WePR: A Tool for Automated Web Performance Measurement

Alemnew Sheferaw Asrese*, Pasi Sarolahti*, Magnus Boye‡ and Jörg Ott§
*Aalto University, Finland *Tuxera Inc, Finland §Technical University Munich, Germany

***Abstract*—In this paper, we present *WePR*, a web performance measurement tool for evaluating the download and the rendering time of a website at large scale. The first component of the tool, WebPerf, measures the technical metrics of a website such as DNS lookup time, TCP connection establishment time, page download time from different vantage points. The second component renders the website based on the metrics recorded by the WebPerf and measures the rendering time of the website in a browser window. The user experience on a website is then inferred from the measured rendering time. Along with describing *WePR*'s architectural design and the metrics it measures, we present initial measurement results recorded March to August 2015.**

The results show that transmission latency, DNS lookup time, the number of HTTP elements affect the rendering time of the website and therefore the end user experience.

I. Introduction

There are well established long-standing metrics for measuring Internet communication performance, such as TCP throughput. Most of those measurements are often taken from packet traces near the network interface, which gives an accurate point for measuring communication throughput, round-trip latencies or packet loss behaviour. However, for different networked applications such as web browsing the actual performance depends on several different factors than just the raw communication performance.

Waiting time is the key factor of the end user experience in most Internet applications such as web browsing, video streaming. The shorter the waiting time to get the contents is the more the user is satisfied with the service [6].

This paper proposes a way to measure a metric closer to user perceived performance when accessing complex resources on the Internet. A web page is composed of multiple objects that are downloaded based on separate HTTP requests, from different sources and using multiple TCP connections. In addition to the main HTML page that provides the overall page structure, other objects also need to be downloaded before the page shows correctly to the user. A web page may refer to components that may not have a significant importance to the user such as advertisements.

In the past, researchers have proposed and analysed various improvements to TCP performance in different scenarios (reordering [4], packet losses [7], startup performance [5]), content delivery mechanisms, etc. But these do not reflect all the performance aspects of the web from the user perspective. In this paper, we focus on the overall web page rendering performance, which is more significant metric than, for example, the performance of individual TCP connections that are used to download the web page.

We are not only interested in the effect of object download times, but will analyze the overall web performance, including the time it takes to render a complete page. Therefore we need to use actual web browser rendering engines to measure the real rendering performance.

There are browser-based tools for analyzing web performance from a given host as manually operated by a user, e.g. http://www.webpagetest.org. However, manual operation is not possible if we want to collect data at a large scale from multiple measurement points located at actual home premises 24/7. Instead, we need a system that automatically collects the performance data for a given set of web sites in a continuous fashion, and delivers the measurement results into a centralized storage for further analysis.

We developed an automated measurement system, called *WePR*, to measure the web performance as perceived by the user. The system emulates the communication events involved in a web browsing session, including DNS lookups, setting up and terminating the TCP connections and HTTP requests, and downloading the content in different connections as an actual web browser would do. It measures the time used in different stages of the communication in a detailed manner. Because the measurement boxes are often embedded devices with limited processing capacity, they cannot actually render the web page locally. Instead the *WePR* system replays the web rendering process at a dedicated server in a (headless) browser display using the measured communication time at each measurement point. This all happens automatically, in a distributed fashion, enabling an efficient collection of the web rendering data from a large number of measurement points for further analysis.

During our initial trial, we deployed the measurement software in OpenWRT based Linux boxes. The boxes are located at home user premises[1] next to the Internet gateway, and continuously run measurements to a defined set of websites.

In the rest of this paper we give a short overview of the related works, after which in Section 3 we describe the WePR measurement system. In Section 4 we presents the metrics we are interested to measure. In Section 5 we discuss the results from our initial trials consisting of 50 measurement points, and Section 6 provides concluding remarks.

II. Related work

W3C Web Performance working group [2] standardizes the web performance measurements and APIs in web browsers. The group has specified several web performance measurement APIs. Among those, the Navigation Timing, the Resource Timing and the User Timing APIs help to measure the performance of a website on a real world Internet connection.

