Data Driven Network Performance Inference From Within The Browser

Imane Taibi, Yassine Hadjadj-Aoul, Chadi Barakat

Data Driven Network Performance Inference From Within The Browser

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

Chadi Barakat
Université Côte d'Azur, Inria, France

***Abstract*—The ability to monitor web and network performance becomes crucial to understand the reasons behind any service degradation. Such monitoring is also helpful to understand the relationship between the quality of experience of end users and the underlying network performance. Many troubleshooting tools have been proposed recently. They mainly consist of conducting active network measurements from within the browser. However, most of these tools either lack accuracy, or perform measurements to a limited set of servers. They are also known to introduce non-negligible overhead onto the network. The objective of this paper is to propose a new approach based on passive measurements freely available from within the web browser, and to couple these measurements to deep learning models to estimate the latency and bandwidth metrics of the underlying network without injecting any additional measurement traffic. We develop and implement our approach, and compare its estimation accuracy with the best known web-based network measurement techniques available nowadays. We follow a controlled experimental approach to derive our inference models. The results of our study show that our approach can give a very good accuracy compared to others, its accuracy is even higher than most standard techniques, and very close to the rest.**

***Index Terms*—Network measurement, web browsing, deep learning, controlled experimentation.**

I. Introduction

Monitoring network performance becomes a necessity today, especially for web browsing activities. On one hand, it allows network operators to evaluate and understand the quality they deliver to their customers. On the other hand, performing network measurements gives the user a better understanding of web performance and its relationship with network conditions such as connection speed and latency. Many monitoring platforms and tools have been recently proposed, that are in part standalone applications. But the need to monitor the internet performance from the end user point of view in a portable and easy way has also led to a new trend of network measurement tools that can run from within the browser itself. So, several web measurement tools have seen the light, either in the form of browser plugins and external modules, or in the form of embedded code within the visited Web pages. In particular, one can find Speedtest.net [1], a web tool using Flash and Java Applet to measure the ping (latency), the download speed and the upload speed. Netalyzr [2] is a Java Applet that probes many servers and provides information on the network and the access services. Fathom [3] is a firefox extension that allows performing a set of network measurements, both passive and active, using JavaScript.

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

When it comes to network performance monitoring, existing tools proceed mostly with an active approach by injecting dedicated probing traffic into the network. This way they are able to derive accurate measurements about the network performance, such as the Round-Trip Time (RTT) and the download speed. Speedtest is a typical example [1]. These tools are, however, known to incur cost on the network and the data plane of the end user, especially in a mobile setup, with possible interaction of the measurement traffic with the traffic of the other applications and users.

In this paper, we propose a new approach that is solely based on using the passive measurements freely available in the browser regarding the web browsing quality of service (e.g., Connect Start, Page Load Time [4], [5]) as well as some other page characteristics (e.g., number of objects). We passively capture these measurements and apply deep learning models to estimate the network state from them, without the need to probe the network. We will have then realized our two main objectives: (i) reduce to zero the cost of active measurements, and (ii) infer the path characteristics to the web server of our choice without the need of collaboration from a dedicated measurement server as it is the case with existing tools.

In more detail, we engineer a methodology to collect a large dataset that links web performance metrics to underlying network metrics. Then, we apply a Convolutional Neural Network model (CNN) to infer the network metrics, the justification of using CNN is given in our recent work where we show its performance against other machine learning techniques in modeling the complex relationship between network level and web level QoS [6]. Then, and for the purpose of comparison with existing web-based monitoring solutions, we propose an integrated framework where we implement our solution, and mimic the behavior of other solutions, to ensure a fair comparison between the different approaches in a fully controlled environment. Our results show that our methodology is able to provide very good estimation of underlying network performance, outperforming some existing solutions by times, especially when the network metrics are of large values. We believe this makes from our solution a viable, general and light weight solution for the continuous monitoring of the network from the traces of the web browsing activity of the user.

