When Deep Learning meets Web Measurements to infer Network Performance

Imane Taibi, Yassine Hadjadj-Aoul, Chadi Barakat

When Deep Learning meets Web Measurements to infer Network Performance

Imane Taibi, Yassine Hadjadj-Aoul
Université de Rennes 1, Inria, CNRS, France

Chadi Barakat
Université Côte d'Azur, Inria, France

***Abstract*—Web browsing remains one of the dominant applications of the internet, so inferring network performance becomes crucial for both users and providers (access and content) so as to be able to identify the root cause of any service degradation. Recent works have proposed several network troubleshooting tools, e.g, NDT, MobiPerf, SpeedTest, Fathom. Yet, these tools are either computationally expensive, less generic or greedy in terms of data consumption. The main purpose of this work is to leverage passive measurements freely available in the browser and machine learning techniques (ML) to infer network performance (e.g., delay, bandwidth and loss rate) without the addition of new measurement overhead. To enable this inference, we propose a framework based on extensive controlled experiments where network configurations are artificially varied and the Web is browsed, then ML is applied to build models that estimate the underlying network performance. In particular, we contrast classical ML techniques (such as random forest) to deep learning models trained using fully connected neural networks and convolutional neural networks (CNN). Results of our experiments show that neural networks have a higher accuracy compared to classical ML approaches. Furthermore, the model accuracy improves considerably using CNN.**

***Index Terms*—Passive measurements, Web, QoS, performance, prediction, machine learning, CNN.**

I. Introduction

Web browsing remains one of the most widespread applications in the Internet. It is therefore important for network operators and service providers to ensure that the quality of their services is guaranteed. It is also important for end users to be informed about the reality of their network access especially when the quality of their service deteriorates. The ability to monitor Web and network performance is thus essential to determine when and where degraded network conditions actually happen. In addition, understanding the relationship between the Web and the underlying network can help to better understand the state of the latter by assessing its quality, knowing that any interruption or abnormal behaviour can cause frustration to end users with serious economic impact.

Given the complexity of the relationship between the Web [1] and the underlying network, measuring and understanding this relationship is often difficult to realize [2]. Indeed, the complexity of Web pages has largely increased as now they include tens of objects stored within multiple servers, with Web browsers retrieving such objects through multiple connections and using JavaScript code or related technologies for page creation. The complexity of the underlying protocols has also increased with the advent of HTTP/2 and QUIC. All this has increased the complexity of the relationship between network performance and Web performance and consequently the complexity of the measurement and modeling tasks.

Besides network performance, determining Web metrics that influence the Web Quality of Experience (QoE) is another challenging task [3]. The PLT (Page Load Time) is one of the commonly used metrics. For example, Alexa reports the quantiles of the PLT, and Google uses PLT to rank search results. However, recent studies deduce that this metric alone cannot give a precise estimation of Web browsing quality, hence the need for finding more suitable metrics that are closer to screen rendering and its subjective evaluation by the user [4]. That is why new metrics such as SpeedIndex and Above The Fold (ATF) render time have seen the light.

Bridging the gap between Web performance and network performance requires the deployment of measurement tools at both levels. Whereas Web measurements can be carried out within the browser in an efficient and passive way without extra cost on the user terminal and on the network itself, getting information about the network requires the deployment of standalone measurement tools in the form of either applications (e.g., pathload for bandwidth measurement [5]), browser plugins (e.g., Fathom [6]) or explicit action by the user in visiting a web page to download a file for example. These solutions are either computationally expensive, less generic, or greedy in terms of data consumption. Furthermore, they can disrupt other ongoing communications and may even increase the level of network congestion. Hence, the interest of resorting to native browser level passive measurements in the limit of possible and second them by ML-based algorithms to improve the accuracy of their estimation.

In this paper, we start from this observation and propose to infer the main properties of the underlying network from passive measurements obtained from within the browser. We use machine learning to calibrate algorithms that allow such inference. By comparing deep learning algorithms to classical ML algorithms as Random Forest, we highlight the feasibility of the task, but also its complexity hence the need for sophisticated deep learning algorithms as convolutional neural networks (CNN). We prepare the ground for this modeling by carrying out a sensitivity analysis to understand the dependency between the performance at the Web level and the one of the network. We also design a methodology for collecting a large dataset that links Web performance

This work is supported by Inria within the Project Lab BetterNet.

measurements to underlying network measurements, through in-depth controlled experiments where network configurations are artificially modified. Over an example of three network metrics, namely the round-trip time (RTT), the download bandwidth and the loss rate, we prove that we can estimate them with acceptable accuracy by only using measurements passively obtained from within the browser.

