Revealing QoE of Web Users from Encrypted Network Traffic

Alexis Huet*, Antoine Saverimoutou†, Zied Ben Houidi*,
Hao Shi*, Shengming Cai*, Jinchun Xu*, Bertrand Mathieu†, Dario Rossi*
**Huawei Technologies Co. Ltd.* – {alexis.huet, zied.ben.houidi, shihao19, caishengming, xujinchun, dario.rossi}@huawei.com
†*Orange Labs.* – {antoine.saverimoutou, bertrand2.mathieu}@orange.com

***Abstract*—Internet Service Providers (ISPs) have a lot to gain from estimating the web browsing quality of their customers. However, unlike Content Providers who can easily access in-browser computed application-level metrics to estimate web browsing quality, ISPs come short mainly because of traffic encryption. In this paper, we use exact methods and machine learning to estimate well-known application-level web browsing QoS metrics (such as SpeedIndex and Page Load Time) from raw encrypted streams of network traffic. Particularly, we present and open-source a unique dataset targeting a large set of popular pages (Alexa top-500), from probes from several ISPs networks, browsers software (Chrome, Firefox) and viewport combinations, for over 200,000 experiments. Our results show our models to be accurate, and we particularly focus on their ability to generalize to previously unseen conditions, giving guidance concerning their retraining.**

I. Introduction

With the generalization of encryption [1], [2], ISPs are struggling to get insights from data flowing within their networks. When it comes to the web, encryption forced ISPs to give up on services like transparent caching and redundancy elimination. Today, it is also threatening their ability to understand web traffic, not to talk about estimating the quality of experience (QoE) their users witness when accessing it. Nowadays, when a user complains that browsing is slow and sluggish, an ISP has no objective way to verify it. Besides, human operators doing troubleshooting for the ISP can barely tell if the problem originates from the application, the user home-network, the transport network or because of a change in one of the many content delivery networks that contribute today to the delivery of a single web page [3].

Although directly collecting subjective web QoE is expensive [4]–[6], it is possible to measure objective *Application-level web Quality of Service* (AppQoS) metrics as a proxy of user QoE. The scientific community has come up with a plethora of such metrics in the recent years, each measuring a different aspect of the page load process. Some metrics for example are time-based [7] and measure the time of particular events like the Time To the First Byte (TTFB), the Document Object Model load time (DOM), the Above The Fold time (ATF) [8] or the entire Page Load Time (PLT). Others are integral-based and measure the visual progression speed like the SpeedIndex (SI) [9] or the object or byte progression speed like ObjectIndex (OI) or ByteIndex (BI) [10]. Nonetheless, computing these indicators needs access to application or at least HTTP-level information – an information that the ISP has not anymore access to in the nowadays widespread case of TLS encryption.

In this paper, we tackle the problem of AppQoS metrics estimation from streams of encrypted network traffic in the case of a single session. In particular, we estimate a set of popular metrics collected at the application-level from Timing APIs (DOM, PLT) or javascript code (SpeedIndex and variants). Among these indicators, the ByteIndex (BI) serves as a bridge between network traffic and AppQoS metrics. It is defined as the cumulative progression in time of the downloaded objects sizes in bytes. This metric has been shown to have high correlation with more complex visual metrics such as the SpeedIndex (SI) [10]. Intuitively, information similar and more fine-grained than the application-level BI is still present in encrypted network data at the packet-level: by simply tracking the packet size (or total bytes downloaded every ΔT ms), the cumulative byte progression of a web session can be measured at the network-level (net-BI). Machine learning techniques can then be additionally leveraged to devise estimators of the remaining AppQoS metrics, using time series of byte progression as input.

To do so, we collect a large dataset of web page load controlled experiments in which we measure both the *application view*, from which we gather the groundtruth AppQoS metrics, as well as the *network view*, i.e., packet-level traces. When constructing the datasets, for the sake of generalization, we collect experiments pertaining to a large variety of conditions. We then evaluate the accuracy of our estimators compared to groundtruth AppQoS metrics and study how they generalize to different conditions (different types of pages as well as different network conditions). Our contributions are as follows:

- First, we define exact (net-BI) or data-driven models of AppQoS metrics that are able to operate directly on encrypted network traffic.
- Second, we show that net-BI approximates well the application-level app-BI (6% median relative error, further down to 3% using machine learning models) and further that any other AppQoS metrics can be accurately learned from encrypted packets (less than 14% median

ISBN 978-3-903176-28-7© 2020 IFIP

error across all metrics).

- Finally, we systematically study the transferability of the data-driven models to previously unseen conditions at multiple levels (from browsers, to environment, to viewport settings, location, browser family and version, to L7 and L3 protocols, network conditions, etc): this not only quantifies the expected degradation, but also provides a ranking of the generalization capabilities, so to prioritize retraining along the most varying aspects, and thus carefully plan measurement campaigns.

The remainder of the paper is organized as follows. Sec. II introduces various AppQoS metrics and related work. Sec. III formulates the problem, overviewing our methodology and datasets. Sec. IV describes exact and data-driven models, whose accuracy and ability to generalize is evaluated in Sec. V. Finally, Sec. VI and Sec. VII respectively discusses and summarizes our findings.

II. Background and related work

AppQoS metrics. Web Quality of Experience is generally assessed with the browser using objective application-level metrics (AppQoS metrics). The most prominent metric used in the industry is Page Load Time (PLT) [11], corresponding to the time for loading the page contents from the start of the navigation. In recent years though, additional metrics have been devised: several ones capture a precise event during the page rendering process, such as Time To First Byte (TTFB) that is the time at which the first byte of payload is received, Document Object Model (DOM) time that corresponds to the time for loading the DOM content, Above The Fold (ATF) defined as the time for the browser to entirely render the viewport area (Visually Complete in [8], Time for Full Visual Rendering in [12]). These metrics are generally accessible by W3C APIs such as Navigation Timing [7]. Another kind of metrics, referred as time-integral metrics [10], exploit the whole waterfall of page loading. The intention is to better represent actual user experience and to improve robustness. The best-known metric is SpeedIndex [9] (SI), based on the visual progress of the page rendering process. ByteIndex [10] (BI) is an approximation that replaces the visual rendering progress by the byte progress, where bytes come from the size of objects requested by the browser. Other metrics require the collection of position and area of the different elements on the page, such as RUM SpeedIndex [13] (RSI) and ReadyIndex [14]. *In this work, we propose a method to systematically learn any of these metrics, from encrypted traffic.*

