Deployable models for approximating web QoE metrics from encrypted traffic

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

***Abstract*—Being on endpoints, Content Providers can easily evaluate end users' web browsing quality of experience (web QoE) by accessing in-browser computed application-level metrics. Because of end-to-end traffic encryption, it is becoming considerably harder for Internet Service Providers (ISPs) to evaluate the web QoE of their customers, which is important for management purposes. In this paper, we propose data-driven machine learning techniques and exact flow-level algorithmic methods to infer well-known application-level web performance metrics (such as SpeedIndex and Page Load Time) from raw encrypted streams of network traffic. We prove the efficiency of our approach taking as input a unique dataset of more than 200,000 experiments, targeting a large set of popular pages (Alexa top-500), from probes from several ISPs networks, with different browsers (Chrome, Firefox) and viewport combinations. Results show that our data-driven models are not only accurate for several web performance metrics, but also feature the ability to generalize to previously unseen conditions. Furthermore, we discuss how our extremely lightweight flow-level method has a provable accuracy on a specific metric, and is thus of particular appeal from a deployment viewpoint.**

***Index Terms*—Machine learning; Network monitoring and measurements; Quality of experience; Web**

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, such as simple Page Load Time (PLT) [7] or more advanced indicators related to visual progression as the SpeedIndex (SI) [8]. 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 (e.g. PLT) or javascript code (e.g. SpeedIndex and variants [9]). Among these indicators, the ByteIndex (BI) [9], defined as the cumulative progression in time of the downloaded objects sizes in bytes, acts as a bridge between network traffic and AppQoS: intuitively, 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) from encrypted network data. 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 error) and inference generalizes well to any other AppQoS metrics (less than 14% median error in the worst case).
- Third, we systematically study the transferability of the data-driven models to previously unseen conditions at multiple levels (environment, viewport settings, location, browser family and version, network conditions, etc).
- Fourth, we introduce a flow-level net-BI implementation that we formally prove to have the same accuracy of its packet-level counterpart: in reason of its extreme lightweight, flow-level alternative is of particular appeal on constrained deployment settings.

This paper is an extension of [10], that first described our approach to estimate the QoE based on encrypted network traffic. This paper presents the follow-up of our research

that we extended notably along (i) the comparison of several machine learning models, as well as (ii) the formal proofs of net-BI computation and (iii) the net-BI online computation.

The remainder of the paper is organized as follows. Sec. II presents background material on web QoE performance measurement and Sec. III formulates the problem, overviewing our testbeds and the open source datasets. Sec. IV describes exact and data-driven models for AppQoS inference. Sec. V contrasts the accuracy of multiple models and Sec. VI systematically assesses the ability of such models to generalize to unknown conditions. Sec. VII proposes simple theoretically sound and lightweight flow-level algorithms that (limited to one metric) are appealing for constrained deployment. Finally, Sec. VIII overviews related literature and Sec. IX summarizes our findings.

II. A primer on web performance

From a network standpoint, the rendering of modern web pages requires downloading hundreds of complex objects (style sheets, images, javascript code dynamically generating requests for other objects, etc.) hosted on dozens of domains, which are fetched opening multiple connections per domain, using multiple protocols (e.g., HTTP, HTTP/2, HTTP/3).

Web practitioners use a waterfall diagram to represent the full sequence of these download events, from the initial DNS request to map the domain name of the visited web page, to the final request of the last object referenced in the page. Web performance is generally assessed within the browsers, with objective application-level metrics that can be *directly extracted* (or *require to be computed*) from the web rendering waterfall. We denote such metrics, which are broadly used as a proxy of web user Quality of Experience, as *AppQoS metrics*, of which we here present a taxonomy following the organization presented in [9].

In this work, we do not propose any new metric, but rather an effective method to systematically learn any of these metrics, from encrypted traffic. We point out that while our proposal is quantitatively assessed on few popular metrics, the extent of application of our methodology is more general than its evaluation in this paper.

A. Time instant (direct)

AppQoS metrics of the *time instant* class track important events during the page rendering. Notable examples include the Time to The First Byte (TTFB), the Time to The First Paint (TTFP), the time at which parsing of the Document Object Model (DOM) is completed, the Time To Interaction (TTI) after which a page is responsive for users, the time at which all the content Above-the-Fold (ATF) is rendered [11], and the time at which the page was fully loaded: Page Load Time (PLT). With few exceptions, most of these metrics are defined in the W3C normative reference and are directly accessible via APIs such as Navigation Timing [12], Paint Timing [13], and Resource Timing Level [14]. Fig. 1 depicts the progress of the page download process, that originates at the browser `navigationStart` event ($t = 0$ of the session) and finishes at the `onLoad` event (corresponding to PLT=`onLoad`− `navigationStart` in the figure). Several time instant metrics are annotated as vertical dashed lines in the figure. (more metrics are covered in Sec. VIII).

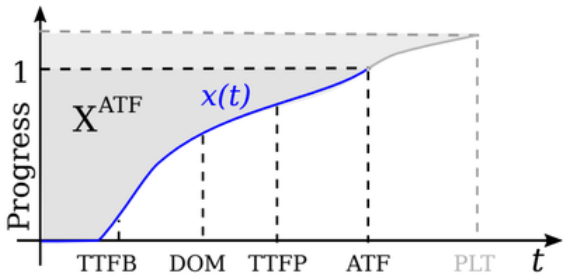


Fig. 1: AppQoS primer from [16]: time instants (vertical lines) and time integral (shaded areas) metrics

Although time instant metrics are apparently simple, it should be noted that (i) different browsers may have different implementations of such metrics, and (ii) slightly different definitions for the same metrics may be used in practice. As for (i), it is important to assess cross-browser compatibility, which has limitedly been done so far [7]. As for (ii), we remark that recent research [7] points out that even for very simple and widely used metrics such as the PLT, there exists significant discrepancies (20% for the median page) depending on the adopted definitions. It follows that for more complex and recent metrics such as the ATF (which identifies the time at which the browser finished entirely rendering the viewport area), further implementation discrepancies may arise. For instance, the testbeds we use in this work implement slightly different implementations of the ATF concept: Web View implements ATF as the Time for Full Visual Rendering (TFVR) [15] whereas Web Page Test implements it as Visually Complete (VC) [11], and both support the Approximated ATF (AATF) [16] chrome plugin we previously released [17].

For consistency, among the time instant metrics we consider DOM and PLT, which are less ambiguously defined, as well as prominently used (PLT in particular) in related literature.

B. Time integral (computed)

Reducing web performance to the timing of *single events* is not without downsides: for instance, ATF definition is not agreed upon, whereas PLT is the extremum of a time-series, and by definition its measurement can be prone to outliers. For such reasons, the web performance community started defining metrics that exploit the timing information of *all events* in the webpage waterfall, which can be conveniently done by relatively simple integrals.

Considering the process $x(t)$ of the normalized page progress in Fig. 1, a time integral metric X is defined as the residual cumulative progress, depicted as the gray shaded area above the $x(t)$ curve, that is computed as:

$$X^{t_{end}} = \int_0^{t_{end}} \big(1 - x(t)\big)dt \tag{1}$$

where t_{end} is the cutoff of the integral (e.g., note that in the example in the figure, the cutoff is set at t_{end}=ATF for which $x(ATF)$ is renormalized to 1). Intuitively, for two runs of the same page ending at the same PLT, it is smaller for the run that appears more responsive at the beginning (i.e., whose progress is faster $x'(t) > x(t)$).

Several metrics have the form (1), and vary only for the specific choice of the $x(t)$ implementation. The first proposal in this class, namely the SpeedIndex [8] (SI), is based on the visual progress of the page rendering process, computed at pixel-level (with problems due to changing content) or by histogram of the pixel-colors (approximation). Similarly, Perceptual SpeedIndex (PSI) [18] enhances SI by further using SSim video similarity metrics. ReadyIndex [19] (RI) tracks the readiness of the page, by averaging visual progress (i.e., paint events) of an element with the progress of its functionality (i.e., whether all JavaScript related to that element have been downloaded). However, SI, PSI and RI are heavyweight and have not been deployed outside lab environment. Real User Monitoring SpeedIndex (RSI) [20] is a lightweight approximation that circumvents this problem by using the bounding box size as proxy of visual completion. Finally, even simpler approximations have been proposed: ByteIndex [9] (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. Similarly, ObjectIndex (OI) [9] uses object count progress, where all objects are counted equally regardless of their size.

As metrics in this class are particularly important, we consider an extensive selection to include SI (as a prominent and reference metric, generally used in the literature), RSI (that has been recently used in a large scale study involving real Wikipedia users [5]), as well as BI and OI (that have as well enjoyed deployment [21]).