The rest of the paper is organized as follows. In section II, we discuss the related work. In section III, we describe our ap-

proach for training the deep learning model and for collecting the required dataset. In section IV, we present the integrated platform used for the comparison of our solution with existing solutions, as well as the results of our experiments and our main observations. We conclude the paper in Section V.

II. Related Work

A. Troubleshooting platforms and tools

Several measurement tools and approaches have been proposed for monitoring and troubleshooting the performance of the web ecosystem in the last few years. We can divide them into two main parts: network measurement platforms and web-based measurement tools [7], [8].

A network measurement platform is a server-based infrastructure dedicated to test and measure Internet and network performance. Basically, many researchers use PlanetLab [9] to support the development and testing of new network services, even if the latter is not really meant to be a measurement platform. Measurement Lab (M-Lab) [10], launched by google, also allows and facilitate this kind of task. However, these platforms can't be used in end systems. The necessity to perform measurements in end systems led to the development of new solutions: web-based measurement tools. Many of these tools have recently seen the light. For example, the Janc's method [11] provides multiple services based on Flash and JavaScript to measure the RTT and the throughput in a fully controlled way. Netalyzr [2], which is based on a Java applet accessible from a web page, provides measurements of the latency, the bandwidth and the buffering at the edge of the network. Speedtest [1], which operates mainly over TCP testing with an HTTP fallback for maximum compatibility, measures the latency, the download speed, and the upload speed. How's My Network (HMN) [12] is a website that provides predictions for common Internet activities. Table I gives a summary of main tools and the approach they follow. In this paper we position ourselves on the front of web measurements tools given their increased level of practicality and interest for the end user.

In general, the methods to measure the RTT and the bandwidth from within the browser consist of recording the time before and after retrieving an object from the web server. The measurement process used by these methods could be HTTP-based or socket-based. HTTP-based methods are implemented through JavaScript or Flash. One way of doing this is by using the Javascript function `Date.getTime()` in the date API, with DOM [13] to fetch the wanted resource. This technique is very simple and supported by all the browsers. Another method is to use the XHR (`XMLHttpRequest`) API [14] to request objects using JavaScript's browser environment. Alternatively, one can also use Flash, in particular the `URLLoader` class, which is able to download data from a URL as text, binary data, or URL-encoded variables. It is useful for downloading text files, XML, or other information to be used in a dynamic data-driven application. On the other hand side, socket-based methods are implemented through TCP or UDP connections to exchange binary data. Flash Action script for example, can create TCP sockets, using the following APIs: `Socket`, `SecureSocket`, `ServerSocket` as well as `XMLSocket`. One other way is the use of Websockets, which are based essentially on HTTP Request / Response. A WebSocket exposes the underlying TCP socket, used in an HTTP request, to the Application layer, and hence creates possibilities for application developers to communicate with the server in a full-duplex manner.

To summarize, most of existing web-based measurement tools proceed by the following steps [15], [16]:

- Step 1 – Initialisation phase –: The client downloads from the web server a container web page that contains the measurement code.
- Step 2: The client probes the web server for a period of time by sending "request" messages, to retrieve an object using a specific measurement approach. The requests are sent using a train of IP packets, the time just before sending the requests is recorded.
- Step 3: Here the web server returns a "response" message to the client, using a train of IP Packets (one or more), the time just after receiving the response is recorded.
- Step 4: Recorded times are then used to estimate relevant information about the network access performance such as the two-way delay and the available bandwidth.

III. Estimating delay and bandwidth from web performance metrics

Unlike traditional approaches that consist in generating traffic to measure network performance, our goal in this work is to obtain these measurements at almost no cost. Indeed, the approach, we propose, allows exploiting classical Internet browsing and the associated web performance metrics in order to deduce the state of the network. This estimation technique can be adapted not only for known web pages but also for unknown pages, as we will show later on. Here, we explain our approach in detail. As shown in Fig. 1 and 2, we divide our methodology into two main phases: i) data collection and processing phase, and ii) network estimation phase.

