Troubleshooting web sessions with CUSUM

Christian Callegari[1], Marco Milanesio[2], and Pietro Michiardi[2]
[1]Dept. of Information Engineering, University of Pisa, Pisa, Italy
[2]Eurecom, Sophia-Antipolis, France
E-mail: c.callegari@iet.unipi.it, Marco.Milanesio@eurecom.fr, Pietro.Michiardi@eurecom.fr

***Abstract*—A variety of factors may lead users to a poor quality of experience in a web browsing session, experiencing a high page load time. Without a clear explanation this can be annoying. In this paper, we present a novel algorithm and a whole redesigned architecture to provide an answer to the question "what's wrong with this web site?".**

In more detail, we propose the design and the implementation of a probe, running a novel diagnosis algorithm based on the original use of "classical" troubleshooting techniques merged together with statistical change point detection tools. Our proposed probe is able to correctly determine the root cause of poor web navigation experience, distinguishing, among the several portions of the network, the one responsible for the problem. The presented experimental results demonstrate the effectiveness of the proposed method.

***Keywords*—Quality of Experience; Web session troubleshooting; Change point detection; CUSUM; Anomaly Detection.**

I. Introduction

The 2014 Akamai State of the Internet report [1] shows more than 788 million unique IP4 addresses, exchanging requests and responses with an average connection speed greater than 4Mbps with peaks of 73Mbps. Web browsing occupies the vast majority of the described traffic load, with averages in page load time ranging from 1 to 8 seconds worldwide on broadband. With the increase of the average connection speed, it may become difficult to understand for an end user a slow page load time during a browsing session. Hence, it becomes relevant to develop automatic ways to provide an explanation for it, as many distinct, unrelated, and independent factors can cause a poor quality of experience (QoE) in a web browsing session: an overloaded client, a misconfigured router, a server outage, or a failure somewhere in the path between the user and the server providing the web page.

Diagnosing the quality of experience in web browsing for end users can be achieved following two different approaches: (i) exploiting network performance degradation measurements collected by different tools deployed in different parts of the network, or (ii) involving explicit users' feedback in a particular time window. Note that network performance degradation measures deal more with quality of service (QoS) metrics (e.g., RTT or packet loss) rather than QoE ones (e.g., page load time): in our approach we combine the two, providing a unified view of the browsing experience.

Most of the current approaches in web browsing troubleshooting (e.g., [2]) are based only on browser level measurements, while we believe that also active measurements (e.g., ICMP messages) taken at the time of browsing are relevant to perform a root cause analysis of a degraded user experience. To this aim, we present here a new probe design that, after performing both active and passive measurements, provides insights of possible root causes for a high page load time.

978-1-4799-5344-8/15/$31.00

In this paper we address the following question: "How to diagnose a poor QoE in a web browsing session?". The research question is split into different sub-questions, namely (a) how to define a poor QoE; (b) what measurements to take; (c) how to tell apart the different causes.

Starting from the work in [3], we present here a new design of the Firelog probe, as well as an improved version of the diagnosis algorithm, covering a wider number of cases. More in details: (a) we design and develop a new, lightweight probe suitable for being executed on embedded devices; (b) we enrich the collected data exploiting a network sniffer and active measurements; (c) we improve the diagnosis algorithm introducing the CUSUM methodology for distinguishing the different cases; and (d) we run experiments on a controlled testbed to validate the diagnosis scheme.

It is worth highlighting that the rationale behind the use of CUSUM is to detect the root cause of an anomalously slow browsing session, while automatically adapting to changing network conditions. Indeed, with respect to the use of a standard threshold mechanisms, the main characteristic of CUSUM (i.e., a change point detection technique) is the ability to adapt to slow changes in the network performance, still being able to detect abrupt changes. This feature makes this method strongly suitable for QoE monitoring, where user dissatisfaction is more related to a sudden worsening of the browsing performance (e.g., a web page that is slow with respect to the previously browsed URLs) than to the absolute browsing time (indeed, in that case we could suppose that the QoE level is mainly dependent on the access network performance).

The paper is organized as follows. In Section II we present the related works as well as a summary on the CUSUM methodology. Then we present the new probe design in Section III, giving a description of the overall architecture. We present the diagnosis algorithm in Section IV, while in Section V we describe the test-bed used to validate our proposal and discuss the preliminary experimental results, obtained in such a controlled environment. Section VI concludes the paper with some future directions.

II. Related Work

Page load time is widely considered the main QoE metric to be investigated. In addition, network performance metrics can be used to drill down on the perceived QoE: in [4] authors point out the QoS metrics (e.g., packet loss) can influence

QoE, even if they are strongly related to two distinct views: the network centric (QoS) and the user centric (QoE).