C. User perception (QoE)

AppQoS metrics are objective quantities meant to approximate web users' subjective Quality of Experience (QoE). Web QoE has been the object of recent attention, with several studies focusing on involving real users or volounteers, either in lab [16], by means of crowdsourcing [4], [22], or through real campaigns [5] where users are asked for different types of perceptual satisfaction (a feedback 5-grade Absolute Category Ranking [6], [16], a binary user acceptance [5], [23], or a user perceived delay [4], [24]). In a nutshell, while the instant/integral AppQoS metrics differ quantitatively, they are qualitatively based on the assumption that the quicker the content is loaded, the higher the user QoE, which has been ascertained by studies involving real web users [5], [16], [24]–[26]. We remark that different AppQoS metric pairs are highly correlated: for instance, the observed correlation on top-100 Alexa web pages [9] between SI and PLT (resp. BI) is 0.79 (resp. 0.82). However, the relationship of subjective experience and AppQoS metrics is far from being fully elucidated yet: e.g, [27] shows that animated content can impact the perception of ATF, and [24] shows that different objective AppQoS metrics are better fit to represent different classes of users having different perceptual abilities. *This suggests to consider several AppQoS metrics as relevant proxy of user experience.*

Furthermore, while an exact model for web users' QoE is still not agreed upon, a relatively simple rule of thumb can be employed to gauge the rough expected impact that AppQoS delay has on web QoE perception. In particular the classic psycho-behavioral Weber-Fechner law [28], which shows a logarithmic separation of human perception timescales, has been later successfully casted to the computer network domain [29]. As human nature and senses have not fundamentally changed, "the basic advice regarding response times has been about the same for thirty years" [29], i.e., 100ms is the limit for having the user feel that the system is reacting instantaneously, whereas 1s is the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. *For the web, this boils down to saying that 100ms are practically unnoticeable from the user – which can help us appreciating the accuracy of our models later on.*

III. Testbeds and datasets

We provide a brief overview of the testbed and tools we use and of the experimental workflow we follow. In particular, Fig. 2 provides a view at a glance of our measurement campaign, and Tab. I reports further details on the datasets that we collect and make available [30].

A. Testbeds

The testbed we built is based on two software tools, namely Web Page Test (WPT) [36] and Web View (WV) [37]. We point out that while both WPT and WV collect measurement directly from the browser, using more than one tool in the data collection process is a necessary step to ensure the generality of the QoE inference. In a nutshell, the use of both tools allows to explore a wide set of complementary aspects (cf Tab. I), which in turn is necessary to ensure that inference models are broadly applicable and their results representative of end-user experience.

1) Web Page Test: Web Page Test is a popular tool for measuring and quantifying websites performances. WPT is based on a software deployed on machines located at different points of the world. Public WPT instances enable anyone to perform tests to evaluate one website from a specified location of the planet, as it could be experienced by a user located in that place. For this type of use, WPT offers a public page to enable anybody to launch a test by selecting the location of the vantage point, the browser she wants to use and the target URL of the website. WPT allows to perform a single test (first view) or two tests (first view + repeat view) and measures the information for the repeat view: this emulates someone revisiting a website and allows to assess the impact of cached contents (both locally and in network proxy caches). Private WPT instances, such as the one we use in our testbed, run the same WPT software and can be instructed to perform large scale measurements. Some of our private WPT instances run on VMs in our lab servers using their native broadband connections, while others run from laptops in our labs that are either connected to WiFi or to 4G networks.

TABLE I: Summary of the datasets in this work (WPT, WV) vs related work. Datasets available at [30].

Factor	Web Page Test (WPT)	Web View (WV)	Literature
Locations	{Asia, EU, US} × {Public, Private}	EU	Generally in EU [31], [32] EU, US and others in [33]
Target pages	Top-500 × {world, China} × {main, subpages}	50 main pages from Top-500	Limited (3-20) in [31], [33], [34] Up to Top-10K main pages in [32]
Network conditions	50% native + 50% (Fiber, DSL, 4G, 3GFast, ...)[1]	(Fiber, ADSL)[2]	Ethernet, mobile broadband [31]
Browsers	Chrome	Chrome, Firefox	Mainly Chrome only [6], [34] Chrome and Firefox in [31], [32]
Viewports	Default	1920×1080, 1440×900	Default or not specified [19], [34] Five viewports in [32]
Protocols	{HTTP/2} × {IPv4}	{HTTP/2, HTTP/3} × {IPv4, IPv6}	From HTTP/1 only [33] to {HTTP/2, HTTP/3}× {IPv4} [31], [32] {IPv6} [35] but no AppQoS metrics
Metrics	BI, PLT, SI, OI, DOM	BI, PLT, RSI, OI, DOM	PLT, RSI, SI, ATF [31], [32]
Samples no.	119,395	111,171	

[1] https://github.com/WPO-Foundation/webpagetest/blob/master/www/settings/connectivity.ini.sample
[2] https://webview.orange.com/monitoringParameters

For each test, WPT requests the content of the specified page and logs several key metrics (see Sec. II) of the browsing process, and other information such as the volume of downloaded content, the number of connections and requests, etc. This is the information we exploit in this paper. Additionally, WPT assesses how good are the practices of the website: for instance, it informs if it properly compresses the text and the images, reuses existing connections, if it relies on a CDN network, etc. WPT offers a simple visual interface with a score (A being the best) associated with a color code (green, yellow, red). WPT argues that if the score is not A or B, issues should be addressed by the web owners to improve the design of their website (which is useful from the website developer viewpoint, but orthogonal with respect to the focus of our work).

2) Web View: Web View (WV) is a software tool developed by Orange whose main objective is to perform automated web browsing measurements. Similarly to WPT, WV collects page timing and browsing information, with the ability to configure several test parameters (such as the website list, the browser, the screen size, the use of ad blocker, etc.). In addition, WV can use specific transport protocols (HTTP/1, HTTP/2 or HTTP/3), which is useful to evaluate the adoption of these protocols by web servers, and their impact on the performance of its users (e.g., repeat tests can evaluate the benefits of website certificate re-use for faster handshake).

However, whereas WPT adopts the viewpoint of a web user, WV adopts the point of view of the ISP that users access to reach their favorite web pages. As such, WV probes are located behind commercial home boxes, allowing to know exactly the QoE that end-users in this access network could get. Moreover, WV is designed to continuously measure representative information of web pages from strategically placed network probes, in order to better qualify and understand web browsing performance and especially track how performance changes over time. To further identify the root cause of performance changes, WV also offers the ability to know content provenance: i.e., WV identifies the locations of the content providers and informs if the content is provided by CDN, cache servers or origin servers (which is useful from the ISP's viewpoint, but orthogonal with respect to the focus of our work).

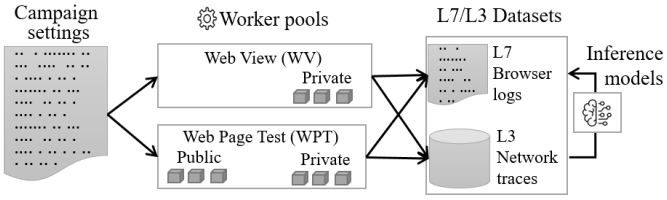


Fig. 2: Synoptic of the workflow in this paper

B. Workflow

We automate execution of data collection by instrumenting several workers (either public or private instances). Both tools have access to application layer information and can thus provide us with a groundtruth for the metrics we target to estimate. Each worker executes multiple web sessions, consisting, each, 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.

Given an AppQoS browser 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 Sec. IV.

Once the model has been trained, we test its prediction ability on new samples in Sec. V. 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 may degrade in other network settings. Since we are well aware of this fallacy, we particularly stress-test the generalization ability of our models in Sec. VI.

C. Datasets

To stress test the model's generalization capabilities, we purposely collect two distinct datasets with two different tools, that are available at [30]. 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 (55967) of WPT experiments are performed using the online service www.webpagetest.org at different locations in Europe (Frankfurt, Paris, London), Asia (Seoul, Singapore, Tokyo) and USA (Dulles). For the the other half (63428) of experiments we use private WPT instances in three locations in China (Beijing, Shanghai, Dongguan). The list of target URLs comprises the top-500 worldwide and the top-500 in China. We additionally randomly take 5 subpages from each main page, to increase content diversity with respect to the landing page.