Measuring web performance in the wild. Recent works have focused on measuring AppQoS metrics in the wild based on browser-based active experiments [15]–[19]. In particular, the authors of [15] performed a six-months long measurement campaign to study how known AppQoS metrics vary depending on various conditions. Similarly, [18] contains a data set of 2 million visits from mobile networks of four different countries with measure of TTFP, PLT and RSI for 7 pages, whereas [16] includes eight websites for a duration of two weeks and [17] leverages 3 websites during 3.5 years. While very valuable, however each study focuses on a limited subset of the most important aspects that can affect AppQoS performance – *whereas heterogeneity of the experimental conditions is a particularly important aspect of our work.*

User Experience. All the AppQoS metrics represent slightly different attempts at quantifying the loading speed of the page, with the assumption that the quicker the content is loaded, the higher the user QoE, which has been ascertained by studies involving real users (see [5], [20]–[22] and references therein). We remark that correlations exist between different AppQoS metrics pairs: for instance, observed correlation [10] between SI and PLT is 0.79, whereas correlation between SI and BI reaches 0.82. This suggests monitoring of any objective AppQoS metrics relevant as a proxy of user experience.

It should be noted that while monitoring the aforementioned AppQoS metrics is straightforward for content providers (which have direct access to the application side, see e.g., at [23]) it has become impossible for ISPs (which only have access to the network packets). Indeed, a decade ago encryption was hardly the norm, and it was possible for ISPs to leverage HTTP logs to develop (rather complex) algorithms to detect the different page loading events [3]. In recent years however, TLS encryption has become pervasive, preventing ISPs accessing payload content, practically defeating valuable approaches similar to [3] and leaving ISPs without tools – *such as those that this work provides to fill this gap.*

Measuring web performance from the dark. To date, with few exceptions [24], [25], work in the literature assumes to have access to HTTPS header information [5], [10], [11], [13], [14], [20]–[22], which makes it unfit to be deployed by ISPs, which is the issue we address in this paper. Under this light, closest to our work is [24], that deals with (i) isolating a single web session from the network stream, containing potentially background traffic and several concurrent web page sessions as well as (ii) defining indicators of web browsing quality from passive measurement. Particularly, authors define a passive web performance indicator (as well as a supervised indicator called BestCheckpoint) which correlates with basic Page Load Time (PLT) metrics. Finally, a preliminary version of our effort was briefly demonstrated at [25] – *In this work, we systematically analyze the accuracy of our approach and the portability of the models across a very wide range of conditions.*

III. Methodology and dataset

We provide a brief overview of the methodology we use in this paper with the help of Fig. 1. We automate execution of data collection by instrumenting several workers (either public or private instance) based on two tools, namely Web Page Test (WPT) [27] and Web View (WV) [19]. Both tools have access to application layer information and can thus provide us with a groundtruth for the metrics we target to estimate. Web View is a user-oriented measurement tool whose main

TABLE I
SUMMARY OF THE DATASETS IN THIS WORK (WPT, WV) VS RELATED WORK. DATASETS WILL BE OPEN SOURCED AT [26].

Factor	Web Page Test (WPT)	Web View (WV)	Literature
Locations	{Asia, EU, US} × {Public, Private}	EU	Generally in EU [15], [18] EU, US and others in [17]
Target pages	Top-500 × {world, China} × {main, subpages}	50 main pages from Top-500	Limited (3-20) in [16]–[18] Up to Top-10K main pages in [15]
Network conditions	50% native + 50% (Fiber, DSL, 4G, 3GFast, ...)	Fiber, ADSL	Ethernet, mobile broadband [18]
Browsers	Chrome	Chrome, Firefox	Mainly Chrome only [6], [16] Chrome and Firefox in [15], [18]
Viewports	Default	1920×1080, 1440×900	Default or not specified [14], [16] Five viewports in [15]
Protocols	{HTTP/2} × {IPv4}	{HTTP/2, QUIC} × {IPv4, IPv6}	From HTTP/1 only [17] to {HTTP/2, QUIC}× {IPv4} [15], [18]
Metrics	BI, PLT, SI, OI, DOM	BI, PLT, RSI, OI, DOM	PLT, RSI, SI, ATF [15], [18]
Samples no.	119,395	111,171	

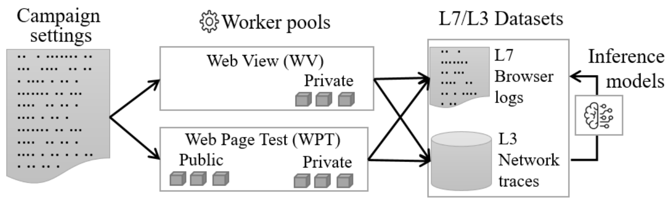


Fig. 1. Synoptic of the workflow in this paper

objective is to perform automated web browsing sessions, measuring representative information of web pages in order to better qualify and understand web browsing, both in terms of performance and delivery. The probes emulate end-users' web browsing within a real end-user environment: real web browsers, residential access network, etc. Each worker is tasked with execution of multiple web sessions, consisting of opening a browser under specific conditions and loading a web page until the end of network activity. For each session, we collect simultaneously (i) application level information from the browser and (ii) raw network packet traces. The remainder of this section details the collected datasets.

Given an AppQoS metric of interest, we then build a labeled dataset from the collection of controlled experiments: each network packet trace is associated with the corresponding value of the AppQoS metric. This labeled dataset feeds a machine learning model, with as input a function of the encrypted packet trace, and as output the value of the AppQoS metric, as detailed in Section IV.

Once the model has been trained, we test its prediction ability on new samples. One of the main fallacies for a successful application of supervised machine learning in real network deployment is represented by the possible lack of generalization capabilities. In other words, the model works well on the collected dataset, but performance significantly degrade in other network settings. Since we are well aware of this fallacy, we particularly stress-test the generalization ability of our models in Section V.