[1]This work was done in collaboration with SamKnows Ltd., who produces and deploys such "measurement probes" to thousands of homes in different parts of the world.

978-1-5090-2482-7/16/$31.00 ©2016 IEEE

Also the Page Visibility, Efficient Script Yielding and Display Painting Notification APIs provide basic information about the rendering state of the web page and help the developers to write a resource (CPU and power) efficient web applications. For instance, the Page Visibility API enables a developer to determine the current visibility of the page. These APIs help to measure the performance of websites in the browser, but they cannot substitute the visual perception of the end user.

WebPageTest [3] is an online tool that enables Internet users to dissect a complete web page loading time with different test browsers, from multiple locations with different network characteristics. It measures the page loading time by dissecting the page load session into the DNS lookup time, TCP connection establishment time, SSL handshake time, HTTP response time, etc. The tool also measures the resources consumed during the measurement. However, *WebPageTest* is an interactive tool that is not suitable for large-scale measurements.

Kreibich *et al.* developed *Netalyzr*, a web based network measurement and debugging tool that lets the Internet users to evaluate their internet connectivity. It tests different properties of users' network access, such as use of IP addresses and address translation, DNS resolver fidelity and security, TCP/UDP service reachability, and so on [9]. *Netalyzer* does not particularly focus on web performance.

Joumblat *et al.* implemented an end host measurement tool — *Host View*, for collecting network performance data annotated with the users' perception in the network quality. The tool collects information about the traffic and network performance statistics, application level context, and system performance and environmental data such as network type. The data collected is used for characterizing end host diagnostic problems that combine the performance and the perception [8].

Sundaresan *et al.* developed a headless web client that can be deployed on home router devices to measure page load time. The tool downloads the home page of the web site and parses it to determine the static objects that are required to render the page. It separates the page load time into DNS look time, TCP connection time, download time for each object, etc, but does not emulate the real browser performance [11], unlike *WePR*

III. *WePR* DESIGN AND ARCHITECTURE

Our goal is to develop a scalable measurement tool for assessing web performance as perceived by user, based on the data collected from a number of measurement probes located at users. Because the probes are either lightweight and cannot run a full web browser or machines which we do not have full control of, we run the browser engine at server-side machines, relying on network measurements on individual web object downloads, as performed at the probes. Figure 1 illustrates the overall architecture of *WePR*. In this section we will discuss the main components of the tool.

A. Parsing server

The parsing server parses the web page of a given URL and lists the URIs of each element (i.e., all the embedded objects and those generated by JavaScript executions) that make up the website. It takes the website's URL and sends HTTP request (#1 in Figure 1) in a headless browser called *PhantomJS*, and records all the URIs of the objects downloaded as a result of the request. The Parsing server also executes the JavaScripts content and resolves the URIs of the objects generated by the execution of the script.

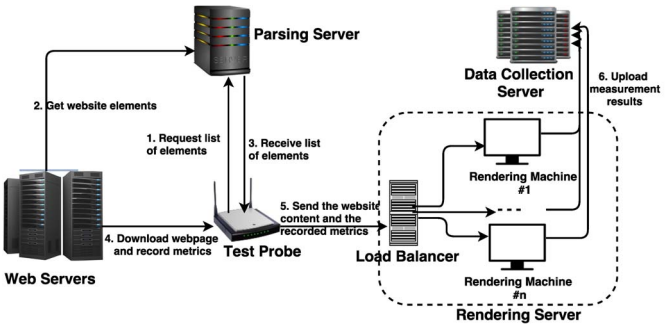

Fig. 1. The distributed architecture of *WePR*.

Once the URIs are generated, they are transferred to the test probes (#3 in Figure 1) which will perform the actual communication performance measurements for DNS resolution, TCP connection management and HTTP downloads for each object. The network measurements are then collected back at the server to assess the overall web rendering performance.