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 other shaping/loss conditions (conditions are normalized in WPT; details are accessible through WPT configuration file pointed in Tab. I and are reported for each experiment in [30]). 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 at two different locations in France (Paris and Lannion). 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 HTTP/3, 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 HTTP/2 vs HTTP/3 [31], [32]. Lower layer protocols (e.g., impact of IPv6 that we consider in this work) have been disregarded to the best of our knowledge: indeed, existing work studying the impact of IPv6 on web performance [35] focused mainly on simple L3 (IPv6 reachability) or L4 statistics (TCP connect times of website), unlike ours which focuses on actual L7 AppQoS metrics. Similarly, vantage points are typically spread in few locations in few continents [31], [32]. With few exceptions [31], [32], most related work also 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 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 and illustrate how BI metrics can be computed from packet-level data, and later describe how to extend the inference to other AppQoS metrics.

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. It 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 [9], 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 [38]. In this case, the size s_w cumulates the size of packets received during the window $\sum_{j \in w} s_j$,

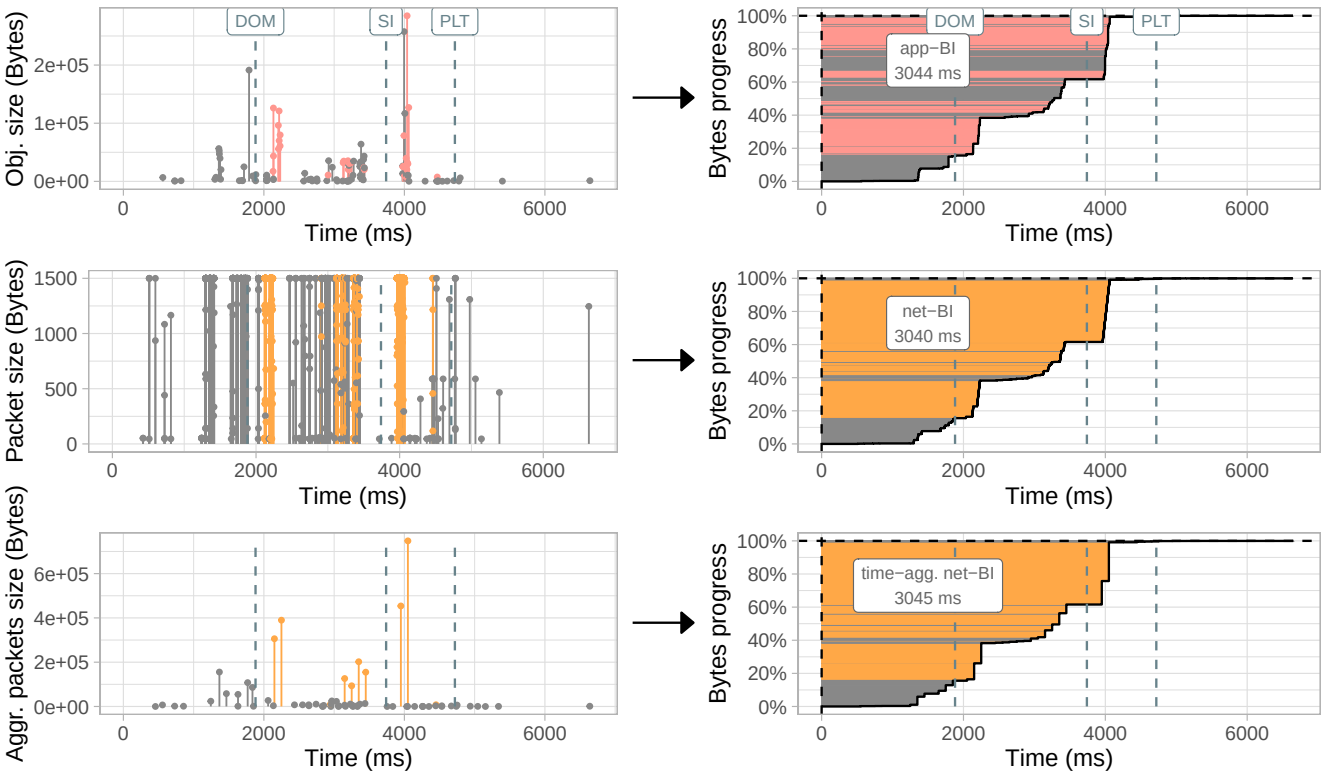


Fig. 3: Illustration of application (top) and network views (middle and bottom) for Byte Index metrics, i.e., the colored area on top of the $x(t)$ progress curve (black continuous line). Several other AppQoS metrics (DOM, SI, PLT) are annotated in the picture.

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. 3 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). 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. 3. 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). In addition, application-level BI is known to have high correlation [9] 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. To ensure that the models are portable (across types of web pages, different 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 based on the time-aggregated $x(t)$ shown in the bottom plot of Fig. 3. 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 3), 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 $y = f(\underline{x})$: For the regression task, we employ state of the art approaches from different branches of ML, namely (i) decision trees, (ii) gradient boosting and (iii) deep learning.

As for (i), decision trees are known to be extremely simple yet accurate for regression tasks: as such, they provide an insightful baseline that lower bounds computational complexity of ML methods. For this category, we consider the popular CART algorithm [39] which we use as a baseline.

However, (ii) ensemble techniques are known to provide superior accuracy with respect to individual trees, at the price of additional computational complexity. For this class, we leverage the LightGBM [40] gradient boosting method, which is the current state of the art in this class. We tune its hyperparameters using Bayesian optimization: a satisfactory set of hyperparameters used in the following 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. Clearly, in reason of the additional number of estimators, we expect a performance payoff at the price of moderate complexity increase – since the tree depth is bounded, and since tree operations during inference boil down to simple *if-then-else* branches.

Finally, as for (iii) deep learning techniques, we resort to 1D-Convolutional Neural Networks (CNN) as they offer superior performance on time series: this is due to the non-linear feature extraction capabilities of the convolutional layers, which are well suited in capturing time-series patterns of high complexity. The computational cost of CNN (for training and inference) is higher than that of the previous methods: as such, deployment consideration may discourage the use of more complex alternatives (e.g., LSTM or transformers), unless justified by a significant accuracy improvement. We devise a deep learning 1D-CNN architecture and hyperparametrization that we use consistently for all QoE metrics, and that operates on the time-aggregated net-BI samples in input (opportunely normalized between 0 and 1). In particular, we adopt a VGG-16 [41]-like architecture, by stacking consecutively series of Convolutional and Pooling layers before closing with a pair of dense layers. As in VGG-16, we choose a small filter size and a small stride (respectively 3 and 1 in our case). Overall, our 1D-CNN model employs 9 layers for over 2 million trainable parameters. Given that our dataset is not particularly big with respect to the model size, we rely on dropout layers to avoid overfitting: in particular, we use a 0.5 dropout after each dense layer and a 0.25 dropout after each set of convolutional and pooling layers. We expect our 1D-CNN to provide superior accuracy, at the price of an even increased complexity due to the large number of matrix computations involved in the inference process.

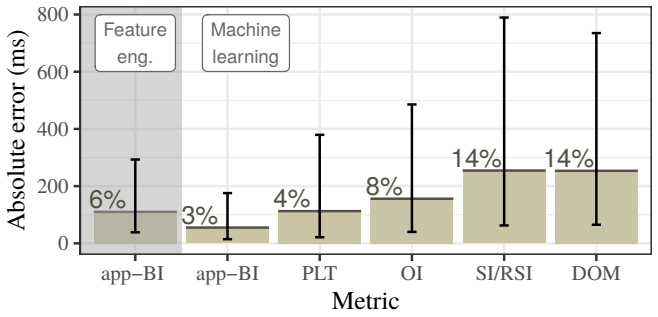


Fig. 4: Absolute median errors (bar plot) with first and third quartile (error segments) for predicting five AppQoS metrics in the WV and WPT datasets, with feature engineering (gray, for app-BI only) and machine learning using LightGBM. The relative median error in percent is added close to each bar.

4) Training and evaluation: We use two different processes to respectively (i) quantify accuracy and complexity of the above models, to select the best candidate for deployment in Sec. V and (ii) stress test model generalization capabilities in Sec. VI. As for (i), as typically done in the literature, the whole set of experiments is randomly split into five equally-sized subsets to perform *5-fold cross validation*: in this case, models are exposed during training to all conditions in the dataset, so that it is possible to relatively compare ML models in a principled way (due to cross-validation), although it remains unclear how such model would perform when exposed to completely different environments (e.g., a new browser).

As such, we (ii) stress test model generalization by performing more challenging split of the different folds along the different conditions of a dimension to test. While the methodology is detailed in Sec. VI-A, in a nutshell the idea is to, e.g., check generalization capability over browsers by training over Chrome and testing over Firefox (and vice versa), ensuring a wider discrepancy of the training-testing conditions.