A. Datasets

To stress test the model's generalization capabilities, we purposely collect two distinct datasets with two different tools, that we plan to release at [26]. We perform a large set of controlled experiments with both WV and WPT, varying a number of relevant parameters and conditions, for a total of 200K+ web sessions, roughly equally split among WV and WPT. Tab. I gives an overview of our measurement campaign put in perspective with the closest datasets in the literature in terms of scale: compared to existing ones, our dataset is unique in terms of geographical coverage, scale, diversity and representativeness (location, targets, protocol, browser, viewports, metrics).

About half of WPT experiments are performed using the online service www.webpagetest.org at different locations worldwide (55967 experiments in Europe, Asia, USA), the other half uses private WPT instances of WPT in three locations in China (63428 experiments in Beijing, Shanghai, Dongguan). The list of target URLs comprises the main pages and five random subpages from Alexa top-500 worldwide and China. We vary network conditions by leveraging WPT traffic shaping capabilities: half of the experiments use native WPT connections and the other half is apportioned among 4G, fiber, 3GFast, DSL, and custom shaping/loss conditions. The other elements in the configuration are fixed: Chrome browser on desktop with a fixed screen resolution, HTTP/2 protocol and IPv4.

Using the Web View (WV) platform, we further collect 111171 experiments from three machines located in France. Compared to the WPT experiments, we select two versions of two browser families (Chrome 75/77, Firefox 63/68), two screen sizes (1920x1080, 1440x900), and employ different browser configurations (one half of the experiments activate the AdBlock plugin) from two different access technologies (fiber and ADSL). From a protocol standpoint, we use both IPv4 and IPv6, with HTTP/2 and QUIC, and perform repeated experiments with cached objects/DNS. Given the settings diversity, we restrict the number of websites to about 50 among the Alexa top-500 websites, to ensure statistical relevance of the collected samples for each page.

We remark that prior work has focused each time only on specific aspects, such as L7 protocols HTTP/1 vs HTTP/2 [6] or QUIC vs HTTP/2 [15], [18]. Lower layer protocols (e.g. the impact of IPv6 that we consider in this work) have been disregarded to the best of our knowledge. Similarly, vantage points are typically spread in few locations in few continents [15], [18]. With few exceptions [15], [18], most related work focus on a single browser (typically Chrome), with the default viewport. In this work, we carefully balance all conditions to build datasets that contain variability in all the factors mentioned in Tab. I. By doing so, we are able not only to study the ability of machine learning models to forecast user QoE from encrypted traffic, but also to stress-test the generalization properties of such models to very heterogeneous conditions.

IV. Network-level inference of AppQoS metrics

In this section, we illustrate how to approximate in-browser AppQoS metrics directly from encrypted traffic. We first show how BI metrics (Sec. IV-A) can be computed from packet-level data, and then describe how to extend the inference to other AppQoS metrics (Sec. IV-C).

A. ByteIndex (BI) metrics

Whereas BI considers the object sizes as seen by the browser, net-BI instead takes the packet size as seen by the network. Is is useful to recall the ByteIndex (BI) definition: given a session consisting of loading a single web page, let $x(t)$ the percentage of bytes already retrieved at time t. By definition, $x(t)$ is a monotonic non-decreasing curve, going from 0 at the initial time $t = 0$ of the session to 1 after the time to last byte is retrieved. The BI is the area above the curve (and below 1), formally defined as $\text{BI} := \int_0^{+\infty}[1 - x(t)]dt$.

The numerical evaluation of this formula depends on the timescale used for populating $x(t)$. In the rest of this section, we use different time granularity, yielding to different sizes of individual events, that yield to (slightly) different BI metrics.

- **app-BI** Having access to HTTPS header, the original application-level BI [10], denoted as app-BI in this work, is computed at object arrival times t_i, using the size s_i of the received i-th object , so that $x(t)$ is incremented at t_i by $s_i/\sum_j s_j$, i.e., the proportion that object s_i represent of all objects $\sum_j s_j$ in the page.
- **net-BI** At the finest granularity, network-level BI can be computed from individual packets: whereas the formula to compute net-BI remains the same as in the previous case, now t_i and s_i correspond to the reception time and size of the i-th packet. Since the navigation start time is not available from the network side, we select the first DNS query time, as seen in the DNS record packets, as initial time of the session.
- **Time-aggregated net-BI** A more convenient granularity is to aggregate packets received during windows having fixed duration W. We consider $W = 100\,ms$ in this work, that was shown to provide accurate results in our preliminary work [25]. In this case, the size s_w cumulates the size of packets received during the window $\sum_{j\in w} s_j$, represented by the time t_w of the w-th aggregate (centered in the window, so $t_w = (w-1)W + W/2$).

B. Application vs Network views of web page rendering

For the sake of illustration, we visually compare in Fig. 2 the three different set of events for the same single session, from top to bottom: application BI, packet-level net-BI, and time-aggregated net-BI. Pictures on the left represent events by segments with different times and sizes. Pictures on the right cumulates events, depicting $x(t)$ curves with black lines and BI metrics with colored areas. To avoid cluttering the picture we do not attempt at differentiating all events (domains or flows), and only highlight the top contributor: the DNS domain contributing the most to app-BI is colored in pink (top picture), while the IP contributing the most to net-BI is colored in orange (middle and bottom). Dark-gray segments and areas correspond to the remaining domains/servers.

Several interesting remarks can be gathered from the picture. First, the time-series of events cannot be directly compared: few objects are loaded in the browser, that correspond to long sequences of packets. However, the cumulative curves look similar, and the areas of app-BI and net-BI directly annotated in the pictures give remarkably similar values in this example (a thorough and statistically relevant analysis is reported in Sec. V-A). Second, it is clear that aggregation of packets in windows or objects levels changes the nature of the events, but the cumulative curve and the BI values remain similar. It follows that time-aggregation is a simple method for reducing the amount of data to store in memory while keeping the same shape for $x(t)$, providing a lightweight yet accurate view of the traffic.