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III we describe in details our approach for estimating the underlying network metrics from Web performance measurements as well as the data collection process. In Section IV we discuss the results of our experiments and we conclude the paper in Section V.

II. Background & Related work

A. Background

1) Web performance metrics: The loading of Web page involves a long list of events encompassing the request of the page, the response of the server and the rendering of the downloaded content. Monitoring this loading in all its finite steps is a challenging problem. Though, the literature introduces many types of Web metrics that can be obtained.

In our study, we opt for measuring the maximum of timing components related to navigation, which are well presented in W3Cs specifications 'Chrome Navigation Timing API' [7] and the 'Paint Timing API' [8]. So we consider the following: Connect Start, the time to start the connection with the server. DNS, the time when finishing looking up for the domain. Request, the time when sending the request to a server to retrieve a resource. Response, the time when receiving the last byte of the response. DOM, when the load of the document object model finished. First Paint (FP), the time when the first pixel is rendered. First Contentful Paint (FCP), the time when the first bit of content is painted. And finally the page load time (PLT), when the web page finishes loading.

2) Underlying network metrics: Before planning how to collect data on the network and the service traffic, it is important to identify which metrics are necessary to infer the Web quality. Taking into consideration the asymmetric type of Web traffic, we identified three metrics to collect. One of these metrics is the Round-Trip Time (RTT) which is the latency necessary to communicate from one host and back to it through our final destination. Then, we have the Download Loss Rate, which is the percentage of packets lost in the download direction (i.e., the number of packets lost over the total number of packets sent). Finally, one can find the Download Bandwidth, which stands for the maximum end-to-end throughput in the download direction.

B. Related work

Several recent studies have been carried out to estimate the quality of the Web [9]–[13]. These studies can be divided into two broad classes. First-class techniques are based on crowd-sourcing, which consist in collecting data from a large set of real users encountering real network conditions [14], [15]. Network conditions being driven by real scenarios, these techniques do not explore unrealistic areas, where for example losses are too high or bandwidth is too large. Some scenarios might exist but given their low probability, they tend to dilute in the mass of data corresponding to common scenarios. In general, the heterogeneity of users resulting from differences in terminal capacities, which is not necessarily known, the difference in access networks, whether or not their traffic is throttled, etc., all this means that the measurements taken by crowd-sourcing are biased by various factors that are not really controllable. Fortunately, the large number of users can reduce this measurement bias. On the positive side, and in addition to capturing the real scenarios, these techniques allow to detect scenarios that someone might not think of beforehand.

Second-class techniques are based on the construction of a model (or several models) for Web quality based on datasets collected by controlled experiments [16], [17]. Unlike the previous class, the approaches of this class, since they do not necessarily know the network conditions that users may face, must widely explore the different possible network conditions. Several methods have been proposed to effectively explore the very wide space of possible conditions, such as the quasi-Monte Carlo method [18], the active learning method [19] or the Fourier Amplitude Sensitivity Test (FAST) [20] that we are exploring in this paper.

Note here that in opposition to existing approaches that focus on estimating Web quality based on network conditions, our approach focuses on the opposite. Indeed, our objective is to leverage Web measurements to estimate the network conditions. The advantage of this approach stems from the fact that Web measurements are of passive nature and are available at very low cost, at the user terminal inside the browser, whether mobile or fixed. If proven to capture the network conditions, they can avoid overloading the network with active measurements (i.e., traffic injection), source of measurement bias and consumers of CPU and data at the access.

III. Estimating network Status from Web performance metrics

A. Methodology

In order to predict network status departing from Web measurements the first step is to collect a dataset that captures the link between Network QoS metrics and Web QoS metrics. We proceed by extensive controlled experiments where network configurations are artificially modified and measurements of both network and Web browser are collected. Then, we apply data analysis techniques to estimate the network status from the Web metrics. For that, we proposed a distributed system based on different entities that provides a platform to link the input (underlying network metrics) to the output (Web performance metrics) (see Fig. 1 for more details).