A. Data collection and processing phase

In order to predict network status departing from web-level measurements, we first collect a dataset that captures the link between network QoS metrics and web QoS metrics. Table II gives an overview of the considered network QoS metrics, web-level performance metrics as well as other web page characteristics used in this work for estimation purposes.

We proceed by extensive controlled experiments, where network configurations are artificially modified and measurements of both network and web browsing are collected. For that purpose, we develop a distributed system based on different entities to provide a platform linking the input (underlying network metrics) to the output (web performance metrics).

The "Experimenter" in Fig. 1 generates network configurations (Delay and Bandwidth), enforces these configurations using the *tc* tool in Linux – Step 1 –, and then launches the web client – Step 2 –. This component is in charge of returning

TABLE I: Main network troubleshooting tools and their approaches

Troubleshooting tool	Nature	Methodology	Approach
Speedtest.net	Website, plugin	Flash	HTTP
Janc's study	Native	JavaScript: XHR,DOM and Flash	HTTP
Netalyzr	plugin	Java Applet: TCP socket	Socket
HMN	plugin	Java Applet	Socket
NDT	plugin	Java Applet	Socket
Fathom	plugin	Java Script	HTTP
Navigation timing API	Native	DOM	HTTP

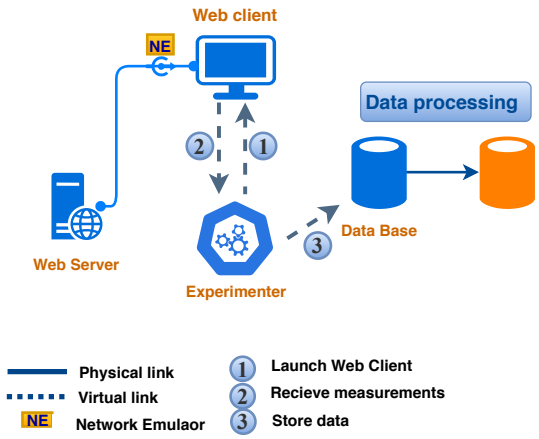


Fig. 1: Data collection and processing phase

TABLE II: Web and Network performance metrics

Network metrics	RTT – round trip time –
	Bandwidth – maximum end-to-end throughput –
Web metrics	Connect Start
	DNS
	Request
	Response
	DOM
	First Paint (FP)
	First Contentful Paint (FCP)
	Page Load Time (PLT)
Page characteristics	Page Size (Size)
	Number of objects (NumObjects)
	Protocol supported (Protocol)
	Median of objects size (Median Objects)
	Total objects size (Objects' Size)
	First Quantile of object's size (Q1 Objects)
	Third Quantile of objects size (Q3 Objects)
	Maximum objects size (Max Objects)
	Minimum objects size (Min Objects)

a valid experimentation scenario. It is in part developed in Python as a simple Finite State Machine (FSM) that sets the network conditions, and is built in another part built in a web page to control the experiment (loading of web pages).

The "Web Client" is composed of two entities: the browser (in our case Google Chrome) and the extension. The browser is responsible for loading a specific Web page, while the extension developed in JavaScript, collects all the information that we need to build our model using "Chrome Navigation Timing API" and "Performance Navigation API", which are w3c recommendations. The web page and the experimenter communicate through a Websocket via custom commands. The collected data are stored in – Step 3 – (see Fig. 1).

Network configurations are generated using the FAST method (Fourier Amplitude Sensitivity Test [17]). FAST is a sampling methods that covers a given space based on relevant frequencies, which allows an efficient scan of the area to be sampled. Once a scenario is selected and instantiated in *tc*, traffic generated by the service goes through the emulator that enforces the network conditions as defined by the scenario. It has to be pointed out that delay and bandwidth control are only applied to the download web traffic.