The parsing server can be within the probes if they are able to run (headless) browsers.

B. Test Probe

Probes are measurement boxes, such as small embedded OpenWRT Linux devices, which execute the WebPerf test software. WebPerf records the DNS and HTTP metrics, TCP statistics, downloads the objects that make up the website (based on the list of URLs it received from the parsing server) and pushes them to the Rendering Server for actual performance analysis (#4 and #5 in Figure 1, respectively).

The probes specify the number of TCP connections opened and the maximum number of simultaneous connections from the same domain. It uses pipelining techniques to send multiple HTTP requests in a single TCP connection. The WebPerf test software can be cross-compiled for any architecture and deployed to run in any Linux platform such as workstation or mobile devices.

C. Rendering Server

The Rendering Server calculates the page rendering time, the most important web performance factor for the end user. The server comprises two components: a load balancer and one or more rendering machines. The load balancer receives rendering requests from the test probes and distributes them to the available rendering machines using a round robin mechanism.

The rendering machines execute two different applications: (a) the Playback module, and (b) the rendering manager module. The playback module emulates a DNS server and

an HTTP server, and replies to the locally made requests, throttling down the response time and transmission rate to emulate the network delay shown at the probe based on the measurements from probes fetching the respective web objects.

The rendering manager module computes the perceived page load time of the website as it would have been seen by a user with the same network connection performance of the probes. It replays the web page download session in a (virtual) web browser based on the rendering requests received from probes. The server-side HTTP request is handled by the playback module as described above. The rendering manager computes the progression of a web page download session by calculating pixel changes in rendered web image shown in the browser window in 100ms intervals. When there are no changes in the rendered part of the web page, we conclude that the page is complete and can assess the page download performance, as visible to the user.

Determining when the page is complete can be difficult if there is changing content in the web page. Our current solution is based on the observation that many of the images change less frequently than every three seconds [2]. Therefore, once the page has started loading and if there are no changes for three seconds, the page is declared complete and the page load time is the time where we see the last change.

We choose the distributed architecture for the following reasons. First, often the probes at the customer premises have limited resources to execute the parsing server and the web rendering test. Second, assuming a probe with sufficient resources, there is no way to control the user's machine to configure as we want it to be and install the software we need such as ImageMagick. For instance, in the web rendering test every DNS and HTTP request is redirected to the local host and served by the service we developed. We would not have the privilege to set the users machine to redirect those requests to the local machine. Therefore, offloading the different functions into different machines is necessary.

One may wonder that the rendering done at the Rendering Server and at the probes on the customer premises are indeed the same. For example, the probes may access the website in a slow network connection and the rendering server accesses it from the local machine. Since the playback service module of the Rendering Server respond to any DNS and HTTP request according to the recorded WebPerf measurement results and therefore we are certain that the rendering done at Rendering Server and at the probes is exactly the same.

IV. Metrics

The components described in the previous section use different metrics to measure the web page performance and the user-perceived web rendering time. In this section we describe those metrics and the methodologies used to measure them.

[2]In this work, our assumption is for fixed networks and the users will not be patient to wait longer if nothing has changed for three seconds.

A. Web Performance Test (WebPerf)

WebPerf is the application that runs on the probes and measures webpage download performance. All the objects that make up a web page, including those generated by JavaScript, are downloaded and a set of metrics are recorded. The metrics include DNS lookup time, the number of messages exchanged during DNS lookup, the time to establish TCP connections, the time to perform SSL handshake, HTTP header size, the page download time, and TCP statistics. The following are the list of the most important metrics that affect the perceived user experience and recorded by the WebPerf.

DNS lookup time: The time it takes to resolve the domain name of an element's URL into an IP address. Any delay in the DNS lookup phase affects the rest of the web object transfer and therefore increase the page load time of the website.

TCP connect time: The time it takes to establish a TCP connection between the client and the web server where the element is hosted.

TLS handshake time: The time taken by TLS handshake when a secure connection is used between the client and the server.