C. Learning AppQoS metrics

1) High level idea: The aggregated network view additionally provides a more homogeneous view of the traffic, that can be leveraged as input for learning various AppQoS metrics in addition to BI, such as those annotated as vertical lines Fig. 2. We also remark that, according to the definitions of app-BI and net-BI, we expect these alternative formulations to be similar, with small discrepancy even in degraded network conditions (this is confirmed in Sec. V-A). In addition, application-level BI is known to have high correlation [10] with the most important AppQoS metrics (such as SI, PLT, etc.). As such, it is reasonable to resort to supervised machine learning to learn the just illustrated similarities between application-layers and the network-level views.

In a nutshell, given any AppQoS metric of interest y available in our dataset, our methodology is to train a *supervised model* $y = f(x)$ from sessions of web browsing, of which application labels y and network features x are simultaneously gathered (recall Sec. III). To ensure that the models are portable (across types of web pages, different

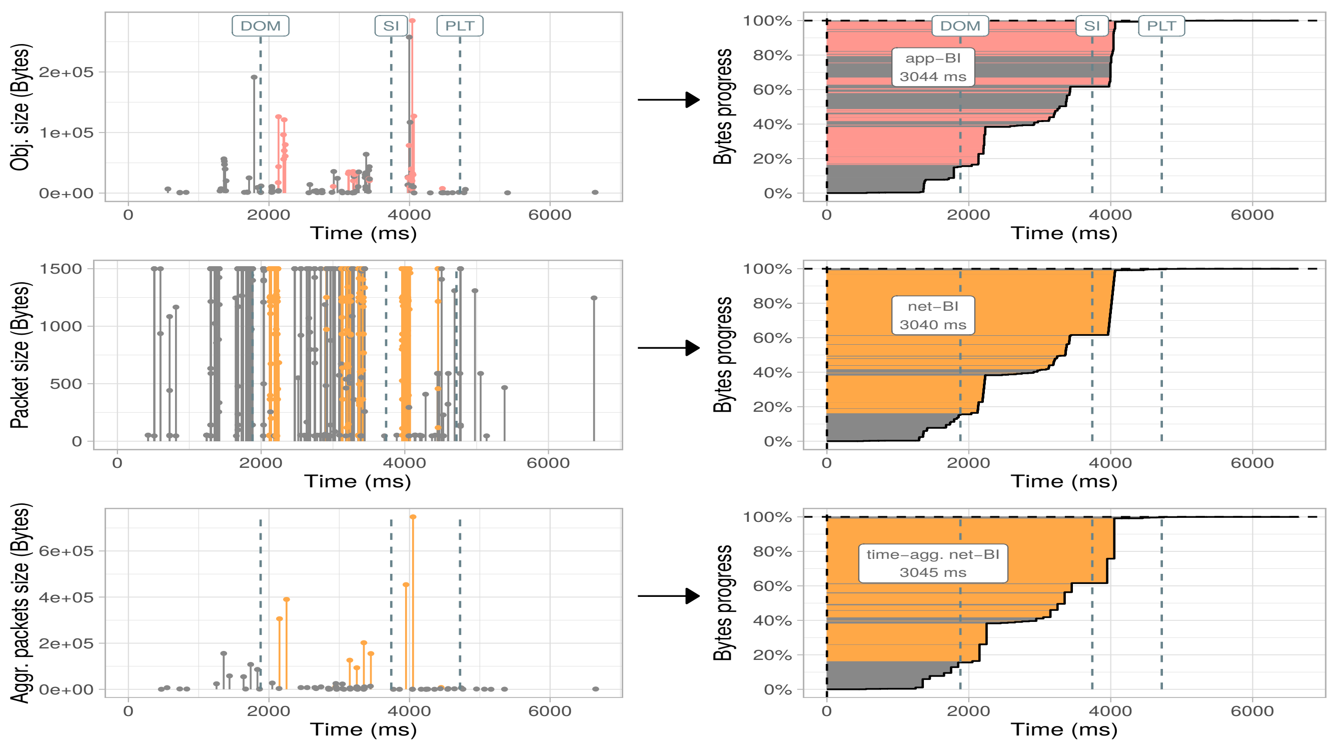


Fig. 2. Illustration of application (top) and network views (middle and bottom) for Byte Index metrics, i.e., the colored area on top of the $x(t)$ progress curve (black continuous line). Several other AppQoS metrics (DOM, SI, PLT) are annotated in the picture.

network conditions, etc.) we test our models in a wide range of conditions (recall Tab. I).

2) Inputs $\underline{x}$ vs Output y: Our methodology works with different sets of inputs gathered at either packet or flow level. For lack of space, in this paper we only discuss inputs based on the time-aggregated $x(t)$ shown in the bottom plot of Fig. 2. In particular, we sample $x(t)$ every 100ms from 0 to 10s. Then, we compute the net-BI value, which is the area above the curve $x(t)$. The input $\underline{x}$ is the fixed-size vector of length 101 consisting of appending the discretized time series (a vector of length 100) with net-BI (of length 1). For each output y (i.e., the PLT and SI metrics shown as vertical reference lines in Fig 2), we then train a specific model. We focus on a subset of important AppQoS metrics out of overall collected ones: ByteIndex (BI), SpeedIndex (SI), RUM SpeedIndex (RSI) and Page Load Time (PLT). Since results are qualitatively similar, this allows us to report results in a more concise way.

3) Regression $y = f(\underline{x})$: In this paper we only report results gathered with the LightGBM implementation of Gradient Boosting [28] implemented in scikit learn, and in which, we tune the hyperparameters using Bayesian optimization. A satisfactory set of hyperparameters is given by an ensemble of 1700 estimators where each tree has a depth of 7, and with at least 21 elements in each leaf. We point out that we obtained similar results with Random Forest and 1D-CNNs. At the same time, LightGBM training time is significantly faster, which is essential to the main goal in this paper: indeed, as we shall see in Sec. V-B, a thorough evaluation of the model transferability requires to *systematically retrain* the models to stress-test their generalization capability – which requires fast training methods.