V. Model accuracy

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 dataset as a whole. We next dissect deeper the performance across multiple models and testbeds.

A. Bird-eye view: Aggregated results for all metrics

We start by quantifying the accuracy of machine learning models in inferring the range of AppQoS metrics in our WPT and WV datasets using 5-fold cross validation. Fig. 4 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

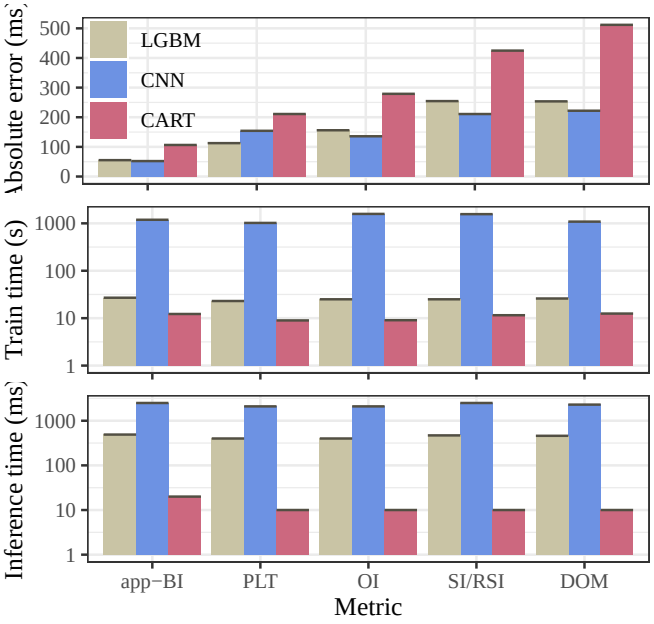


Fig. 5: Absolute median errors (top), training time (middle) and inference time (bottom) for different ML/DL models and AppQoS metrics.

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 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%).

In light of the Weber-Fechner law, and the timescale separation of human perception for which 100ms difference in rendering times are hardly perceivable by end-users, and delays up to 1s are tolerable, it can be seen that AppQoS metrics can be learned with a useful degree of precision.

B. Digging deeper: Comparison across models

Whereas the general workflow for learning supervised regression models is the same, the use of different models such as a single CART decision tree, vs an LightGBM boosting tree ensemble, vs a CNN deep learning model yield to different operational points in the trade-off between model accuracy and complexity. Fig. 5 reports the median and quartiles model accuracy (top plot), together with the training complexity (middle) and inference complexity (bottom) complexities for all AppQoS metrics and models. Notice that the error statistics for LightGBM model map thus directly to those reported in Fig. 4.

Expected tendencies are immediate to gather. In terms of complexity, CART is the fastest to train, specifically more than 2× faster than LightGBM and over 100x than CNN.

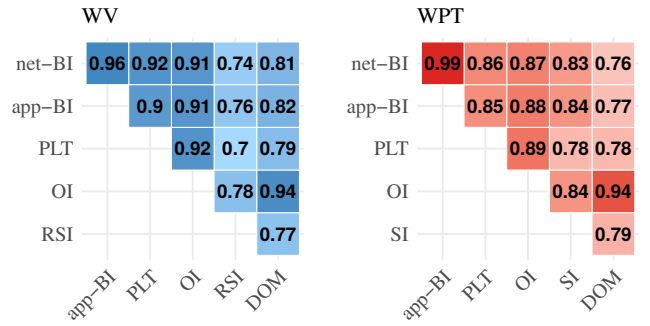


Fig. 6: Spearman's correlations between net-BI and AppQoS metrics for the WV (left) and WPT (right) datasets

Inference times are also favorable to CART that is about 40–50× faster than LightGBM, that is already 5× faster than CNN. Note that these inference time refer to inference for the *whole dataset*, after the dataframes are loaded in memory: it follows that inference complexity is negligible from a practical viewpoint. More importantly, model simplicity comes at the price of reduced accuracy, as the relative error is also roughly 2× worse than LightGBM and CNN. On the other side, the accuracy is best for CNN, which outperforms LightGBM on all metrics except PLT. However, the reduction of few percents of relative error wrt LightGBM comes at the price of more costly training (and inference) processes. Notice also that in absolute terms, CNN error improves by few tens of milliseconds, which may have little practical impact compared to the timescales of human perception. These observations have important consequences for our work since, as we shall see in Sec. VI, a thorough evaluation of the model transferability requires to *systematically retrain* the models to stress-test their generalization capability – which requires fast training methods.

Summarizing, whereas CART trees are inexpensive yet inaccurate, and CNN accurate but costly to train, LightGBM stands out for having performance that is close to CNN, with far smaller training times. In the reminder of this paper we thus limitedly use LightGBM models, as they represent a "good enough" choice for practical deployment.

C. Digging deeper: Comparison across testbeds

We now extend the comparison to explicitly contrast datasets collected across different testbeds. We start by examining in Fig. 6, the Spearman correlation between AppQoS metrics in both WPT and WV datasets. Two important remarks can be gathered from the picture. First, net-BI and app-BI have high correlation with the other AppQoS metrics: coupled to the low error in app-BI estimation, this partly explains the good performance in estimating any time-related performance indicator, since additional non-linear dependencies can be 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,

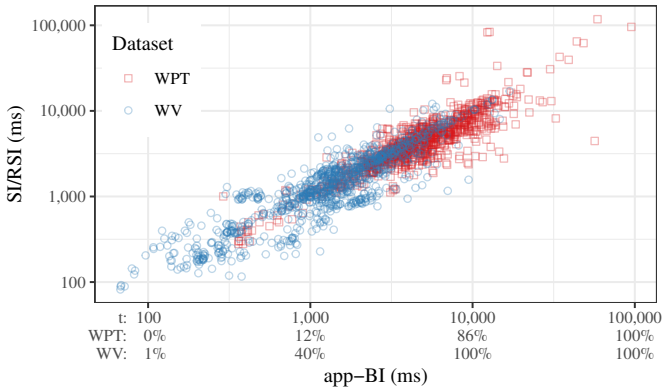


Fig. 7: Scatter plot for 1000 sampled points. x-axis denote the proportion of elements having app-BI $\leq t$ for each dataset.

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.

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. 7 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. The underlying reasons of these differences may be multiple, as e.g., this can be due to remote Asian locations used in WPT, or public dataset instances of WPT, etc. A question arise about ML models whose performance have been studied in this section: particularly, it is unclear how well models that have been exposed to only part of these conditions during training, generalize to previously unseen conditions.

Otherwise stated, the above observations reinforce the need to widen the investigation boundaries 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: assessing the model portability is thus our main focus in what follows.

VI. Model Generalization

We now focus on portability and 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 datasets.

A. Methodology

The previous results have 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 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.

In principle, to stress test generalization capabilities of the models, this methodology can be extended to consider pairs of conditions to be left-out from training. In practice however, the breakdown of settings across testbeds reduces the combinatorics of the conditions that can be left out: indeed, several combinations cannot be tested (e.g., any combination involving two testbeds, such the impact of Viewport, available only on WV, vs the network conditions, available only on WPT) while the available combinations would reduce the heterogeneity (e.g., as the impact of AdBlocker vs L7/L3 protocols could be tested only on the WV, or similarly the network conditions vs browser combinations are available only on WPT). As such, we limitedly consider each condition in isolation in the following.

