core domain. It however requires ground truth data captured at client-side for each core domain of interest.

We have investigated the supervised design, and improvements are indeed possible for particular websites. The deployment of this alternative, however, requires a resource-consuming testbed to be deployed with PAIN, in which training should be performed, periodically, for each monitored websites, with the support of in-browser instrumentation. We have eventually decided to follow the unsupervised approach, since it broaden the PAIN deployability and dramatically enhances training scalability.

3 TESTBED AND DATASETS

3.1 Testbed

We evaluate PAIN using synthetic traces produced in a testbed. Our goal is to compare the output of PAIN to objective metrics collected directly in the browser. For this, we instrument a desktop PC with the *WebPageTest,*[4] a tool set for web performance assessment. Given a list of URLs, it automatically navigates through each page in a controlled environment while saving detailed statistics. Many options are available, choosing 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 (i) emulate users'

[4] WebPageTest can be downloaded from https://www.webpagetest.org/. WebPageTest emulates networks based on DummyNet, a live network emulation tool that can be found in http://info.iet.unipi.it/~luigi/dummynet/.

Table 1: 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

browsing considering realistic clients and network conditions and (ii) export metrics related to the page rendering process.

WebPageTest exports the HTTP Archive (HAR)[5] file for each visited page. It contains overall information about the visit as well as statistics for the each object: from HTTP-headers, to network-level statistics that describe the TCP connections opened to download objects. In particular, it records the time in which the TCP connection starts, and server domain name associated with it, which we use to train and test PAIN, i.e., the information that would come from flow exporters in real deployments.

Additionally, WebPageTest saves many objective metrics related to QoE. Here, we consider two popular metrics:

(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 parsed;

(ii) **SpeedIndex**: Proposed by Google[6], represents the time at which visible portions of the page are displayed. It is computed by capturing the video of the browser screen, and tracking the visual progress of the page during rendering.

The testbed is connected via a 1 Gbps Ethernet cable to the Politecnico di Torino campus network.

3.2 Datasets

We build two datasets to validate PAIN, with respectively *typical* and *artificial* network conditions.

The *typical* dataset is built by letting WebPageTest visit 10 popular domains in Italy including, e.g., news, e-commerce and weather services. For each domain, WebPageTest downloads the homepage and 9 internal pages (e.g., arbitrary articles from the news websites), for a total of 100 URLs.

Since PAIN must work seamlessly regardless of client configurations, we consider 4 different client browser and device combinations, which we summarize in Tab. 1. 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.

We further consider 8 network technologies, which are summarized in Tab. 2. The configurations are created by WebPageTest by imposing traffic shaping policies that mimic actual parameters of the technologies. For the *Native* case, WebPageTest imposes no shaping – i.e., the 1 Gbps Ethernet network connecting the testbed is used without changes. For other cases, it uses DummyNet as emulation environment to enforce typical bandwidth and Round Trip Time (RTT) faced by end-users contracting the technologies.

[5]http://www.softwareishard.com/blog/har-12-spec
[6]https://developers.google.com/speed/docs/insights/about

Table 2: Network settings in the *typical* dataset. *Native* corresponds to a scenario with no traffic shaping (i.e., 1 Gbps Ethernet).

Name	Down Link	Up Link	RTT
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

At last, we visit each page twice for each setup: (i) with empty browser cache; and (ii) few seconds later for profiting from the local cache. The resulting traffic is expected to vary strongly, since many objects are in the cache 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 represents *artificial* network conditions, in which the *Native* case is degraded controlling (separately) additional link delay or bandwidth limit on the testbed. That is, we simulate scenarios in which users see increases in page load time caused by network worsening. We simulate 10 cases: (i) adding from 100 ms to 500 ms extra delay and (ii) imposing a limit from 5 Mbit/s down to 312.5 kbit/s on bandwidth. Again, we visited each page twice (with and without caching) and with 4 client browsers. WebPageTest has performed 8 000 extra visits for building this second dataset (taking ≈60 h).

Note that client capabilities could also impact performance of websites – e.g., slow clients will take longer to parse and render web pages. Testing PAIN in such scenarios is left for future work.