In our system, the *Experimenter* unit communicates directly with the *sampler*, using the `GetSample()` function, which requests the next configuration to experiment. The configurations consist in tuples of RTT, download bandwidth and download packet loss rate. It then enforces these

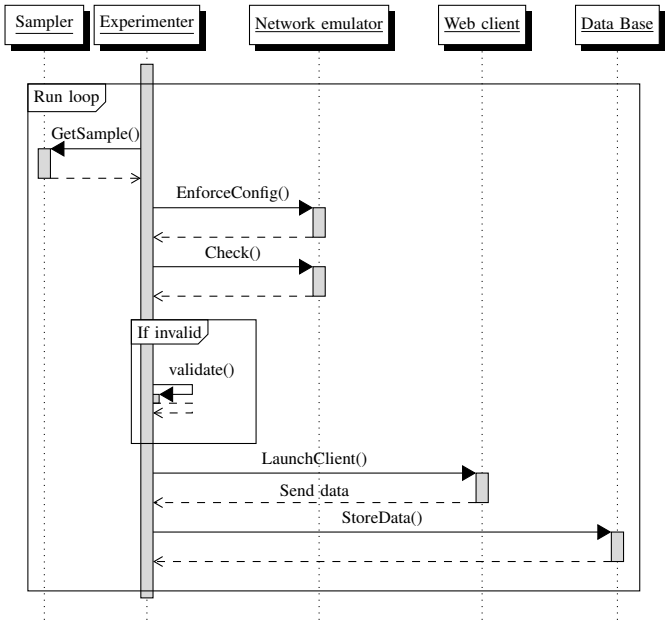

Fig. 1: Experimentation methodology

configurations thanks to a *Network Emulator*, using the `EnforceConfig()` function.

Our approach is based on lab experiments, where we aim to have under our control the network conditions between the client and the real Web servers. Whereas *tcconfig*, which was used to enforce network configurations, allows to control the access network, it does not provide control over the entire Web path. On one side the service Provider is out of our experimental network and on the other side the experimentation system requires to perform a real page loading from a cloud that is again out of our control. It follows that *tcconfig* is not always able to enforce the wanted network conditions (e.g., lower bandwidth or larger delay than needed). In order to handle the noise coming from the uncontrolled part of our experimental setup, we integrated measurement tests (using the `Check()` function) to ensure the validity of the samples. In particular, we implement at each desired configuration (i) a TCP throughput test to ensure that the available bandwidth is larger than the one we want to enforce, and (ii) an RTT and loss rate noise estimation tests to make sure that the total RTT and loss rate (measured plus real) are the ones we target.

Once the network configuration has been validated, the *Web client* is launched, using the `LaunchClient()` function. Our developed Web extension monitors the different Web metrics (Connect Start, DNS, Request, Response, DOM, FP, FCP, PLT), as well as other Web page characteristics, such as the number of objects, the size and the protocol supported. At the end of the experiment, the results are retrieved and the resulted statistics are stored by the *Experimenter* using the `StoreData()` function.

Note that we collect statistics from 100 top popular Web pages according to the Alexa ranking[1]. We will give more details on the implementation of our platform later in the text. The dataset obtained applying our methodology is composed by 10000 entries for each of the 100 top popular Web pages. These entries correspond to 10 repeated downloads of each page under the same network conditions, for a number of different network conditions equal to 1000 obtained using the FAST method defined next [20].

[1]https://www.alexa.com/topsites

B. Sensitivity Analysis

In order to have a clear understanding of the correlation between the measured Web metrics and the enforced network configurations, we start our study with a sensitivity analysis. This allows to better see the interplay between the network and the Web browsing performance. The idea here is to reveal whether the combination of different Web metrics brought information about the underlying network metrics.