B. Results

Fig. 8 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

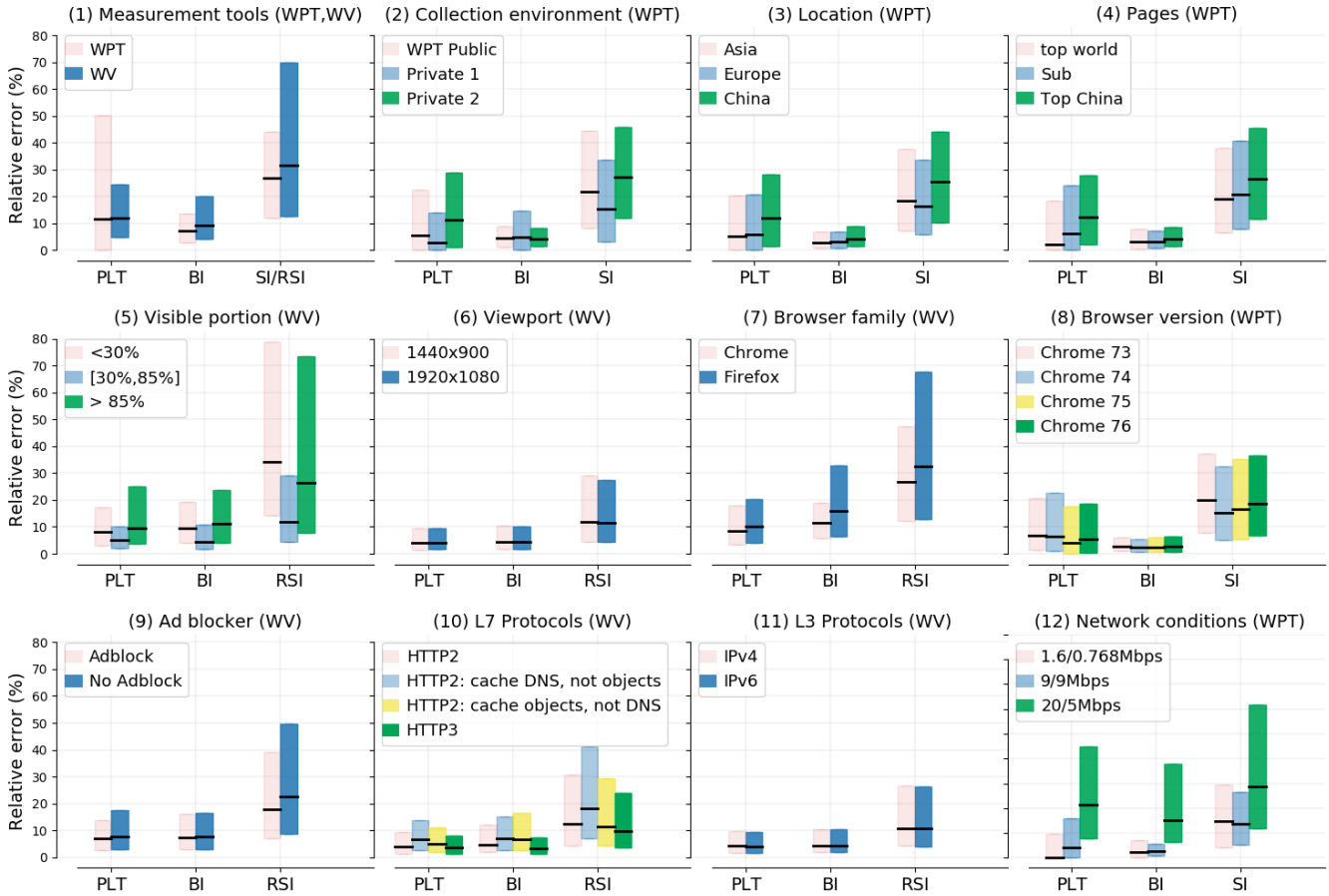


Fig. 8: Model generalization: leave-one-out tests, where models are purposely built so that conditions in the test fold are not exposed during training.

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* (HTTP/2 vs HTTP/3, 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.

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. Further work can though help to make sure that our findings remain valid for different types of pages or less popular ones from top 1M Alexa.

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" [11]. 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 browser 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.

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 (HTTP/2 vs HTTP/3), 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.

C. Implications

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. 8 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.

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%	

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 be able to capture their users QoE in spite of the fast-paced changes in the application domain that happen outside their control.

These considerations are reassuring, and give the research community a clear view of the most important relevant aspects from an operational perspective.

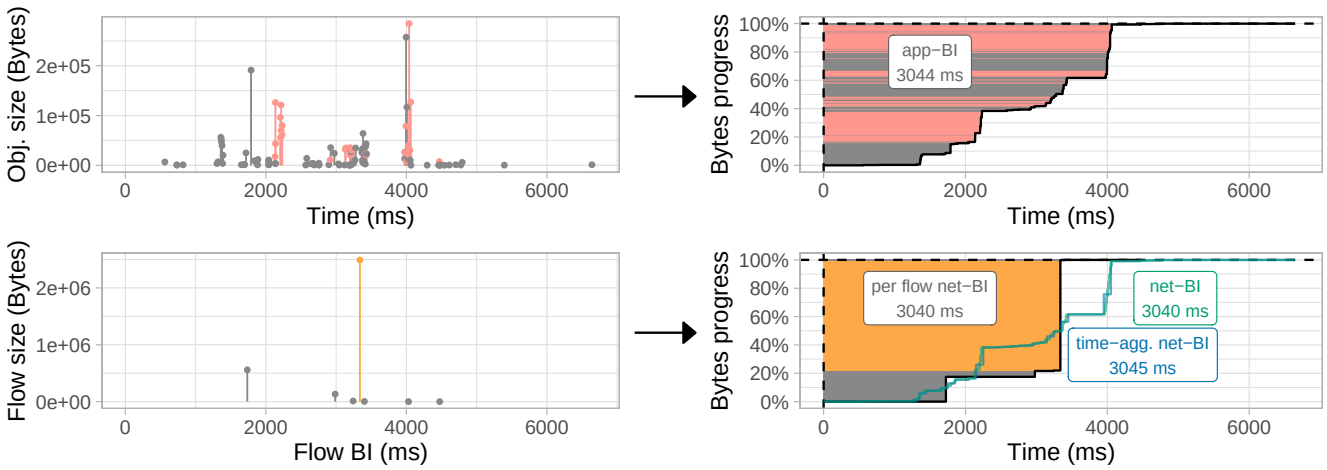


Fig. 9: Illustration of application view (top) and network view at the flow level (bottom). In the network view, the reference time for each flow is taken as its *flow BI time* (which is not the start time of the flow). This choice allows exact reconstruction of the net-BI metric, see Sec. VII-B for definition of *flow BI time* and details.

VII. A LIGHTWEIGHT FLOW-LEVEL ALGORITHM FOR PRACTICAL DEPLOYMENT

In this section, we consider the viewpoint of an ISP or equipment vendor that is interested in a practical deployment of the above models. Particularly, whereas previous sections have focused on learning ML models for AppQoS inference (Sec. IV), evaluating the accuracy and complexity of such models (Sec. V) and stress testing their generalization capabilities (Sec. VI), the operational complexity of gathering the necessary packet-level input features was never properly accounted for.

Particularly, notice that the app-BI considers object sizes as seen by the browser, and only requires to perform simple *object-level* computation (i.e., after each object is received). Conversely, the net-BI approximation from encrypted traffic as well as the ML models introduced earlier, need to operate at *packet-level*, by e.g., aggregating packets received during windows having as duration $W = 100$ms, which is significantly more complex in practice, compared to easily available *flow-level* measurements.

In this section, we propose a new solution so that the net-BI of a web session can be computed from simple *flow-level* computations (i.e., when the net-BI of individual flows is available through IPFIX [42]) and formally prove that this can be done without any loss of accuracy. We additionally present a simple online algorithm (operating at packet-level of a single TCP flow) to compute the *flow-level net-BI*, that is of high practical appeal for deployment (e.g., for IPFIX export of net-BI of individual flows).

A. High Level Idea

For the sake of illustration, we first contrast from a high level perspective the object-level app-BI to our new flow-level net-BI in Fig. 9. Note that to compute the flow-level net-BI, we introduce on a new time, called flow BI, which can be computed per flow: it corresponds to the flow Byte Index that we introduce later (BI_j in Eq. 12).

First, as previously, on the left, events at different granularity (browser objects on top, TCP/IP flows on the bottom) are represented by segments with different times and sizes. The DNS domain contributing the most to app-BI is colored in pink, while the TCP/IP flow corresponding the most to net-BI is colored in orange, whereas other objects and flows are depicted in gray. On the right, the corresponding cumulative curves $x(t)$ are drawn in black, and the cumulative integral metrics of the BI families are represented as the shaded areas. For reference, the $x(t)$ curves corresponding to the time-aggregated net-BI and packet-level net-BI reported earlier in Fig. 3, are now overlayed to our new flow-level net-BI.

We recall that the time-series of application-level and network-level events cannot be directly compared: few objects are loaded in the browser, which originate from either an even smaller number of flows (as in Fig. 9), or from a very large sequence of packets (recall the packet-level view in Fig. 3). In particular, it can be seen that any flow-level representation will place all the bytes corresponding to all the objects for the dominant domain in the example (the pink events on top), at a single point corresponding to the aggregate size of all objects carried by the flow (the single orange event on bottom), depicted as an impulse occurring at a particular time. Several choices are possible concerning the placement of this impulse in time (e.g., flow start time, flow end time). We show that placing the impulse at *flow BI time* (defined later in Sec. VII-B) yields to an exact reconstruction of the net-BI. In particular, whereas the flow-level $x(t)$ curve looses in terms of granularity with respect to packet-level or object-level curves, we see in Sec. VII-B that it faithfully captures the most important information, i.e., the area above the $x(t)$ curves that correspond to metrics of the BI family.