There exist many tools for debugging or monitoring web sessions (e.g., Firebug [5] or Fiddler [2]) that actually lack a systematic troubleshooting model, as well as well-defined troubleshooting techniques. Furthermore, they use only browser level metrics (e.g. HTTP headers, web page size and so on). Other works are aimed at correlating bad browsing performance with web page properties (e.g., number of objects, use of CDNs [6]), or to include user participation in performance evaluation, simply indicating "satisfaction" thresholds [7][8]. User dissatisfaction prediction is the goal of the work in [9]: exploiting explicit users' feedback, authors develop a classifier for supervised learning, based on network metrics (e.g., RTT, jitter, retransmissions).

Fathom [10] introduces a Firefox browser plugin for network troubleshooting. Fathom broadly measures a wide spectrum of metrics that characterize a particular Internet access such as access bandwidth, bottleneck buffer size, and DNS behavior. It can also enable additional active measurement tasks. Similarly, we exploit browser events and active measurements, but we couple them with the corresponding TCP flows captured by the network interface, and we provide a fast diagnosis on the current web browsing session.

As stated in the introduction, the basis of our work is the one described in [3], where the authors propose a browser plugin able to diagnose the cause of slow web browsing performance. Our proposal significantly advances this design both from the probe architecture and the diagnosis points of view, including a more comprehensive algorithm. Moreover, to the best of our knowledge, our proposal is the first to introduce statistical anomaly detection techniques, able to dynamically adapt to network conditions, in a troubleshooting tool.

Given that a review of the state of the art of anomaly detection techniques is out of the scope of this work, we refer the reader to the survey [11] and references therein for the most widely used techniques used to detect anomalous behaviors in network traffic.

A. Theoretical Background: CUSUM

The CUSUM (or cumulative sum control chart) is a sequential analysis technique, typically used for solving the change detection problem. Let us suppose to have a time series, given by the samples x_n from a process: the goal of the algorithm is to detect with the smallest possible delay a change in the distribution of the data. The assumption of the method is that the distribution before and after the change ($f_{\theta_1}(x)$ and $f_{\theta_2}(x)$) are known. As its name implies, CUSUM involves the calculation of a cumulative sum, as follows:

$$\begin{aligned} S_0 &= x_0 \\ S_{n+1} &= \left(S_n + log\left(\tfrac{f_{\theta_2}(x)}{f_{\theta_1}(x)}\right)\right) \end{aligned} \tag{1}$$

The rationale behind the CUSUM algorithm is that, before the change, the quantity $log\left(\frac{f_{\theta_2}(x)}{f_{\theta_1}(x)}\right)$ is negative, whereas after the change it is positive: as a consequence, the test statistics S_n decreases before the change, and it increases linearly with a positive slope after the change, until it reaches the threshold ξ when the alarm is raised. Figure 1 shows an intuitive derivation of the method.

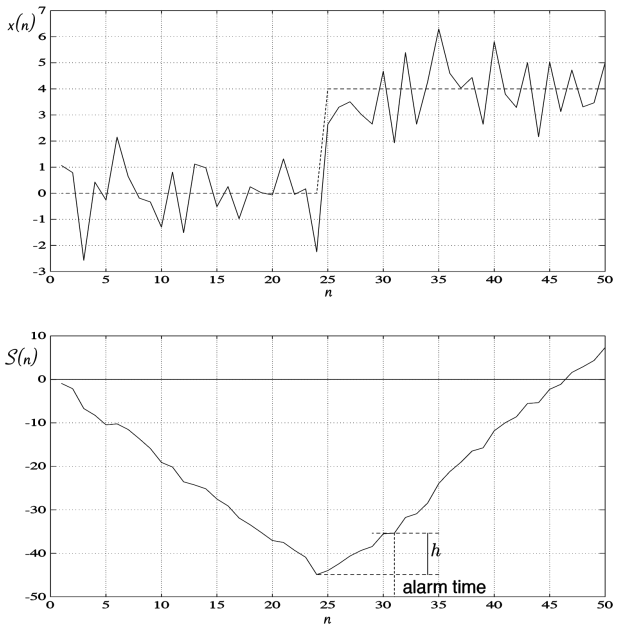

Fig. 1. Intuitive derivation of the CUSUM: time-series (upper graph) and CUSUM statistics (lower graph)

Note that the assumption about the knowledge of the two distributions $f_{\theta_1}(x)$ and $f_{\theta_2}(x)$, implies that CUSUM is only able to decide between two simple hypotheses. But, in case of network problems we cannot suppose that the distribution after the change is known (usually neither the distribution before the change is known). This implies the need of using the non parametric version of the algorithm [12], which leads to a different definition of the cumulative sum S_n. In more detail in this work we have used the non parametric CUSUM (NP-CUSUM), in which the quantity S_n is defined as:

$$\begin{aligned} S_0 &= x_0 \\ S_{n+1} &= (S_n + x_n - (\mu_n + c \cdot \sigma_n))^+ \end{aligned} \tag{2}$$

where μ_n and σ_n are the mean value and the standard deviation until step n, c is a tunable parameter of the algorithm, and the operator $(x)^+ = \max(0, x)$ is introduced for making the implementation simpler, as S_{n+1} may become negative, as in the example pictured in Figure 1.