To do so, we consider the Fourier Amplitude Sensitivity Test (FAST) [20], one of the most widely used sensitivity analysis techniques. FAST implements a periodic sampling strategy based on appropriate frequencies (one frequency per dimension) to ensure good coverage of the area to be sampled. In addition to providing a sequence of input tuples to experiment with, the method also allows to assess the sensitivity of the output labels (i.e., Web metrics in our case) to the input metrics (i.e., network metrics in our case) through the analysis of the spectrum of the obtained labels. Although this technique does not consider the properties of the network model in question here, it allows exploring the space of possible network configurations without missing important points. Indeed, unlike a generation of samples based on the Monte Carlo method [18], this technique allows to cover the space of network configurations efficiently by avoiding repetitions, thanks to the variation of frequencies. The partial variances obtained in this way make it possible to see the contribution of network metrics to the overall variation measured. The partial variance of an input metric corresponds to the partial energy of the output label summed over all frequencies multiple of the characteristic frequency of the input metric. The total variance (or total energy) is the sum over all frequencies. The ratio of partial variance to total variance (being real number between 0 and 1) models the sensitivity of the output label to the input metric. One can see this as the participation of the input metric, by its variability, to the total variability of the output label.

Sensitivity indices of a variance-based method are calculated via ANOVA-like decomposition of the function for analysis. Suppose the function is $Y = f(X) = f(x_1, x_2, ..., x_n)$ where x_i is a model parameter and Y is the output. For a parameter x_i, $V^{(x_i)}$ represents the partial variance in model output Y resulted from parameter x_i. $V^{(x_1,...,x_n)}$ represents the variance of model output Y resulting from uncertainties in all model parameters. Namely, $V(Y) = V^{(x_1,...,x_n)}$. The sensitivity index of input metric x_i is then defined as the normalized conditional variance:

$$S_{x_i} = \frac{V^{(x_i)}}{V^{(x_1,...,x_n)}} = \frac{V^{(x_i)}}{V(Y)} \quad (1)$$

We obtain in our case, for each Web metric and for each Web page, three sensitivity indices (S_{RTT}, $S_{Bandwidth}$, and $S_{LossRate}$). The boxplots (see Figures 2, 3, 4, 5, 6, 7, 8, and 9) display the dispersion of sensitivity indices over Web pages and network metrics for each of eight Web performance metrics. The y-axis in the figures shows the sensitivity index. We notice a strong correlation between input and output features, with some particular behaviors. In particular, Connect Start, DNS and Request are more sensitive to delay. Response is more sensitive to delay and bandwidth. First Paint and First Contentful Paint are more affected by bandwidth. As for the PLT, we don't observe a big difference between the different network metrics in terms of their impact.