B. Theoretical properties of net-BI metrics

1) Alternative form of the BI metrics: The representation of Byte Index as an integral makes it easier to understand in the narrative. However, it does not give a straightforward way

for computing it. In this section, we introduce an alternative summation form that makes its computation as well as the formal proof of some relevant properties easier.

We consider a web session consisting of a sequence of packets such that the i-th one occurs at time $t(i)$ (relative to the session start) and transmits $b(i)$ bytes. We are interested in the individual contribution of each packet to the whole session, for which we derive

$$p(i) := b(i) / \sum_k b(k). \tag{2}$$

We are going to show that the network-level BI (net-BI) can be expressed as follows:

$$\text{net-BI} = \sum_i p(i)t(i). \tag{3}$$

First, we describe $x(s)$ the curve corresponding to the progression of bytes. At any instant s, the progress $x(s)$ is the sum of individual contribution until s, so

$$x(s) = \sum_i p(i)\mathbf{1}_{t(i)\leq s} \tag{4}$$

Then, we note that $\sum_i p(i) = 1$, which allows deducing a simple form for the complement of the progression:

$$1 - x(s) = \sum_i p(i) - \sum_i p(i)\mathbf{1}_{t(i)\leq s} = \sum_i p(i)\mathbf{1}_{t(i)>s}. \tag{5}$$

Putting all together, we express the Byte Index as follows:

$$\text{net-BI} := \int_0^\infty [1 - x(s)]ds = \int_0^\infty \sum_i p(i)\mathbf{1}_{t(i)>s}ds. \tag{6}$$

We switch the summations using Fubini's theorem, which gives us the desired result:

$$\text{net-BI} = \sum_i p(i) \int_0^\infty \mathbf{1}_{t(i)>s}ds \tag{7}$$

$$= \sum_i p(i) \int_0^{t(i)} ds \tag{8}$$

$$= \sum_i p(i)t(i). \tag{9}$$

Recall later that switching summations between time and size gives a first intuition on how the BIs from different individual flows should be combined to find the flow net-BI: instead of interpreting events based on the instant of occurrence (x-axis), we interpret them as a progress of the completion (y-axis). Equivalently, instead of calculating the Riemann integral (*vertically*, by integrating the visual progress corresponding to a given time increment dt), it is more appropriate in this case to calculate the area above the $x(t)$ curve with a Lebesgue integral (*horizontally*, by evaluating the elapsed time corresponding to a given increment of the visual progress dx). Notice that visual clues about Lebesgue integration are already given representation of Fig. 3 and Fig. 9: the colored horizontal stripes indeed cover the full area above the $x(t)$ curve, exactly as a Lebesgue integral would compute it.

2) Decomposition of net-BI into flow-level elements: We decompose the net-BI of a web session into flow-level elements. To do so, we start from Eq. 3 and *rearrange the packets* belonging to the same flow together:

$$\text{net-BI} = \sum_i p(i)t(i) = \sum_j \sum_{i\in \text{flow}j} p(i)t(i). \tag{10}$$

This decomposition into smaller elements is illustrated in Fig. 10. On this figure, the predominant flow is colored in orange whereas other ones are in gray. Graphically, the rearrangement of the packets corresponds to reordering and grouping elements on the y-axis, as shown in Fig. 10 (b).

Afterwards, we consider the relative proportion of each flow among all flows by doing a *renormalization*. Instead of only normalizing packet size relatively to the session size, as conveyed by $p(i)$, we first normalize the packet size contribution over the current flow size, and later normalize the flow size over the whole session size. To do so, we let $B = \sum_i b(i)$ the total number of bytes for the session, B_j the number of bytes for the flow j, where by definition we have: $B = \sum_j B_j$. For a packet i belonging to flow j, we thus rewrite:

$$p(i) = \frac{b(i)}{B} = \frac{b(i)}{B_j}\frac{B_j}{B}. \tag{11}$$

This multiplicative form separates the contribution of the packet to the flow j from the contribution of the flow j to the whole session. Rewriting from Eq. 10:

$$\text{net-BI} = \sum_j \frac{B_j}{B} \left[\sum_{i\in \text{flow}j} \frac{b(i)}{B_j} t(i) \right] \tag{12}$$

$$= \sum_j \frac{B_j}{B} \text{BI}_j. \tag{13}$$

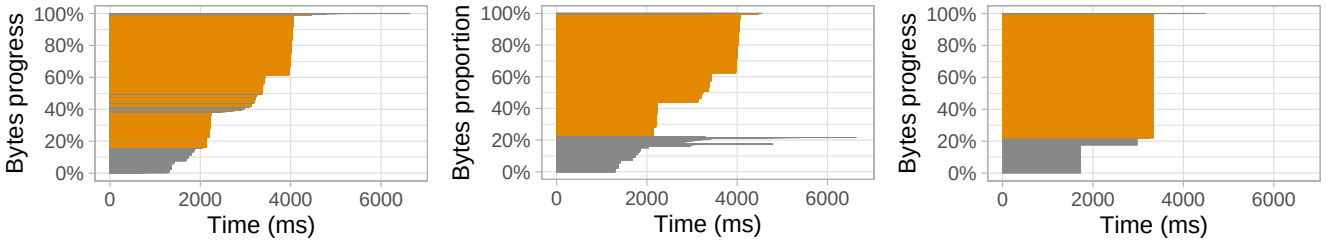
This introduces the new term BI_j, which we call the *flow BI time* and which can be computed from (i) the byte progression of each individual flow and (ii) the start time of the entire session, similarly to how the net-BI is computed from the byte progression of the entire session and the start time of the session in Eq. 3. The net-BI is finally obtained by weighting this term by the proportion of the current flow size over the total session size B_j/B.

This rewriting of the session net-BI in terms of a sum of products is illustrated in Fig. 10-(c), where each rectangle represents a flow j with height B_j/B and width BI_j. In particular, the orange colored flow has an height of 0.77 and a width of 3337 ms.

C. Online computation of net-BI metrics

Net-BI metrics can be easily computed online. To do so, we (i) compute the flow-BI of each flow independently and (ii) combine BI of all flows of a session a posteriori.

As for (i), we point out that flow BI for a single flow j, relative to its own start time, which we refer to as $\text{BI}_j^{\text{alone}}$, can be trivially computed online by a router. Given a flow consisting of events $(t_i, b_i)_i$ with initial time t_0 of the flow,



(a) (b) (c)

Fig. 10: Illustration of breaking down the Byte Index of a session into the Byte Index of individual flows. Packets of the main flow is colored in orange while packets of other flows are colored in gray: (a) the original bytes progress curve over time; (b) individual contribution of each packets are shifted over the y-axis in order to group packets of the same flow together; (c) each flow is summarized by its mean value relatively to its proportion. In each figure, the total area remains constant and equal to the net-BI.

at each packet the router updates the computation of the flow size B and of the *absement* A:

$$B = \sum_i b_i \tag{14}$$

$$A = \sum_i b_i (t_i - t_0) \tag{15}$$

where the flow size B counts each byte equally, whereas the absement A weights bytes by their arrival time since the beginning of the section. At the end of the flow, its BI can be simply derived as the ratio of the absement over the flow size:

$$\mathrm{BI}_j^{\text{alone}} = \frac{A}{B} = \frac{\sum_i b_i (t_i - t_0)}{\sum_i b_i} = \sum_i \frac{b_i}{B}(t_i - t_0) \tag{16}$$

As for (ii), a flow j is part of a session consisting of multiple flows, with $\min_k t_{0,k}$ the time origin of the session. Then, it is easy to recognize that the flow BI time in the session, i.e., the contribution of this flow on the whole session, satisfies:

$$\mathrm{BI}_j = \mathrm{BI}_j^{\text{alone}} + t_{0,j} - \min_k t_{0,k}. \tag{17}$$

where the difference corresponds to a translation (adding a rectangle of height 1 and width $t_{0,j} - \min_k t_{0,k}$ in terms of area).

From a practical deployment perspective, this solution offers the advantage of being online and lightweight, yet providing the exact session net-BI as computed at the packet-level. For instance, among the three statistics to be collected, start time t_0 and size of the flow B are already commonly exported by the router. The novel proposed absement statistic A is compatible with IPFIX [42] export and has a lightweight per-packet computation (one multiplication and one sum at each packet arrival, one division per flow). From those three per flow statistics, the net-BI is then reconstructed with a simple weighted-average across flows as in Eq. 13.

Summarizing, flow-level computation of net-BI can be sufficient from a practical deployment, since it closely approximates app-BI (cf Fig. 4), is correlated with the other AppQoS metrics (cf Fig. 6) and can further be leveraged to learn other metrics via ML, depending on the ISP/equipment vendors thirst for simplicity vs accuracy.