HTTP download time: The time it takes to download an element of a web page. The download time of an element is measured from the time of the first request is sent until the final response is received. A delay in one of the elements, especially if the object is the first element to be downloaded for instance the HTML file that contain the layout of the website, affects the total page load time. The impact of the delay in the embedded objects of the website is low in comparison with the first element of the website to be downloaded [10].

Number of DNS iterations: The number of iterations done while resolving the URL of the element into an IP address.

Number of HTTP redirects: The number of the URL redirection done while retrieving the elements of a website. URL redirects cause a page to load slowly because it adds extra time to resolve the new URL into an IP address and make a TCP connection to the new address. The redirects could be done at the server side which is cacheable and hence it is relatively faster. It could also be done at the client side which is not cacheable and it makes the process much slower.

Number of HTTP(S) elements: The number of web objects that makes up the website. It counts objects that are accessed both in secured and plain connection.

Object size: The size of each element in Bytes that make up the website.

Page load time: The total time it takes to download the content of the website. The (network) page load time is the combination of the DNS look time, the TCP handshake time and the time taken to download all the elements of the page such as the HTML files, JavaScript, CSS and images.

HTTP redirect time: The time taken as a consequence of the HTTP redirection which includes the DNS lookup time, the connect time, and the pre-transfer and transfer time from the new address.

The WebPerf test takes the URL of a website and sends a request to get the list of URLs of the elements that make up the

website. Once it gets the URL of each element of the website it measures all the metrics mentioned above and at the same time it downloads the content. We developed our own library called HURL[3] for handling the HTTP downloads and extract HTTP related information. HURL was specifically designed to allow developers to intercept request processing at key points in the HTTP download process. HURL provides hooks close to TCP socket operations, which makes performance measurements more precise than what for instance cURL could provide. Additionally, parsed HTTP protocol information used by HURL during the download process is also exposed via hooks. WebPerf uses all of these hooks to record timestamps, extract header information, and save the received data.

WebPerf overrides the default DNS resolver of the host system and utilizes our own DNS client for name resolution. Our DNS client measures the delay caused by the DNS resolution when a new connection is established. It also records details about the resolution process and the final query result. Understanding how DNS performance is essential to reduce the connection setup time and user waiting time. A common DNS cache is used for all queries and DNS metrics are extracted using the hooks of the DNS client. During a web page download, domain name resolution is performed once per domain and only one of the several elements will have metrics of the DNS resolution. All the other elements from the same domain will include a reference to the element which triggered a DNS resolution and therefore contain the DNS metrics.

B. Web Rendering Test

While the WebPerf test measures the technical metrics of a website, Web Rendering test measures the time takes to show different visible portion of a website in a browser window. This web rendering time is used to infer the web quality of experience of a user. The downloaded contents by using the WebPerf test are used to recreate and replay the website from the local machine and render it into a browser screen such that the perceived page load time of the website can be calculated. The websites are replayed based on the recorded performance metrics so that the rendering process emulates a user who surf a web page with the same network connection characteristics on which the WebPerf measurement is done. The web rendering test measures the rendering time by looking at the changes in the visible part of the website on the browser. For computing the rendering time we look at the pixel changes in every 100ms[4] interval on the browser screen for 15 seconds. If no pixel change has been observed in 30 consecutive screen shots/ frames (that is, no rendering event is observed for 3 seconds), then we declare that the website reaches a stable stage and we took this as the point where the website is fully rendered. In some cases, the pixel change continues and the website may not reach to a stable stage within 15 seconds. This might be due to a very frequent animation in the website, or because the website takes too long to load or render the content. In such cases, the usability of the website is questionable.

[3]https://github.com/wperf/webperf/

[4]The normal human visual system do perceive changes between 150ms and 200ms and we believe that 100ms interval is quite fair.