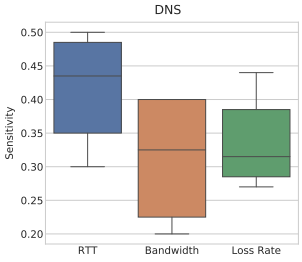

Fig. 2: DNS sensitivity to Net QoS

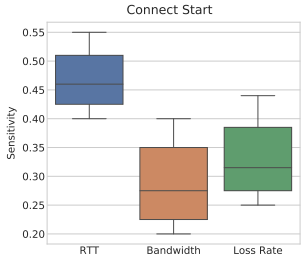

Fig. 3: Connect Start sensitivity to Net QoS

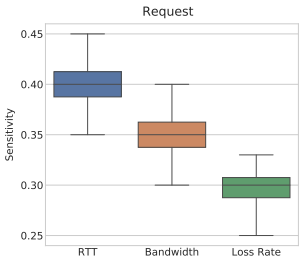

Fig. 4: Request sensitivity to Net QoS

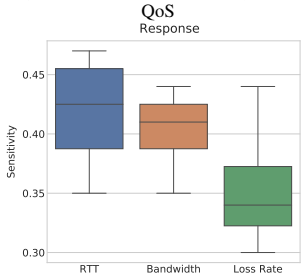

Fig. 5: Response sensitivity to Net QoS

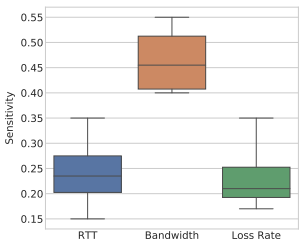

Fig. 6: DOM sensitivity to Net QoS

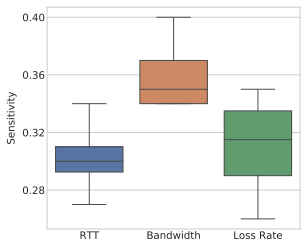

Fig. 7: FCP sensitivity to Net QoS

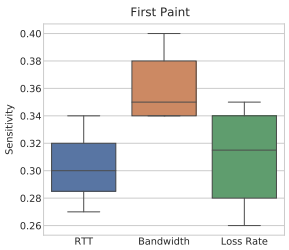

Fig. 8: First Paint sensitivity to Net QoS

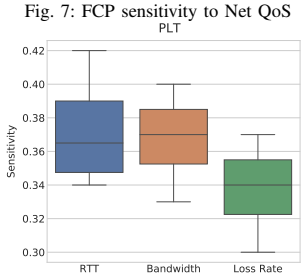

Fig. 9: PLT sensitivity to Net QoS

From these results we conclude that the patterns between network and Web metrics are really complicated, thus the need of advanced models if we want to estimate network status from a specific Web metric combination.

C. CNN-based network performance estimation

In this section, we give a detailed overview on the model we opt for to perform the estimation task of network conditions from Web metrics. We justify the use of such model later by comparing it to other learning algorithms.

The Convolutional neural network is a deep network topology that typically combines convolutional filter layers in conjunction with a fully connected network. A CNN works well for extracting more detailed features within the data which will then be used to form more patterns within higher layers. Based on this advanced technique, We want to build a model that estimates the network status by performing a CNN driven regression analysis, considering as input the Web performance metrics (Connect Start, DNS, Request, Response, DOM, FP, FCP, PLT, Global size, Protocol, Number of Objects), and as output the estimated value of the underlying network metric. Typically we will have three regressions, one for each of the network metrics we consider in this work.

We start by processing the obtained data in order to fit the estimator, then we separate it into training and test sets; for that we pick 70% of the dataset randomly as training set, and we consider the rest as test set. We use the training part to calibrate our Convolutional Neural Network Model and the test part to validate its performance. Our CNN estimator consists of : (i) an input layer, where we have 11 elements (i.e. the considered web metrics), (ii) a first 1D CNN layer (one dimension CNN layer), (iii) a second CNN layer, (iv) a Max pooling layer, (v) and finally fully connected neural network layers.

Regarding the hyper parameters tuning, the most important ones are the kernel size N_s, the number of kernels (or filters) N_k and the number of hidden neurons on the fully connected network part N_n. For the convolutional layers we define 100 filters (also called feature detector) $N_k = 100$ with a kernel size $N_s = 3$, this allows us to train 100 different features on the first layer of the network. For the fully connected feed forward neural network we considered two hidden layers with the same number of neurons. The optimal number of neurons for these layers is studied in the following section. In the hidden layers we use the activation function ReLU which is a non-linear activation function.

We train our model using ADAM optimization algorithm, this learning method is very effective since it converges fast and provides good results compared to the classical gradient descent approach. Furthermore, as training progresses, the ADAM algorithm computes a loss function, the back-propagation of this function modifies the network in order to minimize the loss which leads to optimize the estimator.

In order to evaluate the performance of the model, we calculate the Mean Absolute Percentage Error (MAPE) over the test set. We target the estimation of each of the network conditions given Web metrics and page features.

IV. Performance Evaluation

A. Platform Implementation

Having introduced the general methodology before (see Fig. 1), we now focus on its implementation. In particular we have the Experimenter that executes the tests and starts the experiments by communicating with all the other entities. This Experimenter is composed of four parts. The first one developed in Python as a simple Finite State Machine (FSM), sets the network conditions and generates network configurations to experiment with. These configurations are enforced in the network through the network emulator *tcconfig*. The second part is built in a Web page that uses the Service API to control the experiment and collect statistics. The Web client is composed of two entities: the browser (we use Google chrome) and the extension; the Browser is responsible of loading the Web pages while the extension is a plugin developed in JavaScript to perform measurements based on *Chrome Navigation Timing API* and the *Performance Navigation API*, both being W3C recommendations. The resulting data are then retrieved by the Experimenter and stored in the database.

B. Results

Here, we compare the performance of our CNN-based network estimator with two ML approaches: Fully Connected Feed Forward Neural Network (NN) and Random Forest (RF).

a) Hyper-parameter tuning of the estimators: In order to fine-tune the hyper-parameters of the proposed estimator and the models with which we compare our solution, it was necessary to use various configurations.