VIII. Related work

There are three separate lines of work that are related to ours: namely, (i) definition of AppQoS metrics, (ii) large scale monitoring of web performance within the browser and (iii) methods to infer web performance from encrypted traffic.

A. Defining web performance metrics

A lot of prior work [7], [9], [16], [19], [20], [25], [26] proposed interesting solutions to estimate QoE, that is however generally fit for browser-based measurements.

Our previous work [9] introduces ObjectIndex and ByteIndex metrics, showing based on experiments of the top-100 Alexa web pages, that they provide simple yet good approximation of the SpeedIndex. In general, studies proposing new objective-metrics generally seek agreement with subjective user-experience metrics, involving users through crowdsourcing. For instance, [16] (resp. [18]) perform a large-scale campaign to collect user subjective feedback to compare ATF (resp. SpeedIndex) against MOS (resp. user perceived PLT). Recent work has proposed several other metrics such as interactive Load Time (iLT) and Total Completed interactive Load (TCiL) [43], First Webpage Time (FWT) [44], the Perceptual Speed Index [18], or the ReadyIndex [19] to name a few. In our work, we consider the most popular ones, and especially focus on those whose code is already available.

In this paper, we do not propose new metrics, but rather a methodology that is able to learn any already existing metric from encrypted traffic. Additionally, we show that ByteIndex can be easily reconstructed in the network without machine learning, even in the presence of encryption, and from simple easy-to-compute flow level information.

B. Measuring web performance in the browser

Close to this work is a recent wave of efforts to measure AppQoS metrics in the wild based on browser-based active experiments [5], [6], [22], [31], [32], [34], [37].

In particular, Web View [37] is used to perform a six-months long measurement campaign [32] to study how known AppQoS metrics vary depending on various conditions to evaluate the impact of the different conditions on the QoE.

In this work, we leverage the same Web View platform to gather part of the dataset we make available [30]. Next, [22] proposes a framework able to integrate network measurement infrastructures with crowdsourcing platforms: the solution is however relying on DNS information which will be likely less accessible in the future (e.g., due to DNS over HTTPS) as well as client IP-geolocation which is defined nowadays as non-friendly according to European General Data Protection Regulation (GDPR). Our solution operates on encrypted traffic and is independent of underlying network protocols or IP-geolocation, taking into account all types of workers. The *Webget* measurement tool is used from 182 SamKnows probes to study web browsing under fixed-lines and Long Term Evolution (LTE) on 3 websites during 3.5 years [33]. Other large scale studies include [31], that leverage a dataset of 2 million visits from mobile networks of four different countries measuring TTFP, PLT and RSI for 7 pages. Complementarily, [5] focuses on a single services (namely, Wikipedia) and leverage answers from over 60k real users over several months and networks. Finally, [34] reports measurement results of eight websites for a duration of two weeks. While very valuable, each study focused on a limited subset of the most important aspects that can affect AppQoS performance.

Our work differs from the above in that we set to approximate AppQoS from encrypted network packets, assuming thus that we do not have access to browser information. Additionally, we take into account a larger set of websites under a significantly larger set of network conditions and metrics – heterogeneity of the experimental conditions is a particularly important aspect of our work.

C. Measuring web performance from the dark

Our work is motivated by the difficulty of QoE estimation from network traffic. A decade ago encryption was hardly the norm, and it was possible for ISPs to leverage HTTP logs to develop algorithms to detect the different page loading events [3]. TLS encryption has become pervasive in recent years, preventing access to payload, practically defeating approaches as to [3] and leaving ISPs without tools – such as those we propose in this work to fill this gap.

In this domain, work closer to ours has been conducted [45]–[48]. In particular, PAIN [45] 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. In more details, authors define a new passive web performance indicator which correlates with basic PLT metrics, as well as a supervised indicator (called BestCheckpoint). Our work is different, as we do not propose new metrics, and more general, as we learn any proposed metric. Our work, however, which focuses on the single web session case, can leverage PAIN's concurrent sessions methodology to isolate traffic of single page visits.

Similarly, a two-step approach to evaluate the QoE for web browsing is proposed in [46]: first, the system classifies the packets generated during the browsing of websites into clusters; then, an LSTM neural network, using network information such as the packet size and the packet arrival times, is used to predict the ATF time. We remark that authors first classify sessions, as they use different prediction models depending on the website category. This approach is different than ours, since it also need to access to the HTTPS header (or TLS domain name) for the website classification step. Our approach is simpler, blind to the type of website, and is still effective and portable as results show.

Recent work [47] propose a method that employs Cauchy loss to measure the discrepancy between observed and predicted QoS performance. The authors stress that their method is resilient to outliers: however we point out that, as web services (and underlying QoS) relentlessly change over time, these so-called outliers are prime indicators of poor web quality of experience. Our models have proven to be accurate over all conditions, and are also able to generalize to previously unseen conditions. Finally, recent work [48] focuses on a similar yet complementary aspect – i.e., notably to which extent QoE models developed for the desktop web are also portable to the mobile web. Results show that SpeedIndex based models are generally portable, which is in accordance with our findings in this paper.

Shortly, this work is the first to provide methods for an accurate, general, portable and lightweight inference of several web quality of experience metrics from encrypted traffic.

IX. Conclusion

In this paper, we propose a methodology to infer application-level objective Quality of Experience metrics of web browsing directly from raw and encrypted network traffic. Our methods are either able to readily compute specific metrics such as ByteIndex, or leverage standard machine learning and deep learning models, fed with the curve of byte progression, to learn any popular metric 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 that we release to the scientific community [30].

Our results show that (i) inferring AppQoS metrics from encrypted network traffic is possible, and (ii) some metrics can be accurately inferred from very simple algorithms, directly at flow level. Shall complexity consideration prime over accuracy (e.g., deployment constraints), the simple algorithm we propose limitedly measure a single metric (i.e., ByteIndex), yet it achieves the best trade-off between simplicity (simple per-packet operations) and accuracy (provable by design).

Whenever the accuracy and flexibility considerations are more important than deployment simplicity, it is instead recommended to exploit data driven models. By leveraging state-of-the-art machine learning and deep learning models, we have further shown that data driven models (iii) have a generally satisfactory accuracy for a broad range of metrics and (iv) generalize quite well to unseen conditions, although accuracy degradation depends on the specific environmental condition – for which we further provide guidelines about the most important aspects in the dataset collection campaign to ensure successful deployment in operational settings.

ACKNOWLEDGEMENTS

This work was carried out in the context of the Joint Innovation Project (JIP) on "AI for QoE prediction" between Huawei and Orange.