As far as the estimations of μ and σ are concerned, we can use the Exponential Weighted Moving Average (EWMA) algorithm defined as:

$$\begin{aligned} \mu_n &= \alpha \cdot \mu_{n-1} + (1-\alpha) \cdot x_n \\ \sigma_n &= \alpha \cdot \sigma_{n-1} + (1-\alpha) \cdot (x_n - \mu_n)^2 \end{aligned} \tag{3}$$

where α is a tunable parameter of the algorithm.

III. Probe Description

Firelog[1] is a hybrid probe capable of performing both active and passive measurements over web browsing sessions. At

[1] http://firelog.eurecom.fr/mplane/ Last Visited: Feb. 2015

its origin, Firelog was a full browser-based probe [3], namely a Firefox plugin collecting browser metrics. We changed the architecture of the probe, by developing part of its logic into a standalone application: we enhanced it by using well known Ping and Traceroute tools to perform active measurements, and by having two sets of passive measurements: timings of browser events, and captured network traffic dumps. We lightened the overhead due to a full browser by exploiting an instrumented headless browser (i.e., phantomJS[2]), and we use a modified version of Tstat[3] for capturing the network traffic.

The collected metrics (see Section III-A) are stored in a local database and in the form of a HTTP archive format file[4], helpful to visualize the overall browsing process (e.g., objects, size and time elapsed to fetch it) for a specific web page. These results are processed to produce a first evaluation on the last browsing session and sent to a central repository for further analysis and diagnosis for troubleshooting.

Note that, as the probe is targeted to diagnose poor QoE in a web browsing session, users are given the ability to explicitly signal a poor QoE.

A. Collected Metrics

Firelog collects both active and passive measurements, as follows.

Given a URL, the probe browses the URL. That is, it performs a DNS query to resolve the name and downloads the web page as usual, by contacting also all the possible secondary servers[5] needed to retrieve all the objects. We collect at this stage a number of metrics regarding browser events (e.g., IP addresses, page load time, request time, time for DNS resolving, number of objects loaded, time between the HTTP GET message and first byte of data received, and many others). We call this a *session*, and we collect also information on the status of the probe itself (namely, CPU and memory usage).

For each object in the session, a unique identifier is generated and attached to the TCP level streams incoming and outgoing from the probe via the Tstat network sniffer. By doing so, we couple the TCP stream measurements (e.g., the TCP handshake time) to the corresponding object. For each collected IP address, we send ICMP messages (Ping and Traceroute) to compute the path and the RTTs towards the destinations. Ultimately, each URL is associated to a complete set of metrics coming from the browser, the ICMP messages and the passive sniffing, giving us a snapshot of what happened in the session. All the raw data are then stored, processed and sent directly to the central repository.

The relevant metrics which are used by the diagnosis algorithm presented in Section IV are summarized in Table I. In brief, active measurements are ICMP messages and the passive ones are collected as browser events or through the network sniffer (i.e., T_{tcp}). T_{idle} is the sum of "gaps" in which no browser activity is performed (i.e., small fractions of time elapsed between the last byte received of an object and the sending of the GET message for the next object).

[2]http://phantomjs.org/ Last Visited: Feb. 2015

[3]http://tstat.polito.it/ Last Visited: Feb. 2015

[4]https://code.google.com/p/harviewer, Last visited: Feb. 2015.

[5]For example, consider a news web site: several objects may be retrieved either by other servers in the same domain or from different servers on different domains (e.g., video, advertisements and so on)

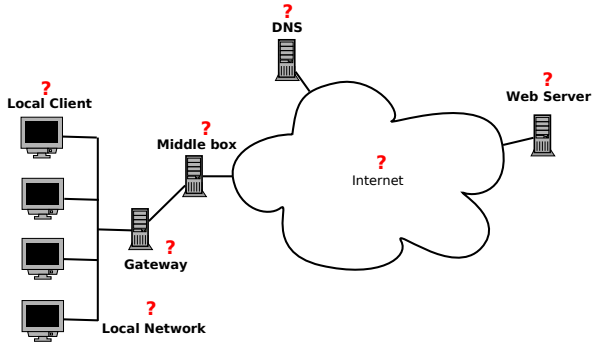

Fig. 2. Network scenario: question marks point to the possible source of a poor QoE.