V. Results from initial trials

A. Measurement setup

We cross compiled the WebPerf test and deployed in OpenWrt devices provided by SamKnows Ltd. The test has been deployed on more than 50 test probes located in different ASes mostly in Europe. Each probe is set to measure 4 websites (www.bbc.com, www.ebay.com, www.sina.com.cn, and www.reddit.com). The measurement runs every four hours. We selected these websites based on Alexa's top list and those which do not need user interaction or authentication to show significant content. These websites are from the categories of the most widely used Internet applications like News, e-commerce, and social media. Further more, the design style such as font types used in the web content, and the hosting locations (that is, in relative to the distance to the measurement vantage points) are also taken into consideration while choosing those websites.

The Rendering Server is running in a cluster of servers, which have virtual machines (VMs) inside. We used three VMs to run the rendering server. Each VM runs Linux (Ubuntu 14.04.2 LTS , CPU: 4 cores - Intel Xeon(R) 2.65GHz , and 4 GB RAM). We set the system configuration of the DNS resolution service to the local host as described in the previous section. We used Google Chrome browser (Version 46.0) for rendering the websites.

B. Performance and scalability of WePR

The rendering server emulates the rendering process of a real web browser by running a virtual screen that can be deployed in a server host, or in a virtualized cloud infrastructure. This allows parallel analysis of multiple measurements and more efficient use of available resources, which is necessary when analysing multiple web sites over multiple measurement points. We used "Xvfb", a display server which performs all graphics operations in memory without showing a screen output. A browser automation tool called "Selenium" [1] is used to start a browser virtually. Selenium allows us to take periodic snapshots the browser window state so that can we can track the changes in the window while the web page download is progressing.

The downside of our approach based on frequent snapshots over the rendered web page is that is hungry for CPU and memory resources. The progress of rendering process is measured by comparing the differences between two consecutive screen snapshots. Such pixel-wise comparison operation is heavy, and modern web browser rendering engines are not light-weight, either.

We measured the CPU and Memory usage in our test environment. Figure 2 and 3 show the CPU and memory consumption by a single rendering process. Approximately 80% of 5500 web rendering processes took nearly 1 minute of CPU time and 400 MB of memory.

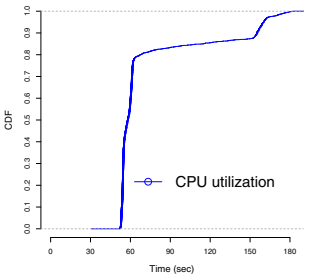

Fig. 2. The CPU utilization by a single rendering process.

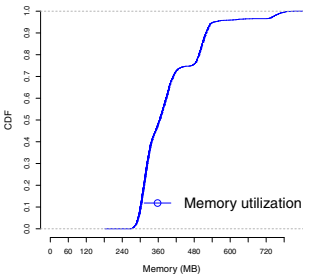

Fig. 3. The memory utilization by a single rendering process.

The most resource consuming part of the Rendering Server operations are the screen recording and the image processing to calculate the perceived page load time. That is, currently we record a video of the web browsing process for 15 seconds and then the video is converted into a sequence of images with 10 frames per second frequency. Improving the screen recording method would enhance the performance of the Rendering Server, and investigating more efficient performance analysis methods is a subject of ongoing work.

Another performance issue in the distributed architecture of *WePR* is that after the probes download the website content to measure the communication performance, they upload the results and downloaded data to the Rendering Server for analysis of the rendering performance. This significantly increases the amount data traffic between the measurement probe and the rendering server.

To validate our approach and the results we measured the rendering time of a website when the contents are downloaded by real browser and by our WebPerf software from the same computer. We download the content first by our WebPerf tool and immediately by a Chrome browser. We run this experiment for 1000 times. As shown in Figure 4 the rendering time of the website have almost similar behaviour in both cases. Only in about 30% of the cases the web site is rendered faster when the contents are downloaded by WebPerf. Furthermore, we did a manual check on web page download behaviour, appearance of the web page. Our tool behaves exactly the same way as the real browsers do to show the contents to the user. For example, during browsing a web page the web page can instantly appear, a broken page can display in browsers, or the browser window can be just a white until further reloading. We verified that all these browsing behaviours exist in our crawling tool.