REFERENCES

[1] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the S in HTTPS," in *ACM CoNEXT*, 2014.
[2] L. Vassio, I. Drago, M. Mellia, Z. B. Houidi, and M. L. Lamali, "You, the web, and your device: Longitudinal characterization of browsing habits," *ACM Transactions on the Web*, vol. 12, no. 4, p. 24, 2018.
[3] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *ACM IMC*, 2011.
[4] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki, "Eyeorg: A platform for crowdsourcing web quality of experience measurements," in *ACM CoNEXT*, 2016.
[5] F. Salutari, D. D. Hora, G. Dubuc, and D. Rossi, "A large-scale study of Wikipedia users' quality of experience," in *WWW*, 2019.
[6] E. Bocchi, L. De Cicco, M. Mellia, and D. Rossi, "The web, the users, and the MOS: Influence of HTTP/2 on user experience," in *PAM*, 2017.
[7] T. Enghardt, T. Zinner, and A. Feldmann, "Web performance pitfalls," in *PAM*, 2019.
[8] (2012). [Online]. Available: https://docs.webpagetest.org/metrics/speedindex/
[9] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the quality of experience of web users," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 4, pp. 8–13, 2016.
[10] A. Huet, A. Saverimoutou, Z. B. Houidi, H. Shi, S. Cai, J. Xu, B. Mathieu, and D. Rossi, "Revealing QoE of web users from encrypted network traffic," in *IFIP Networking*, 2020.
[11] J. Brutlag, Z. Abrams, and P. Meenan, "Above the fold time: Measuring web page performance visually," 2011. [Online]. Available: http://conferences.oreilly.com/velocity/velocity-mar2011/public/schedule/detail/18692
[12] (2021) . [Online]. Available: www.w3.org/TR/navigation-timing-2
[13] (2017) . [Online]. Available: https://www.w3.org/TR/paint-timing
[14] (2021) . [Online]. Available: https://www.w3.org/TR/resource-timing-2
[15] A. Saverimoutou, B. Mathieu, and S. Vaton, "Web browsing measurements: An above-the-fold browser-based technique," in *IEEE ICDCS*, 2018.
[16] D. Da Hora, A. Asrese, V. Christophides, R. Teixeira, and D. Rossi, "Narrowing the gap between QoS metrics and web QoE using above-the-fold metrics," in *PAM*, 2018.
[17] (2018) . [Online]. Available: https://github.com/TeamRossi/ATF-chrome-plugin/
[18] Q. Gao, P. Dey, and P. Ahammad, "Perceived performance of top retail webpages in the wild: Insights from large-scale crowdsourcing of above-the-fold QoE," in *ACM SIGCOMM, Internet QoE Workshop*, 2017.
[19] R. Netravali, V. Nathan, J. Mickens, and H. Balakrishnan, "Vesper: measuring time-to-interactivity for web pages," in *USENIX NSDI*, 2018.
[20] (2014). . [Online]. Available: https://github.com/WPO-Foundation/RUM-SpeedIndex
[21] (2020). *Tools ip-label*. [Online]. Available: https://tools.ip-label.io/
[22] R. K. P. Mok, G. Kawaguti, and k. claffy, "QUINCE: A unified crowdsourcing-based QoE measurement platform," in *ACM SIGCOMM Posters and Demos*, 2019.
[23] T. Hossfeld, P. E. Heegaard, M. Varela, and S. Moller, "QoE beyond the MOS: an in-depth look at QoE via better metrics and their relation to MOS," *Quality and User Experience*, pp. 1–23, 2016.
[24] F. Salutari, R. Teixeira, D. da Hora, V. Christophides, M. Varvello, and D. Rossi, "Implications of the multi-modality of user perceived page load time," in *IEEE MedComNet*, 2020, pp. 1–8.
[25] (2014). *ITU-T G.1030*. [Online]. Available: https://www.itu.int/rec/T-REC-G.1030
[26] T. Hoßfeld, F. Metzger, and D. Rossi, "Speed Index: Relating the industrial standard for user perceived web performance to web QoE," in *IEEE QoMEX*, 2018.
[27] H. Z. Jahromi, D. T. Delaney, and A. Hines, "How crisp is the crease? a subjective study on web browsing perception of above-the-fold," *IEEE NetSoft*, 2020.
[28] R. B. Miller, "Response time in man-computer conversational transactions," in *AFIPS'68 Fall, part I*, 1968.
[29] J. Nielsen, "Response times: The 3 important limits," 1993. [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/
[30] (2020). [Online]. Available: https://webqoe.telecom-paristech.fr/data
[31] M. Rajiullah, A. Lutu, A. S. Khatouni, M.-R. Fida, M. Mellia, A. Brunstrom, O. Alay, S. Alfredsson, and V. Mancuso, "Web experience in mobile networks: Lessons from two million page visits," in *WWW*, 2019.
[32] A. Saverimoutou, B. Mathieu, and S. Vaton, "A 6-month analysis of factors impacting web browsing quality for QoE prediction," *Computer Networks*, vol. 164, 2019.
[33] A. S. Asrese, S. J. Eravuchira, V. Bajpai, P. Sarolahti, and J. Ott, "Measuring web latency and rendering performance: method, tools & longitudinal dataset," *IEEE Transactions on Network and Service Management*, 2019.
[34] A. S. Asrese, E. A. Walelgne, V. Bajpai, A. Lutu, Ö. Alay, and J. Ott, "Measuring web quality of experience in cellular networks," in *PAM*, 2019.
[35] V. Bajpai and J. Schönwälder, "A longitudinal view of dual-stacked websites—failures, latency and happy eyeballs," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 577–590, 2019.
[36] (2008). [Online]. Available: https://www.webpagetest.org/about
[37] A. Saverimoutou, B. Mathieu, and S. Vaton, "Web View: A measurement platform for depicting web browsing performance and delivery," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 33–39, 2020.
[38] A. Huet, Z. Ben Houidi, S. Cai, H. Shi, J. Xu, and D. Rossi, "Web quality of experience from encrypted packets," in *ACM SIGCOMM Posters and Demos*, 2019.
[39] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees. statistics/probability series," in *CRC Press*, 1984.
[40] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017.
[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.
[42] P. Aitken, B. Claise, and B. Trammell, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," IETF RFC 7011, Sep. 2013.
[43] H. Z. Jahromi, D. T. Delaney, and A. Hines, "Beyond first impressions: Estimating quality of experience for interactive web applications," *IEEE Access*, vol. 8, pp. 47 741–47 755, 2020.
[44] L. Yongsheng, G. Yu, W. Xiangjiang, W. Xiaoyan, and F. Yufei, "Measuring QoE of web service with mining DNS resolution data," *ZTE Communications*, vol. 15, no. S2, p. 38, 2017.
[45] M. Trevisan, I. Drago, and M. Mellia, "PAIN: A passive web performance indicator for ISPs," *Computer Networks*, vol. 149, 2019.
[46] E. Song, T. Pan, Q. Fu, R. Zhang, C. Jia, W. Cao, and T. Huang, "Threshold-oblivious on-line web QoE assessment using neural network-based regression model," *IET Communications*, 2020.
[47] F. Ye, Z. Lin, C. Chen, Z. Zheng, H. Huang, and E. Yilmaz, "Outlier resilient collaborative web service QoS prediction," in *arXiv 2006.01287*, 2020.
[48] P. Casas, Z. B. Houidi, S. Wassermann, A. Huet, M. Seufert, N. Wehner, J. Schuler, S.-M. Cai, H. Shi, J. Xu, T. Hossfeld, and D. Rossi, "Are you on mobile or desktop? On the impact of end-user device on web QoE inference from encrypted traffic," in *IEEE CNSM*, 2020.

Alexis Huet is a Senior Research Engineer on Network AI at Huawei Technologies Co., Ltd. He received his MSc from ENS Lyon in 2010 and his PhD in 2014 from Claude Bernard Lyon 1 University. He then worked at Nanjing Howso Technology (2015-2018) as a data scientist, where he was responsible for data mining, modeling and implementation for different machine learning projects. His main research interests include computational statistics and applied mathematics for machine learning.

Antoine Saverimoutou is a data scientist at Orange Labs. He received his Bachelor degree in Applied Mathematics at the University of Cambridge, Bachelor and MSc degree in Computer Science at the University of Caen in France. He received his Ph.D. from Institut Mines Telecom Atlantique in France. His main interests include quality of service and experience, web services and AI.

Zied Ben Houidi is a Principal Engineer at Huawei Technologies. He received his Ph.D. from the University of Pierre et Marie Curie in Paris (2010), working with Orange Labs on data-driven performance analysis of core networks' routing protocols. Before joining Huawei, he worked at Nokia Bell Labs, leading various research projects on network data valorization (e.g. human-level behavior analytics, learning from network data to build recommender systems) as well as automated reasoning.

Hao Shi is a Research Engineer at Huawei Technologies. His research interests include the study of the Quality of Experience management of multimedia services from the perspective of the network and development of applied algorithms.

Shengming Cai is a Principal Engineer at Huawei Technologies, Dongguan. He received the BEng degree in information engineering from Shanghai Jiao Tong University, China (2009), and the PhD degree in electrical and electronic engineering from Nanyang Technical University, Singapore (2015). His research interests include SDN, network slicing, SLA guarantee, operations research and AI. He is the technical leader of related research projects in Huawei and actively engage in standards.

Jinchun Xu is a Senior engineer in Huawei Technologies, Dongguan. She received the MSc in Communication Engineering from Harbin Institute of Technology University, China (2012). At Huawei, she leads the service experience evaluation research project. Her research interests include SDN, network slicing, SLA guarantee and quality of experience.

Bertrand Mathieu is a Senior Research Engineer in OrangeLabs. He received the MSc degree from the University of Marseille, the PhD degree from the University Pierre et Marie Curie in Paris, and the Habilitation à Diriger des Recherches from the University of Rennes. He is working on programmable networks, QoS and QoE and new network solutions. He contributed to 12 national/European projects and published more than 70 papers in international conferences, journals or books. He is member of several conferences TPC and an IEEE senior member.

Dario Rossi is Network AI CTO and Director of the DataCom Lab at Huawei Technologies, France. Before joining Huawei in 2018, he held Full Professor positions at Telecom Paris and Ecole Polytechnique and was holder of Cisco's Chair NewNetParis. He has coauthored 15 patents and over 200 papers in leading conferences and journals, that received 9 best paper awards, a Google Faculty Research Award (2015) and an IRTF Applied Network Research Prize (2016). He is a Senior Member of IEEE and ACM.