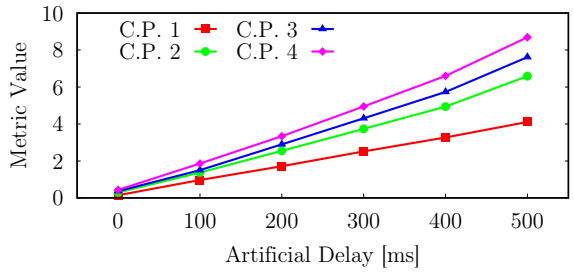
4 EXPERIMENTAL RESULTS

4.1 Parameter tuning and checkpoints

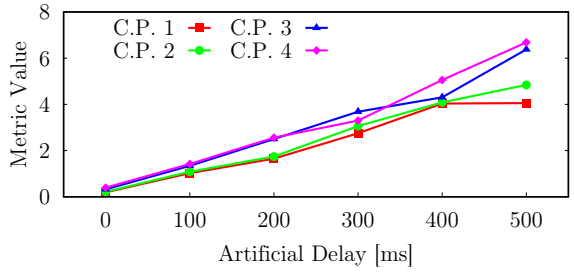
We evaluate PAIN checkpoint times, aiming at (i) getting an initial feeling about checkpoints' behaviors; and (ii) investigating PAIN parameter settings. Remind that PAIN output is a set of checkpoint times (T_k) for each website visit.

We first tune the observation window duration (ΔT) and the threshold to filter out support domains ($MinFreq$) – see Sec. 2.2.1. We note that PAIN is not very sensitive to the parameters. Given that support domains are grouped, and each group is used to extract a single checkpoint, PAIN output remains mostly unaffected if some support domains are not associated to the respective core domain. Following a validation methodology similar to our previous work [13], i.e., measuring the number of support domains that is correctly associated with each core domain for various parameter settings, we conclude that the best settings for PAIN are $\Delta T = 30s$ and $MinFreq = 25\%$.

We next perform a sensitive analysis of n, i.e., the number groups. We vary n, recording the checkpoint times. Trivially, changing n affects values for checkpoints. However, remind that we always



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



(b) ***www.subito.it***

Figure 3: Median time to pass checkpoints when artificially increasing access link delay.

take the arrival time of the last flow in each group as a checkpoint. As such, the last checkpoint is always the same, regardless of n. Moreover, considering checkpoints representing support domains that are usually contacted long time after the core visit, we observe only minor changes for different n. We consider $n = 4$ for the remaining experiments, with other values leading to very similar results when using the last groups as performance metric.

Fig. 3 illustrates checkpoint values when $n = 4$ for two popular websites and considering the *artificial* dataset for increasing artificial delay. We consider the median checkpoint time over all tests with each delay value. As expected, checkpoints increase alongside the delay, starting from around 0.5 s and up to almost 10 s when RTT is 500 ms. That is, checkpoints reflect the network conditions and increase in case of degradation. Note that the not-deterministic visiting order of support domains is reflected in checkpoint times, which may be inverted. For repubblica.it this does not happen, but for subito.it sometimes checkpoint 4 completes before checkpoint 3, due some flow in the latter group being delayed, or to objects in checkpoint 4 being locally cached.

We take the 4th checkpoint as benchmark for comparisons with other metrics in the remaining sections; very similar results are obtained considering the 2nd or the 3rd checkpoints, although if using the 1st checkpoint figures would be considerably noisier – i.e., it leads to higher variance in the metric.

4.2 Impact of network conditions

We now check whether PAIN is generally able to reflect worsening network conditions. Fig. 4 compares PAIN 4th checkpoint to SpeedIndex and onLoad time. A single website is reported (similar figures are obtained for other cases). Each line represents the median value for all website page visits with the given network conditions. 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 PAIN metrics. Thus, with these figures we aim at understanding whether the metrics present similar rate of variation given changes in the network conditions – i.e., whether their values are similarly affected by network impairments.