C. Web rendering performance

The test has been running since March 2015. The data we present in this paper is from March 23 to August 11 2015, a total of about 50,000 data points. In this paper we report only few initial results to show that such a methodology can be used to measure the web QoE objectively. The rendering test measures the time taken to render different portion (50%, 80%, 100%, etc) of the visible portion of the web page.

The rendering behaviour in most websites is that after an initial download period most of the visible content is shown almost instantly. Additional web objects are also downloaded latter, but they do not significantly affect the visible web content. In contrast we observed that there are some cases on which the websites take longer to reach a stable state, that is where the pixel change between frames stop. One reason for this could be the page may have animation images which change frequently such as advertisements.

As shown in Figure 5 the measured website download time is much higher than the actual rendering time of the visible portion of the website. The reason for this could be the browsers keep the TCP connection open or additional objects which do not change the website appearance are being downloaded.

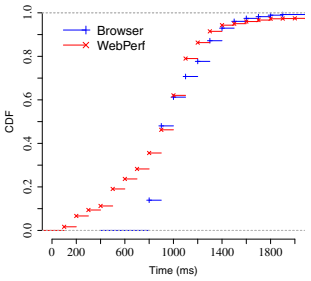

Fig. 4. The rendering time of the websites when the contents are downloaded by real browser and WebPerf software.

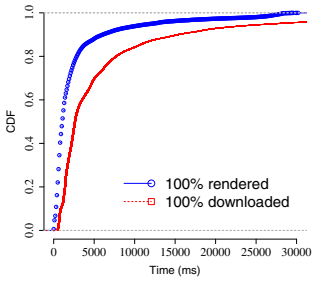

Fig. 5. The download time and rendering time of the websites.

The download time of the web pages varies substantially. Some websites have fewer web objects to download and therefore completes faster. Instead, other websites take longer to finish downloading because the number of objects to fetch are too many. The median web object counts per websites through out the measurement period are as follows:

Website	Median object count
www.bbc.com	65
www.ebay.com	178
www.reddit.com	32
www.sina.com.cn	122

In addition to this, HTTP redirects during the download process contribute to increase the download time. EBay.com involves 6 redirects and have large download time, whereas the others at most involves a single HTTP redirection. This shows that HTTP redirects affect the web page download time and therefore reduce the quality of experience of the end user.

Another interesting observation is the distance between the probes and the web server affects the rendering time. For example, www.bbc.com website is hosted at Tadworth in the UK, while the Chinese website www.sina.com.cn is hosted in Beijing China. Since most of the test probes are located in Europe the effect of the server location on the download time and web rendering time is clearly visible in our results as shown in Figure 6. Figure 7 illustrates the median latency in HTTP body transfer. EBay.com and sina.com.cn have the

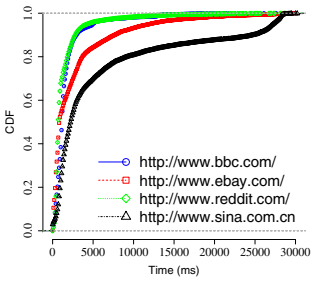

Fig. 6. Rendering time of different websites.

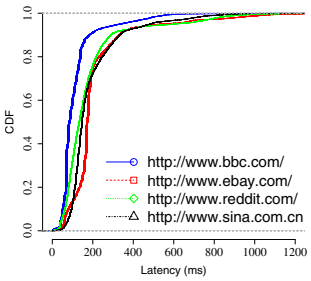

Fig. 7. Latency of downloading the objects of the websites.

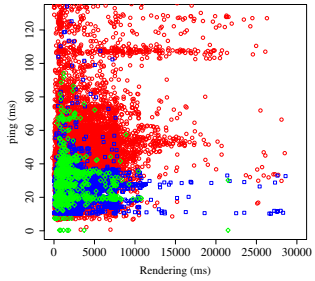

Fig. 8. The correlation between web rendering time test and ping latency test.