As introduced before, our approach is based on lab experiments, where we aim to have under our control the network conditions. This is unfortunately not the case in practice as at least the parts of the ISP and the web servers are out of our control, even if we are connected to the Internet via a high-speed connection and we visit well-engineered websites. In order to handle the noise coming from the uncontrolled, we ensure the validity of the scenarios through connectivity tests before calibrating the network emulator. In particular, we perform (i) TCP throughput test to ensure that the available bandwidth is higher than the one we want to enforce, and (ii) RTT noise estimation tests to deduce the network delay from the emulated one. The dataset obtained applying our methodology is composed by 5000 entries corresponding to 5000 different network scenarios for each of the 500 top popular web pages according to alexa ranking.

Before starting the estimation phase, we should first be sure that the collected dataset fits the estimator well. To do so we perform data cleaning by removing missing and erroneous values, then we perform a data transformation that considers as input the web performance features and as output the network metric (i.e. RTT or bandwidth). Finally, we split our dataset into training and test sets; for that purpose we pick 70% of network scenarios and 80% of web pages randomly as training set, and we consider the rest as test set.

B. Network performance estimation

Several analysis techniques are used to build estimators. One way is to use Convolutional Neural Networks (CNNs) – known

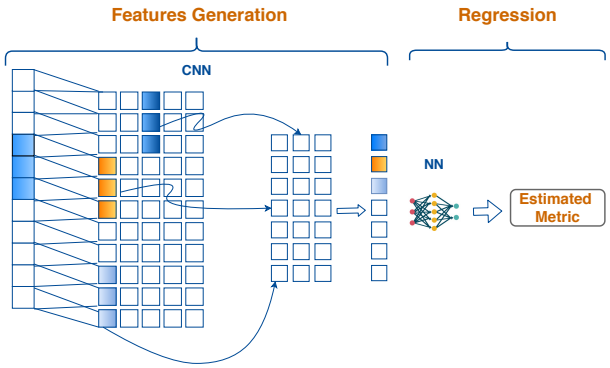


Fig. 2: Estimation phase

by their ability to automate the features' extraction process – , followed by fully connected neural networks, which allow a high level of precision and convergence in the prediction, especially in the presence of large datasets. Based on the results of our recent work [6], we follow the CNN approach, as it has shown the highest accuracy compared with traditional Machine Learning techniques such as Random Forests. We use CNN regression to build a model that predicts the underlying network state and that considers as input the web download metrics of a page together with information on its content (as summarized in Table II). The model gives as output the estimated network metric: RTT or download bandwidth. So basically we have two CNN regressions.

Fig. 2 shows our CNN model which consists of two 1-dimensional convolutional layers, with a number of filters equal to 100, and a kernal size of 3, followed by a Max pooling layer. In the middle, we have two hidden layers of a fully connected neural network with 400 neurons each. We use ReLU as an activation function, which is a simple non-linear activation method. We don't use any activation function for the last layer, since it is a regression and we are rather interested in estimating raw values without transformation. Further, we use the ADAM algorithm during the training in order to optimize the loss function [18]. Finally, we calculate the Mean Absolute Percentage Error to evaluate the performance of our model.

IV. Performance Evaluation

A. Integrated platform implementation

In this section, we give a detailed overview of the platform used to compare our approach with other web-based monitoring solutions. For that, we build an integrated framework where we try to mimic the behavior of the best known troubleshooting tools and services, by implementing and simulating their technical behavior. The platform also integrates our own solution, which is based on the joint use of passive measurements and deep learning. We design, implement and evaluate measurement techniques specialized in measuring the network download throughput and the two-way network delay (RTT). As mentioned in Section II-A, existing techniques share the same common behavior consisting of sending and receiving messages between the web client and the web server. They implement, however, this probing process in different ways. Departing from this observation, we enrich our experimental platform with an automated process that allows to implement multiple measurement methods including Flash, DOM, XHR, Java Applet, and web socket, see Table I, each of which embedded in a PHP or HTML index page. The web client requests the container page, renders its elements and executes the measurement code of the different techniques. This latter step leads to the measurement phase. Finally, the obtained measurements are stored on a dedicated server.