V. Experimental Results

We now report results of AppQoS estimation from encrypted network traffic. We first provide a bird-eye view, summarizing the performance results for each of the metrics on the whole collection, and illustrate the main reasons behind the good performance (Section V-A). We next dig into the generalization capabilities at a deeper level, providing operational suggestions concerning the need for retraining along the different dimensions (i.e., pages, browsers, protocols, etc.) in our dataset (Section V-B).

A. Bird-eye view: Results at a glance

We start by quantifying the accuracy of machine learning models in inferring the range of AppQoS metrics in our WPT and WV datasets. As typically done in the literature, we split the whole set of experiments into five equally-sized subsets to perform *5-fold cross validation*. Fig. 3 depicts the AppQoS metric estimation error in both *absolute* (bars) and *relative* (annotated percentage over the bar) terms, with metrics ranked on the x-axis by increasing amount of median error. Bars report the median error, and error bars extend to the 1st and 3rd quartile of the error distribution: it can be seen that the median error is below 300ms (14%) for all metrics, including SpeedIndex (SI computed on WPT) and RUM SpeedIndex (RSI computed on WV).

Additionally, notice that for one metric (app-BI), the plot reports the error when its estimation is carried out through

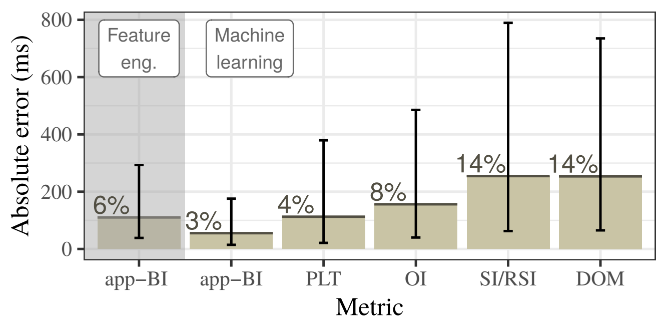
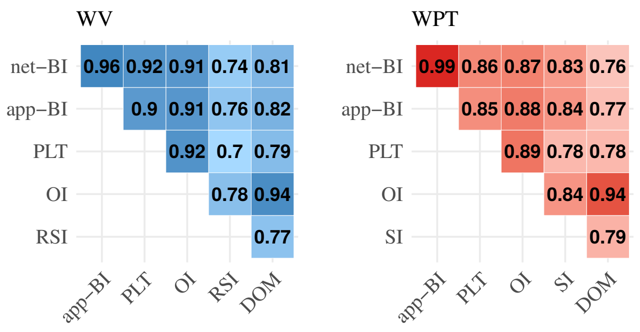


Fig. 3. Median absolute and relative errors (bar plot) with first and third quartile (error segments) for predicting five AppQoS metrics in the WV and WPT datasets.


WV
WPT
net–BI
app–BI
PLT
OI
RSI
SI
DOM

WV	app–BI	PLT	OI	RSI	DOM
net–BI	0.96	0.92	0.91	0.74	0.81
app–BI		0.9	0.91	0.76	0.82
PLT			0.92	0.7	0.79
OI				0.78	0.94
RSI					0.77

WPT	app–BI	PLT	OI	SI	DOM
net–BI	0.99	0.86	0.87	0.83	0.76
app–BI		0.85	0.88	0.84	0.77
PLT			0.89	0.78	0.78
OI				0.84	0.94
SI					0.79

Fig. 4. Spearman's correlations among AppQoS metrics and between net-BI and AppQoS metrics for the WV (left) and WPT (right) datasets.

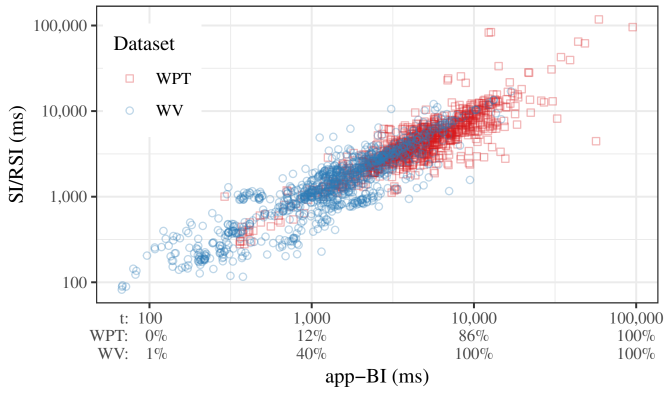


Fig. 5. Scatter plot for 1000 sampled points. x-axis denote the proportion of elements having app-BI $\leq t$ for each dataset.

feature engineering only (gray shaded background) as well as when its estimation is carried out using machine learning from the periodic time-series (white background). Here feature engineering means considering time-aggregated net-BI as a direct approximation of the app-BI, without use of learning algorithm. While app-BI estimation with the time-aggregated net-BI is already fairly accurate (about 100ms error or 6%), it can be seen that machine learning enhances app-BI forecast, lowering the median error to about 50ms (3%).

Particularly, the fact that app-BI correlates with the other metrics, coupled to the low error in app-BI estimation explains the good performance in estimating any time-related performance indicator. This is further expressed in Fig. 4, that depicts the Spearman correlation between AppQoS metrics in both WPT and WV datasets. Two important remarks can be gathered from the picture. First, from the top-row it can be seen that net-BI is highly correlated with all the other metrics, and additional dependencies are further captured by ML models. Second, it can be seen that the exact value of the correlation can vary depending on confounding factors in the measurement campaigns: particularly, notice that RSI vs SI metrics measured respectively in WV and WPT have different definitions, and hence correlation values; also interestingly, whereas RSI (WV) correlation is lower than SI (WPT), for all remaining metrics that are identically measured by WV and WPT, it is generally the opposite, i.e., correlation is higher in WV than in WPT. This reinforces the need to widen the boundaries of the investigation beyond those of a single environment. Otherwise, the risk is that the gathered results may limitedly apply to a dataset, and may not generalize to other environments or real deployments.