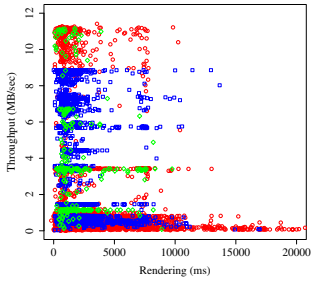

Fig. 9. The correlation between the web rendering time test and speed test throughput.

largest transmission latency and indeed these websites take longer to finish rendering. In fact, the transmission latency of eBay.com might be caused by the number of HTTP objects and the number of HTTP redirects it involve.

We wanted to check the relationship of the web rendering time and a ping test (i.e. for round trip latency for a single packet) for each probe. These two tests are not directly comparable because the ping test targets a single server, and the Webperf test tries at different web servers at different geographical locations. Furthermore, web browsing involves several round trips for different web objects. However, we wanted to see whether the performance trends between different measurement points had any similarity between the two tests.

The web rendering test and ping test are run independently, but we took the hourly averages of the results from both tests for each probe involved in the test, and checked how these results correlate, as shown in Figure 8. In addition, the different measurement sites (two operators plus a few miscellaneous additional measurement points [in green]) are shown with different colours. Some correlation between the latency and rendering time can be seen: the very long web rendering sessions are slightly more frequent when latencies are long. Another observation is that the measurement points at one of the operators have more consistent results on both tests, whereas the other operator has more variability in the ping times and web rendering times. The likely reason for this is that the other operator has longer network distance to the ping server and some of the web servers.

Similarly, we measured the correlation between a throughput test and the web rendering test using a similar approach with the ping test. It can be seen in Figure 9 that nearly always when the web rendering takes long time, also the measured network throughput is very poor. Poor overall network through seems to be the main factor in bad rendering performance.

VI. Conclusions

We developed a large scale measurement tool *WePR* which measures the web performance and the user perceived time it takes to render a web page in a browser window. Along with describing the system design and the metrics we presented the results from our initial trails which have been deployed on more than 50 test probes across Europe. The results from the initial trail shows the rendering performance differs from the measured communication performance. Also the rendering process consumes substantially huge computing resources to do the web rendering test in a large scale.

This work can be extended in different aspects. Currently, we have extended our WebPerf software for mobile networks. We are measuring the web performance in Mobile Broadband Networks from both stationary and mobile nodes using the MONROE platform (www.monroe-project.eu). Furthermore, we will do a subjective test in a controlled environment to find the correlation between the performance we obtained using our tool and the mean opinion score (MOS) rated by the end users.

References

[1] Selenium hq. http://seleniumhq.org.
[2] W3C web performance working group. http://www.w3.org/2010/webperf/.
[3] Web page test. http://www.webpagetest.org.
[4] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM SIGCOMM Computer Communication Review*, 32(1), January 2002.
[5] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM Computer Communication Review*, 40(3):26–33, June 2010.
[6] S. Egger, T. Hofeld, R. Schatz, and M. Fiedler. Waiting times in quality of experience for web based services, 7 2012.
[7] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing web latency: The virtue of gentle aggression. In *Proc. of ACM SIGCOMM '13*, pages 159–170, Hong Kong, China, August 2013. ACM.
[8] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. Hostview: Annotating end-host performance measurements with user feedback. *SIGMETRICS Perform. Eval. Rev.*, 38(3):43–48, Jan. 2011.
[9] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating the edge network. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 246–259, Melbourne, Australia, November 2010.
[10] P. C. Paul and G. F. Quraishy. Impact of HTTP object load time on web browsing qoe. Master's thesis, Blekinge Institute of Technology, Sweden, 2013.
[11] S. Sundaresan, N. Feamster, R. Teixeira, and N. Magharei. Measuring and mitigating web performance bottlenecks in broadband access networks. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, pages 213–226, New York, NY, USA, 2013. ACM.