For neural network models (i.e., CNN and NN), we opted for the Keras[2] platform to evaluate the estimation efficiency. We considered models of neural networks with two hidden layers for which we varied the number of neurons from 100 to 500 neurons in steps of 100. We have as input the Web parameters (Connect Start, DNS, Request, Response, DOM, First Paint, First Contentful Paint, PLT) plus other page features (Global size, Protocol, Number of objects) and the output is the estimated network metrics.

Fig. 10 shows the scatter plots of the estimated loss rate as function of the real one respectively for 100, 200, 300 and 400 neurons and for both NN and CNN. The same results were observed for the bandwidth and RTT metrics. We can see how the performance of both NN and CNN increases significantly when we add more neurons. Particularly, we obtain the higher accuracy when we reach 400 neurons. Beyond this value the accuracy starts to decrease. One can note that CNN clearly outperforms NN in predicting the packet loss rate.

The second model that we assessed is Random Forest (RF), which is a regression technique that consists in boosting multiple decision trees to create a more powerful model. To build the RF model we used the scikit-learn library [3]. The number of trees is a main parameter of RF. We changed this number from 100 to 700 in steps of 100 to study its impact. The metric Mean Absolute Percentage Error (MAPE) is plotted versus the number of trees for the Loss Rate metric in Fig. 11. Lowest error is achieved for 600 trees.

[2]https://keras.io/

[3]https://scikit-learn.org/

(a) Estimated loss rate versus real one for NN

(b) Estimated loss rate versus real one for CNN

Fig. 10: Comparison between NN and CNN using different number of neurons

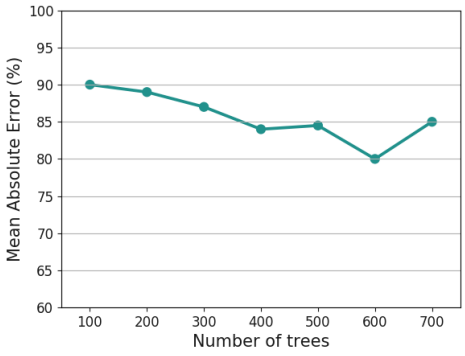

Fig. 11: RF performance versus number of trees for Loss Rate

b) CNN versus NN and RF: Now we compare the three ML techniques using their best tuning found above. Fig. 12 compares them to each other in terms of MAPE for different ranges of Network QoS metrics. Random Forest, despite their known power, shows the least estimation accuracy which can

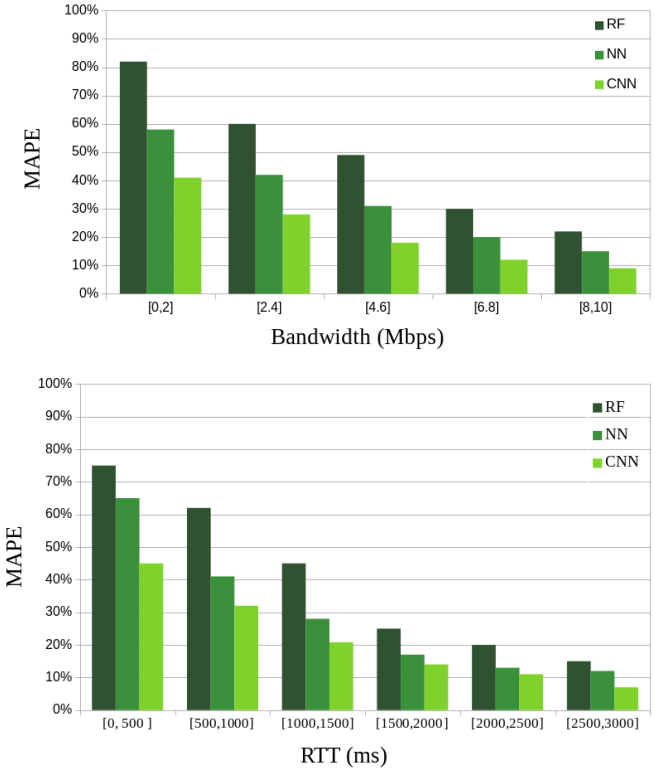

Fig. 12: CNN against NN and Random Forest

go up to 80% for low ranges. This illustrates the difficulty of the task. Fully Connected Neural Networks come next, then they are followed by CNNs, which show the best estimation accuracy. It can also be observed how for the three techniques, the relative accuracy improves when ranges get higher. In particular, when using CNNs, the error drops to less than 10% for a download bandwidth around 10Mbps.