We further visualize the just exposed differences in the VW vs WPT datasets by focusing on the SI/RSI metric, which is among the most important to assess web QoE. Fig. 5 reports a scatter plot of app-BI vs SI (WPT) or RSI (WPT) for the two datasets. Particularly, it can be seen that points in the scatter plot align well around the $y = x$ intercept, which explains the high Spearman correlation. At the same time, notice that, as encoded in the x-axis label, only 12% of WPT (40% of WV) samples have an app-BI up to 1 sec. Also, whereas all WV samples have an app-BI of less than 10 sec, there are 14% of WPT experiments having app-BI in excess of 10 sec. While the underlying reasons of these differences are multiple (e.g., this can be due to remote Asian locations used in WPT, or public dataset instances of WPT, etc.), it is more important to study in which conditions the ML models that have been exposed to only part of these conditions during training, can generalize to previously unseen conditions – our main focus in the remainder of this paper.

B. Deeper view: Model generalization

The previous section has shown that machine learning models provide a satisfactory forecast of AppQoS metrics from encrypted traffic. Whereas this is a positive fact for ISPs, drawing such a conclusion would be naïve in our opinion. Despite the large scale of the experiments, the variability of the dataset and the use of methodologically sound cross-fold validation, the models have so far been tested in quite homogeneous conditions: i.e., they were exposed, during training, to a random sample of *all* tested conditions.

For instance, the previous section has shown that our models work well for two major browsers (Firefox and Chrome, that make up for 70% of the market share according to https://gs.statcounter.com/), yet it is unclear to what extent such performance would generalize to the remaining (non-chromium based) browsers in the market, or can generalize across browser versions or to future browser releases. In the above case, the model generalization capability (aka *portability* in the remainder of this section) can be answered

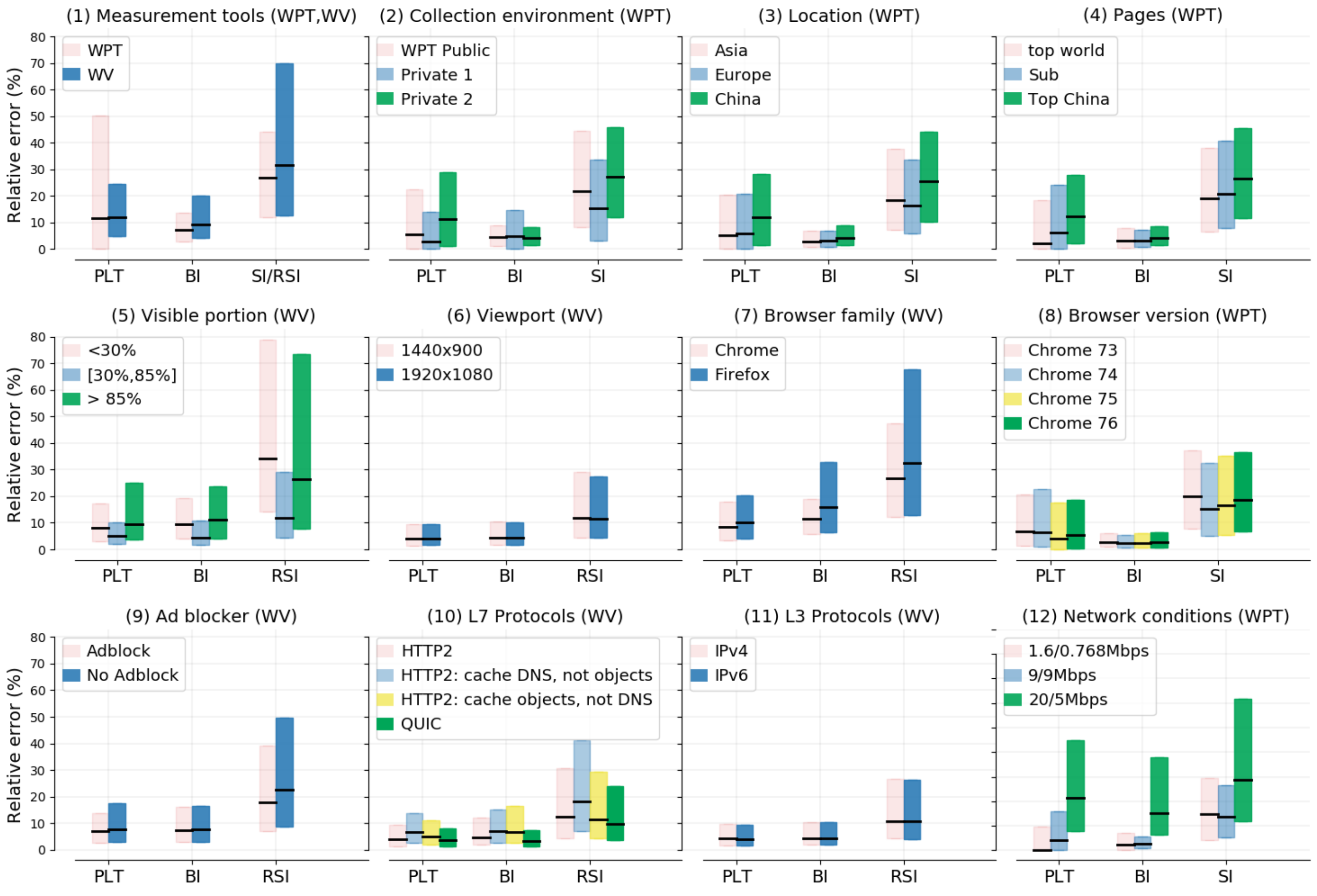


Fig. 6. Model generalization: leave-one-out tests, where models are purposely built so that conditions in the test fold are not exposed during training.

by purposely avoiding to expose the training process to a particular browser B, and testing the model accuracy on exactly that left-out browser B.

More formally, in this section we systematically stress test the model generalization capabilities by exposing it, during training, to only part of the conditions in our dataset. To do so in a principled way, we resort to *leave-one-out* testing where we systematically exclude each time one condition (e.g. Firefox) from the training and test on this excluded condition. This allows us to quantify, in a fine-grained way, what are the most difficult factors that limit the model portability. In turn, this knowledge can be useful to prioritize retraining (e.g., when new browser versions are introduced, or new protocols, etc.) and more efficiently guide future measurement campaigns.