TABLE III: Implemented techniques

	Implemented techniques
HTTP methods	JavaScript: XHR
	JavaScript: DOM
	Flash
Socket methods	Java Applet TCP socket
	Web Socket

We designed a testbed consisting of a web client, an intel core i7, with 32 GB memory, and a local Web Server squid 3.4.7 where we cached 500 landing pages (see Fig. 3). The reason we cache the pages locally is to provide a fair environment for the comparison of the different techniques, otherwise we will get results depending on the location of the server specific to each of the techniques. Another advantage is that we remove the noise coming from the parts of the network that we don't control. In such a fully controlled environment, the platform executes the RTT and bandwidth measurements by doing the following : i) setting a tuple of (RTT, bandwidth) using the network emulator *tc*, ii) checking and validating the values using *ping* tool for latency and *iperf* tool for bandwidth, iii) starting the initialization phase, where the web browser renders the container page, iv) launching the measurements for each scripting technique including our approach, v) lastly, storing the data. Tests were repeated for each fixed tuple of (RTT, bandwidth) 50 times to retrieve 500 web landing pages at each time, then mean values are calculated.

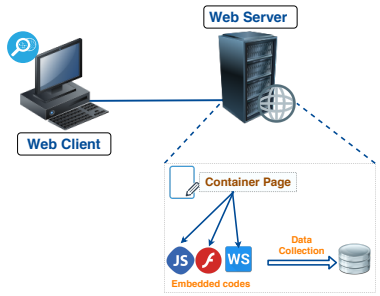


Fig. 3: Integrated validation platform

B. Results

Here, we compare the performance of web browser-based solutions (see Table III) with our own solution as a function of their capacity to estimate the ground truth on RTT and

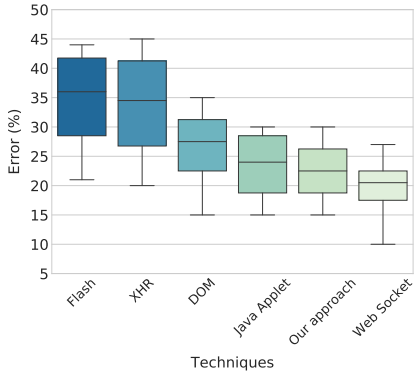


Fig. 4: RTT error of implemented troubleshooting techniques

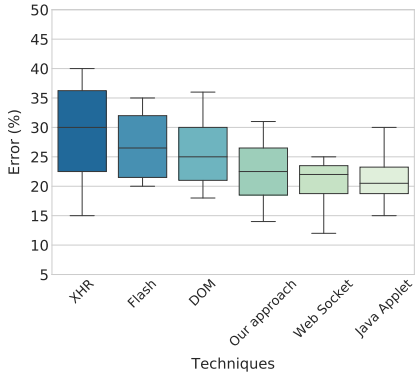


Fig. 5: Download bandwidth error of implemented troubleshooting techniques

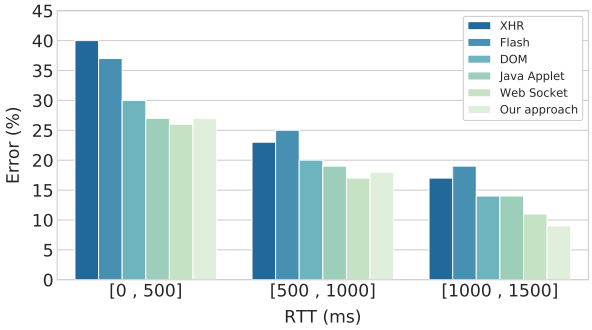


Fig. 6: Error of implemented techniques in terms of RTT

download bandwidth. For the comparison, we consider scenarios where the RTT varies between 0 and 1500 ms and the download bandwidth between 0 and 3 Mbps.