V. Conclusion

We presented in this paper an efficient and novel method to estimate network performance metrics from Web metrics that can be collected passively and easily from within the browser. We developed a platform to collect our own dataset and designed a methodology around deep learning for network estimation and FAST method for sensitivity analysis. Our results underlined the difficulty of the task given the complexity of the relationship between the network and the Web rendering of a page. Only Convolutional Neural Networks were able to provide acceptable results, that can get as accurate as few percents for some ranges of the underlying network metrics. In future work, we will explore how the methodology can be exploited to detect network anomalies and provide hints on their root causes.

References

[1] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Characterizing web page complexity and its impact," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, pp. 943–956, June 2014.

[2] R. Schatz, T. Hofeld, L. Janowski, and S. Egger, *DataTraffic Monitoring and Analysis*. Springer-Verlag, 2013.

[3] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the quality of experience of web users," in *Proc. of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*, ser. Internet-QoE '16. New York, NY, USA: ACM, 2016, pp. 37–42.

[4] D. Da Hora, A. S. 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 - Int. Conf on Passive and Active Network Measurement*, Berlin, Germany, Mar. 2018, pp. 1–13.

[5] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," *Proc. of Passive and Active Measurement Workshop*, 03 2002.

[6] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson, "Fathom: A Browser-based Network Measurement Platform," in *ACM Internet Measurement Conference*. Boston, United States: ACM, Nov. 2012, pp. 73–86.

[7] "Navigation timing api," https://www.w3.org/TR/navigation-timing/.

[8] W. recommendation, "Paint timing api," https://www.w3.org/TR/paint-timing/.

[9] R. Schatz, T. Hofeld, L. Janowski, and S. Egger, *DataTraffic Monitoring and Analysis*. Springer-Verlag, 2013.

[10] M. Katsarakis, R. C. Teixeira, M. Papadopouli, and V. Christophides, "Towards a causal analysis of video qoe from network and application qos," in *Proc. of the 2016 Workshop on QoE-based Analysis and Management of Data Communication Networks*, ser. Internet-QoE '16. New York, NY, USA: ACM, 2016, pp. 31–36.

[11] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "Developing a predictive model of quality of experience for internet video," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 339–350, Aug. 2013.

[12] M. Fiedler, T. Hossfeld, and P. Tran-Gia, "A generic quantitative relationship between quality of experience and quality of service," *IEEE Network*, vol. 24, no. 2, pp. 36–41, March 2010.

[13] 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 *Passive and Active Measurement*, M. A. Kaafar, S. Uhlig, and J. Amann, Eds. Cham: Springer Int. Publishing, 2017, pp. 47–59.

[14] 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 *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, ser. Internet QoE '17. New York, NY, USA: ACM, 2017, pp. 13–18.

[15] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki, "Eyeorg: A platform for crowdsourcing web quality of experience measurements," in *Proc. of the 12th Int. on Conf. on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: ACM, 2016, pp. 399–412.

[16] T. Spetebroot, S. Afra, N. Aguilera, D. Saucez, and C. Barakat, "From network-level measurements to expected quality of experience: The skype use case," in *2015 IEEE Int. Workshop on Measurements Networking (M N)*, Oct 2015, pp. 1–6.

[17] M. J. Khokhar, N. A. Saber, T. Spetebroot, and C. Barakat, "On active sampling of controlled experiments for qoe modeling," in *Proc. of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, ser. Internet QoE '17. New York, NY, USA: ACM, 2017, pp. 31–36.

[18] P. L'Ecuyer, "Randomized Quasi-Monte Carlo: An Introduction for Practitioners," in *12th Int. Conf. on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing (MCQMC 2016)*, Stanford, United States, Jun. 2017.

[19] M. J. Khokhar, N. A. Saber, T. Spetebroot, and C. Barakat, "An intelligent sampling framework for controlled experimentation and qoe modeling," *Computer Networks*, vol. 147, pp. 246 – 261, 2018.

[20] S. Tarantola and T. A. Mara, "Variance-based sensitivity indices of computer models with dependent inputs: The Fourier Amplitude Sensitivity Test," *Int. Journal for Uncertainty Quantification*, vol. 7, no. 6, pp. 511–523, Apr. 2017.