Fig. 6 shows at a glance our portability results across some of the most important dimensions in our datasets. The figure shows the results for three metrics (PLT, BI, SI/RSI) using the LightGBM models. The results for each left-out condition are represented by a boxplot showing the median, first and third quartiles of the relative error when testing the inference model on the left-out condition. From top to bottom, left to right, we present portability across the (1) *measurement tools* (WPT vs WV), as well as the (2) *collection environment* (Public vs Private) and (3) *probe location*. Content plays undoubtedly an important role, for which we break down portability per (4) *cluster of pages* (World, China, Subpages of the same page) and (5) *portion of the visible page*, as not all content is rendered above the fold. We also consider user heterogeneity by contrasting (6) *viewports* that stem from different user devices, as well as (7) *browser families* vs (8) *browser versions*, and (9) the use of *Ad blocker* plugins. Finally, from the network viewpoint, we consider (10) *L7 Protocols* (QUIC vs HTTP2, with/without caching of HTTP/DNS responses), (11) *L3 Protocols* (IPv4 vs IPv6) and (12) different *network conditions*. Several remarks are in order.

(1) Measurement tools, (2) Environment and (3) Location. The first subplot in the figure shows the effect of leaving out the entire WV or WPT datasets from training, which allows us to understand whether a single measurement campaign is sufficient to build AppQoS estimators that generalize well to unseen conditions. We observe that when it comes to PLT, omitting WPT from training (i.e. training on WV only) and testing on WPT, results in higher relative errors (larger inter-quartiles), compared to omitting WV from training. The explanation is that WPT experiments have much higher PLTs (recall Fig. 5) and that learning PLTs of slow-loading pages from examples containing mainly fast-loading pages yield higher errors. Overall, it can be remarked that while median PLT and BI generalize well (median error on previously

unseen conditions on the order of 10%) however the error can grow large for the 3rd quartile (up to 70% for PLT). Similarly, it can be seen that median error for SI/RSI estimation on previously unseen conditions roughly doubles (up to 30% whereas it was 14% when training on both WV and WPT datasets). It is thus clear that there is a benefit in exposing models to an as heterogeneous as possible set of conditions, so that hopefully the estimation accuracy does not degrade in operational deployment. We will come back on this aspect in Sec. VI.

Variability extends also to public vs private instances of the same tool, as well as different locations of the private instances. Particularly, whereas BI estimation remains precise in both cases, we notice that environment and location have a roughly similar impact on PLT and SI estimation.

(4) Target pages and (5) Visible page portion. The variability across groups of pages shows that the model generalization capability to unseen pages of different part of the world (World, China), or different subpages (Sub) remains on par with the environmental variability. This is reassuring as it suggests that there may be no need to provide a very fine-grained per-page modeling.

Conversely, the visible portion of the page plays a significant role and better explains the root of the error. It is well known that not all the page is visible immediately after download, and the portion of the visible page is generally referred to as "Above the Fold" [8]. We first compute for each experiment the ratio between the height of the screen and the total length of the page. This ratio represents the proportion of the page which is above-the-fold. Based on this ratio, we separate the experiments into 3 bins of equal sizes.

Notice that the behavior is non monotonic in the visible portion: this is intuitive since in the case where we are testing visible portion in [30%,85%), the model has been fed with both extremes, containing pages with either a [0%,30%) or [85%,100%) visible portion. As such, the model can generalize by interpolating to intermediate visible portions in [30%,85%) based on the extreme examples, while the opposite is not possible with the same accuracy level.

(6) Device viewport, Browser (7) family / (8) version and (9) AdBlock. User devices (hardware), browser (software) and preferences (AdBlock) are clearly expected to affect model generalization capability.

Intuitively, for any given page, a larger viewport increases the portion of the visible page. This is symmetrical to the previous case where the visible portion was intrinsically due to the page content, and is equally important due to the large variety of user devices. Results show that models generalize well across the two popular 1440x900 and 1920x1080 viewport sizes.

Conversely, as it can be expected, models generalize poorly across browsers families: this is due to the fact that browsers differ in the rendering engine, so that exactly as viewing performance differs across browsers, AppQoS models should include samples from all browsers of interest. On the positive side, models' performance is largely portable across major versions within the same browser family, which means that retraining should not happen on a timescale of nowadays (agile) software evolution.

TABLE II
SUMMARY OF MODEL GENERALIZATION PERFORMANCE.

Factor	WPT	WV	Rel. Err.	Rank
Measurement tool	✓	✓	31.6%	1
Visible page portion		✓	29.1%	2
Browser family		✓	28.8%	3
Probes environment	✓		23.8%	4
Target page set	✓		22.9%	5
Probe location	✓		22.1%	6
Ad blocker		✓	20.3%	7
Network conditions	✓		19.0%	8
Browser version	✓		17.4%	9
L7 Protocols and caching		✓	13.5%	10
Viewport		✓	11.7%	11
IPv4 vs IPv6		✓	10.8%	12
All	✓	✓	14.0%	

Similarly, we see that performance estimation can be affected by presence of Ad blocking plugins, for which including both options in model training seems relevant, particularly for the RSI metric.

(10) L7 protocols, (11) L3 protocols and (12) network conditions. Another aspect influencing performance pertains to the servers configuration (HTTP2 vs QUIC), as well as whether DNS responses or HTTP objects happen to be in the device (or proxy) cache. Interestingly, we see that models generalize well across these different settings. Similarly, the use of IPv4 or IPv6 at the lower layer of the networking stack is only minimally affecting model portability.

Finally, given the abundance of related literature on the topic, it is not a surprise that protocol performance is affected by the bandwidth resources available in uplink/downlink. As such, it is vital that multiple network conditions are included in the model, as this would otherwise hamper the model accuracy, limiting its relevance from an operational standpoint.

VI. DISCUSSION

In this work, we have shown that (i) learning AppQoS metrics from encrypted network traffic is possible (ii) that machine learning models have a generally satisfactory accuracy for a broad range of metrics and that (iii) machine learning models generalize quite well to unseen conditions, with accuracy degradation that are however highly variable depending on the considered condition.