The boxplots in Fig. 4 and 5 display, respectively, the dispersion of error over web pages using the implemented techniques for both metrics: RTT and download bandwidth. The histograms in Fig. 6, and 7, compare the implemented solutions to each other for different ranges of RTT and download bandwidth. In all these figures, the y-axis shows the mean percentage error given by the following equation:

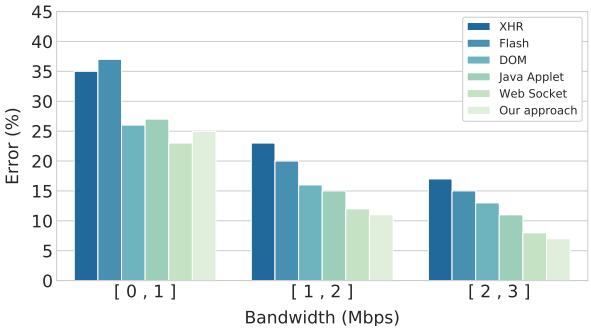
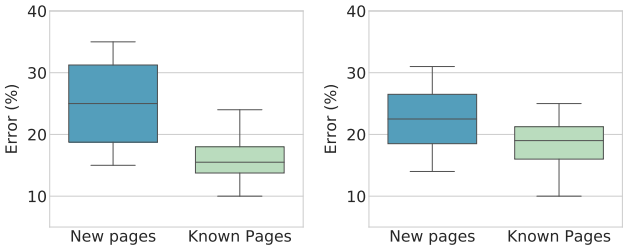


Fig. 7: Error of implemented techniques in terms of download bandwidth



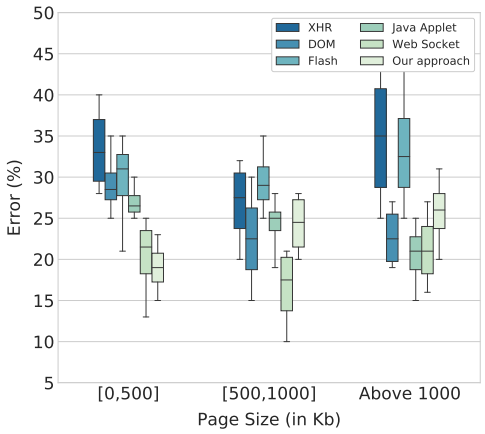
(a) RTT (b) Download bandwidth

Fig. 8: Comparison of performance for estimating RTT and Bandwidth between pages we know and new pages

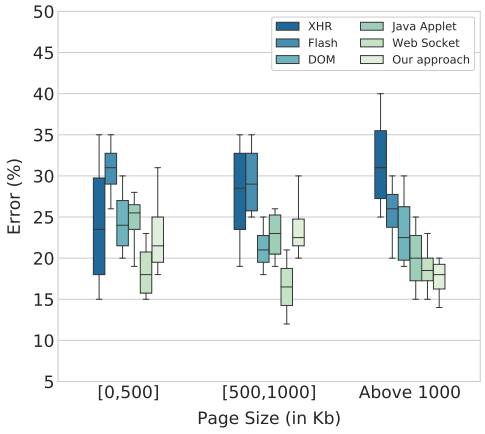
$$Error = 100\% \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Estimated\ or\ Measured\ Value - Real\ Value}{Real\ Value} \right| \quad (1)$$

We notice a significant gap between the different techniques. In general, HTTP-based methods including XHR, DOM, and Flash show the least estimation accuracy for both RTT and download bandwidth. For XHR and Flash techniques the estimation error is very high, and can go up to 40%. DOM technique achieves better results than the latter ones but the error still cannot be neglected. On the other hand, socket based solutions are able to provide the best overall accuracy. We can explain this by the fact that HTTP methods add extra headers to retrieve objects from the web server thus increasing the overhead of the solution and reducing its accuracy.