Focusing on Fig. 4a which compares metrics in the scenario discussed in Fig. 3, notice how the three metrics grow almost linearly with the RTT. The rate of variation in PAIN, seen in the blue lines, is in between the rate seen for SpeedIndex (green) and onLoad (red) time. When artificially varying the bandwidth in the *artificial* scenario (Fig. 4b), the values of the checkpoint change similarly to the rate observed for onLoad time, but faster than SpeedIndex. That is, PAIN is more sensitive to deterioration on the available capacity. Yet, results show that the checkpoint is directly related to page load time. 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, Fig. 4c reports the values for *typical* network scenarios. Again, we see similar patterns among the metrics, with the rate of variation of our checkpoints in between the other metrics.

In summary, all three metrics increase when the network degrades, demonstrating PAIN can highlight deterioration in page load time. Absolute ranges are different, but all reflects degradation in quality.

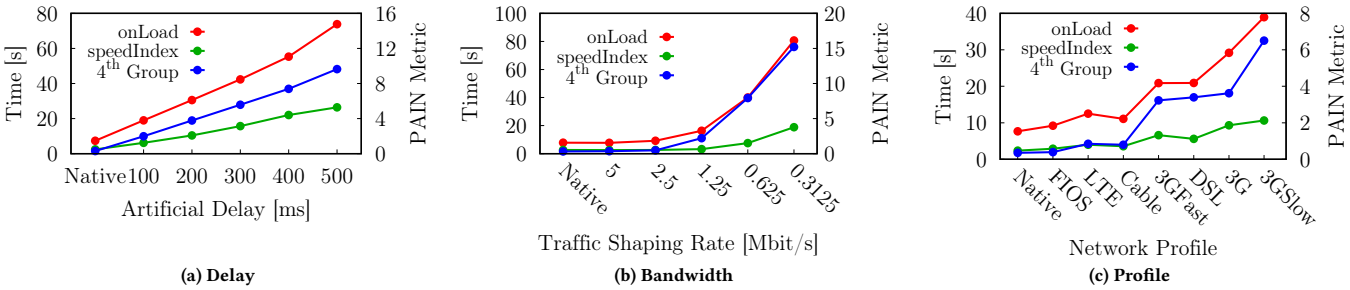
4.3 Correlation with client-side metrics

We formally quantify the correlation of PAIN 4th checkpoint with SpeedIndex and OnLoad time. Fig. 5 shows Spearman's rank correlation coefficients for onLoad time (blue) and SpeedIndex (red). Only results for the *typical* dataset are depicted, with the *artificial* one leading to similar conclusions. All 6 400 visits are used to calculate the coefficients, using separate bars for different websites.

Correlation coefficients are positive and very high (i.e., $\geq$ 0.5), ranging from 0.54 for *www.ebay.it* (SpeedIndex) to 0.90 for *www.gazzetta.it* (onLoad). Most values are close to 0.8 for both metrics. That is, PAIN checkpoints are strongly correlated with both objective metrics for different sites. For comparison, the correlation between SpeedIndex and onLoad ranges in [0.71,0.91]. Results reinforce that PAIN works well as a proxy to quality monitoring, providing strong indications without client-side instrumentation.

5 RELATED WORK

Previous works have focused on estimating QoE. Authors of [7] show that indirect metrics serve as indicators of users' MOS. Considering website performance, metrics such as the onLoad time or SpeedIndex have been shown to be correlated with QoE metrics [5]. Authors of [5] also propose the ByteIndex – a metric based on the bytes delivered to the client to render a page. All those metrics are however computed at the client-side – e.g., the ByteIndex requires



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

Figure 4: *www.repubblica.it* normalized `onLoad`, `SpeedIndex` and 4th checkpoint for various setups.

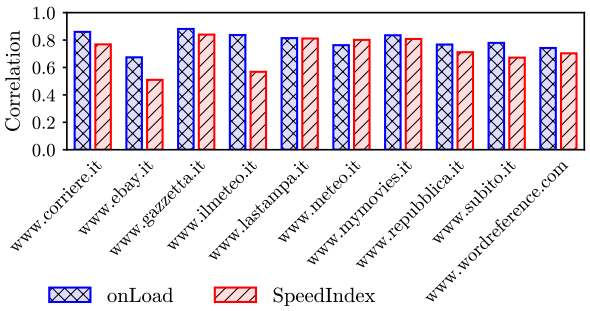


Figure 5: Spearman's correlation of 4th checkpoint with `onLoad` and `SpeedIndex` for analyzed websites.

knowledge of page elements, which is not available with encrypted traffic. PAIN instead targets ISPs, thus observing information from the network side, via passive measurements.