Whereas the previous section has delved into each condition in detail, it is now worth to relatively compare the impact of each condition. To this purpose, Tab. II compactly reports the model generalization ability. Specifically, each of the subplots in Fig. 6 gathered by leave-one-out validation is summarized as a single scalar value, namely the weighted mean of the median relative error over the whole set (and not only the left out condition) for SI/RSI. By considering the most critical metric, we gather a conservative analysis of the relative error in the estimation. The table also reports the average median relative error for SI/RSI on the bottom gathered by 5-fold cross-validation, which is useful as a reference. Based on this table, we now make the following recommendations.

The main takeaway from the table is that *pooling datasets from heterogeneous sources* is extremely beneficial: i.e., when models are trained over both datasets, the model is exposed to a larger variety of conditions, which significantly help in reducing the estimation error (14% vs 31.6%, rank 1). Including public vs private instances (23.8% error, rank 4) from multiple locations (22.1% error, rank 6) is also desirable, although this clearly has a large infrastructural cost.

Second, *stratified sampling of pages with different visible portions* is a good criterion for target page selection: in particular, failing to do so can yield a larger error 29.1% (rank 2) that cannot be simply offset by selecting pages from different geographical areas (error 22.9%, rank 5). This is trivial as it boils down to a more careful selection of the target set, and does not add any further deployment cost.

Third, models generalize poorly across *browser families and plugins* that alter the page rendering process. Including multiple browser families (28.8% error, rank 3) and AdBlocker configurations (20.3% error, rank 7) is a necessary price to pay to increase model generality that is difficult to offset otherwise. Including multiple browser versions is instead far less important (17.4%, rank 9).

Fourth, and rather interesting, it seems that network conditions (19.0%, rank 8) and protocols have a smaller impact on the model generalization. As shown earlier, network conditions have a measurable effect on the performance. As such, it is recommended to expose models to new transport technologies and network vantage points. At the same time, especially for what concerns L7 (13.5%, rank 10) or L3 protocols (10.8%, rank 12), it seems that ISPs should not worry much about the ability to capture their users' Quality of Experience in spite of the fast-paced changes in the application domain that happen outside their control.

These considerations are both reassuring, and hopefully helpful for the research community, in giving a clear view for the most important aspects that are relevant from an operational perspective.

VII. Conclusion

In this paper, we propose a methodology to infer application-level objective Quality of Experience metrics of web browsing directly from raw network encrypted traffic. Our method leverages standard machine learning models, fed with the curve of byte progression, to gather popular metrics such as SpeedIndex, ByteIndex and Page Load Time.

To train and validate the models, we perform a large set of experiments, where we have collected simultaneously network traces and application-level information on a disparate set of conditions. Our experiments show (i) that supervised machine learning models accurately approximate application-level metrics; (ii) that such models further generalize rather well across a large diversity of experimental conditions, and finally, (iii) we provide guidelines about the most important aspects to ensure successful deployment in operational settings, for which we hope that releasing our dataset to the scientific community is a hopefully valuable contribution.

References

[1] D. Naylor *et al.*, "The cost of the S in HTTPS," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 133–140.

[2] L. Vassio *et al.*, "You, the web, and your device: Longitudinal characterization of browsing habits," *ACM Transactions on the Web (TWEB)*, vol. 12, no. 4, p. 24, 2018.

[3] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 295–312.

[4] M. Varvello *et al.*, "Eyeorg: A platform for crowdsourcing web quality of experience measurements," in *ACM CoNEXT*. ACM, 2016, pp. 399–412.

[5] F. Salutari *et al.*, "A large-scale study of Wikipedia users' quality of experience," in *In proceedings of the 30th Web Conference (WWW'19)*, San Francisco, CA, USA, May 2019.

[6] E. Bocchi *et al.*, "The web, the users, and the mos: Influence of http/2 on user experien ce," in *PAM*, Apr. 2017.

[7] https://www.w3.org/TR/navigation-timing-2/.

[8] http://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692.

[9] https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index.

[10] E. Bocchi *et al.*, "Measuring the quality of experience of web users," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 4, pp. 8–13, 2016.

[11] T. Enghardt *et al.*, "Web performance pitfalls," in *PAM*. Springer, 03 2019, pp. 286–303.

[12] A. Saverimoutou *et al.*, "Web browsing measurements: An above-the-fold browser-based technique," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1630–1635.

[13] https://github.com/WPO-Foundation/RUM-SpeedIndex.

[14] R. Netravali *et al.*, "Vesper: measuring time-to-interactivity for web pages," in *USENIX NSDI*, 2018, pp. 217–231.

[15] A. Saverimoutou *et al.*, "A 6-month analysis of factors impacting web browsing quality for QoE prediction," *Computer Networks*, vol. 164, p. 106905, 2019.

[16] A. S. Asrese *et al.*, "Measuring web quality of experience in cellular networks," in *PAM*. Springer, 2019, pp. 18–33.

[17] ——, "Measuring web latency and rendering performance: method, tools & longitudinal dataset," *IEEE Transactions on Network and Service Management*, 2019.

[18] M. Rajiullah *et al.*, "Web experience in mobile networks: Lessons from two million page visits," in *WWW*, 2019.

[19] A. Saverimoutou *et al.*, "Web View: A measurement platform for depicting web browsing performance and delivery," in *IEEE Comm. Mag.,*. IEEE, to appear.

[20] https://www.itu.int/rec/T-REC-G.1030.

[21] T. Hoßfeld *et al.*, "Speed Index: Relating the industrial standard for user perceived web performance to web QoE," in *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, 2018, pp. 1–6.

[22] D. Da Hora *et al.*, "Narrowing the gap between QoS metrics and web QoE using above-the-fold metrics," in *Passive and Active Measurement (PAM)*, 2018.

[23] https://performance.wikimedia.org.

[24] M. Trevisan *et al.*, "PAIN: A passive web performance indicator for ISPs," *Computer Networks*, vol. 149, 2019.

[25] A. Huet *et al.*, "Web quality of experience from encrypted packets," in *ACM SIGCOMM Posters and Demos*, 2019.

[26] https://webqoe.telecom-paristech.fr/data.

[27] https://www.webpagetest.org/about.

[28] G. Ke *et al.*, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.