Our approach based on passive measurements and deep learning outperforms most of the implemented solutions, as it ranked the second after the web socket technique for RTT and the third for the download bandwidth, but with a very small difference up to 3% if we consider the mean over all web pages. Moreover, it ranked the first for large values of RTT and download bandwidth (see Fig. 6 and 7). In particular, the error drops to less than 10% for a download bandwidth around 3 Mbps. Another important result concerns the set of web pages used in the estimation. Previous results are for a validation on a different set of pages than the one on which we train the learner. In fact, as shown in Fig. 8, if we validate



(a) RTT



(b) Download bandwidth

Fig. 9: RTT and download bandwidth estimation error in function of web page size

our solution over the same set of pages, the accuracy improves significantly for both RTT and download bandwidth.

Lastly, we check whether the estimation accuracy varies between pages based on their content. Indeed, some pages might be more suited than others for network performance estimation. We consider the impact of the page size in Fig. 9. We show boxplots displaying the estimation error as a function of the page size for RTT and download bandwidth metrics. The error is also calculated for the other measurement techniques. We can notice how the performance of our approach increases significantly for certain intervals of page sizes, and can even outperform all the other solutions. This for example occurs when the page size is under 500 KB for the RTT (Fig. 9a) and above 1000 KB for the download bandwidth (Fig. 9b). These results give hints on how to choose web pages to achieve the best possible accuracy with our approach.

V. Conclusion

In this paper, we present a new approach to infer network performance, based on passive measurements freely available from within the web browser, and deep learning models to predict RTT and download bandwidth. We develop and implement a methodology consisting of two phases: *(i)* data collection phase where we vary the network conditions and collect web-level measurements, and *(ii)* estimation phase, where we apply a convolutional neural network model (CNN) to estimate network state. Then, we propose an integrated validation platform where we implement our approach as well as several other web-based monitoring solutions. Results show that our approach can give a very good accuracy compared to others, its accuracy can be even higher than most standard techniques. We will keep exploring how we can make this lightweight and accurate monitoring solution detect network anomalies and pinpoint their root causes.

References

[1] Speedtest, https://www.speedtest.net/.
[2] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the edge network." Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, 01 2010, pp. 246–259.
[3] 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.
[4] 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.
[5] 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.
[6] I. Taibi, Y. Hadjadj-Aoul, and C. Barakat, "When Deep Learning meets Web Measurements to infer Network Performance," in *CCNC 2020 - IEEE Consumer Communications & Networking Conference*. Las Vegas, United States: IEEE, Jan. 2020, pp. 1–6. [Online]. Available: https://hal.inria.fr/hal-02358004
[7] V. Bajpai and J. Schönwälder, "A survey on internet performance measurement platforms and related standardization efforts," *IEEE Communications Surveys &amp Tutorials*, vol. 17, 04 2015.
[8] V. TONG, H. A. TRAN, S. SOUIHI, and A. MELLOUK, "Network troubleshooting: Survey, taxonomy and challenges," in *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, Oct 2018, pp. 165–170.
[9] PlanetLab, https://www.planet-lab.eu/.
[10] M-Lab, https://www.measurementlab.net/.
[11] A. Janc, C. E. Wills, and M. Claypool, "Network performance evaluation in a web browser." proceeding of IASTED PDCS, 2009.
[12] A. Ritacco, C. Wills, and M. Claypool, "How's my network? a java approach to home network measurement," in *2009 Proceedings of 18th International Conference on Computer Communications and Networks*. 10.1109/ICCCN.2009.5235346, Aug 2009, pp. 1–7.
[13] DOM, https://dom.spec.whatwg.org/.
[14] XMLHttpRequest, https://xhr.spec.whatwg.org/.
[15] W. Li, R. Mok, R. Chang, and W. Fok, "Appraising the delay accuracy in browser-based network measurement," 10 2013, pp. 361–368.
[16] C. Zhang, X. Hei, and W. Cheng, "Performance evaluation of web-based cloud services in a browser-scripting approach," vol. KSII Transactions on Internet and Information Systems.10, pp. 2463–2482, 06 2016.
[17] 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.
[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.