Past works targeting the ISP scenario either require DPI [2, 11, 12], or rely on ground truth from client browsers to train machine learning classifiers [1, 8]. Training is per-website, and becomes cumbersome to keep pace with website evolution. In contrast, PAIN uses a completely unsupervised approach. PAIN is a step further towards QoE indicators [6, 10] able to operate in a world where encryption is the norm [9]. PAIN automatically builds the models directly from flow-level traces, with no need to access to payload.

6 CONCLUSIONS

We presented PAIN, an automatic and unsupervised system to monitor website performance using flow-level measurements. It builds a behavioral model for 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. Finally, we showed that PAIN's metrics are strongly correlated with well-known objective metrics used as indication of users' QoE, such as the `onLoad` time and the `SpeedIndex`. We are currently working on deploying PAIN in an ISP network to collect real-world data.

ACKNOWLEDGEMENTS

The research leading to these results has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT15-129, BigDAMA.

REFERENCES

[1] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan. Prometheus: Toward Quality-of-experience Estimation for Mobile Apps from Passive Network Measurements. In *Proc. of the HotMobile*, pages 18:1–18:6, 2014.
[2] A. Balachandran, V. Aggarwal, E. Halepovic, J. Pang, S. Seshan, S. Venkataraman, and H. Yan. Modeling Web Quality-of-experience on Cellular Networks. In *Proc. of the MobiCom*, pages 213–224, 2014.
[3] I. Bermudez, M. Mellia, M. M. Munafò, R. Keralapura, and A. Nucci. DNS to the Rescue: Discerning Content and Services in a Tangled Web. In *Proc. of the IMC*, pages 413–426, 2012.
[4] E. Bocchi, L. D. Cicco, M. Mellia, and D. Rossi. The Web, the Users, and the MOS: Influence of HTTP/2 on User Experience. In *Proc. of the PAM*, pages 47–59, 2017.
[5] E. Bocchi, L. D. Cicco, and D. Rossi. Measuring the Quality of Experience of Web Users. In *Proc. of the Internet-QoE*, pages 37–42, 2016.
[6] P. Casas, B. Gardlo, R. Schatz, and M. Mellia. An Educated Guess on QoE in Operational Networks through Large-Scale Measurements. In *Proc. of the Internet-QoE*, pages 1–6, 2016.
[7] P. Casas, M. Seufert, F. Wamser, B. Gardlo, A. Sackl, and R. Schatz. Next to You: Monitoring Quality of Experience in Cellular Networks From the End-Devices. *IEEE Trans. Netw. Service Manag.*, 13(2):181–196, 2016.
[8] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki. Measuring Video QoE from Encrypted Traffic. In *Proc. of the IMC*, pages 513–526, 2016.
[9] R. Gonzalez, C. Soriente, and N. Laoutaris. User Profiling in the Time of HTTPS. In *Proc. of the IMC*, pages 373–379, 2016.
[10] W. Pan, G. Cheng, H. Wu, and Y. Tang. Towards QoE Assessment of Encrypted YouTube Adaptive Video Streaming in Mobile Networks. In *Proc. of the IWQoS*, pages 1–6, 2016.
[11] Sandvine. Measuring Web Browsing Quality of Experience. 2017. https://www.sandvine.com/technology/web-browsing-quality-of-experience.html.
[12] M. Trevisan, I. Drago, and M. Mellia. Impact of Access Speed on Adaptive Video Streaming Quality: A Passive Perspective. In *Proc. of the Internet-QoE*, pages 7–12, 2016.
[13] M. Trevisan, I. Drago, M. Mellia, H. H. Song, and M. Baldi. WHAT: A Big Data Approach for Accounting of Modern Web Services. In *Proc. of the BigData*, pages 2740–2745, 2016.