Symbol	Metric	Source
T_{nhop}	RTT to the n^{th} hop	Active
T_p	RTT to probe p on the same LAN	Active
Δ^n	$T_{(n+1)hop} - T_{nhop}$	Computation
T_{idle}	Client idle time	Computation
T_{tot}	Total web page downloading time	Passive
T_{DNS}	DNS response time	Passive
T_{tcp}	TCP handshake time	Passive
T_{http}	HTTP response time	Passive

TABLE I. METRICS COLLECTED BY THE PROBE AND EXPLOITED BY THE DIAGNOSIS ALGORITHM.

More measurements are available at the probe side, and more tools can be easily integrated for enriching the Firelog measurements (e.g., Tracebox [13], to detect the presence of middle boxes in the network). For sake of generality and clarity in the explanation, however, we do not include all of them here.

IV. DIAGNOSIS ALGORITHM

The proposed algorithm aims at identifying the portion of the connection that is responsible for the anomalously long web page loading time. Hence, we can mainly identify seven different segments, where the problem could be located (indicated with a question mark in Fig. 2): (1) the local client; (2) the local network; (3) the gateway; (4) middle boxes (if any); (5) the DNS sever; (6) the backbone network; and (7) the remote web server.

The algorithm is run whether: 1) the user requests a diagnosis on a URL; or 2) a threshold on the page load time is crossed. See Section V for more details.

Let us analyze in details how the algorithm works.

The first check is made on the local device, by checking the status (e.g., memory and CPU consumption) of the probe itself. If the test passes (i.e., the considered metric exceeds a threshold), then the algorithm concludes that the problem is at the probe side. Otherwise, if the local host does not present

any problem, the algorithm performs a check on T_{http} (average value over all the values corresponding to the different objects of the loaded web page), by verifying if the CUSUM applied to that metric has exceeded a given threshold. Note that this metric can be considered as a rough approximation of the time required for getting the first data packet from the remote web server, thus in case it is normal we can easily conclude that the problem is neither in the network (local or backbone) nor at the remote server side. Hence, the algorithm performs a check, at first, on the web page size (verifying if the number of objects/bytes of the page exceeds a threshold) and then, in case the web page size is not responsible for the problem, it checks T_{tcp} and T_{DNS} possibly concluding that the problem is generated by the long distance towards the remote web server or in the DNS server, respectively.

Instead, in case T_{http} is normal, the algorithm automatically excludes the DNS and the page size cases and proceeds by checking if other devices in the same local network have problems.

At this point, there can be three distinct cases: (1) all the other devices are experiencing some problems; (2) none of the other devices is experiencing any problem, and (3) just some of the other devices are experiencing some problems. Let us analyze how the algorithm behaves in the three distinct cases.

First case. In the first case the algorithm can directly exclude that the problem is due to the remote server (assuming that not all the devices are browsing the same page). Moreover, among the remaining causes (in order: gateway, local network, middle boxes, and backbone network), the algorithm assumes that with high probability the problem is located close to the devices (otherwise not all the devices would experience problems). The algorithm checks thus the remaining causes in the mentioned order. If the tests on gateway, local network and middle boxes fail, it concludes that the problem is in the closest portion of the backbone network, given that all the local devices are traversing it (e.g., same ISP).

Let us see, into details, how the different phases of this part of the algorithm are performed, by beginning with the gateway and local network verification. First of all, the algorithm verifies if the CUSUM applied to the RTT to the first hop (i.e., the gateway) exceeds a given threshold. If this is the case, this can be justified by either the fact that the local network is congested or by the fact that the gateway is overloaded and the Ping response time is high (i.e., the CUSUM statistics exceeds the threshold). To discriminate between this two cases the algorithm checks the status of other devices in the network (if any) by applying the CUSUM to the RTT from the initial probe and another device in the local network and if it is high too, it concludes that the problem resides in the local network, which is probably congested, otherwise it concludes that the problem is in the gateway, which is probably overloaded. Else, if T_{1hop} is "normal" (i.e., the CUSUM statistics does not exceed the threshold), the algorithm cannot yet exclude the overloaded gateway case (because the dependence between the ping response time and the machine load is not always significant), and performs a check on the CUSUM applied to Δ^1 (i.e., $T_{2hop} - T_{1hop}$).

This metric, from a practical point of view roughly represents the sum of the time needed to traverse the gateway, the time needed to go through the first link outside the gateway, and the time required by the second hop to process the ping request. Thus, if it results anomalous, the algorithm also checks the CUSUM applied to Δ^2 (i.e., $(T_{3hop} - T_{2hop})$ and, in case it is anomalous too, it concludes that there is congestion on the first link outside the gateway, which is reported as backbone network problem (note that if there are middle boxes the algorithm instead proceeds to the next phase), otherwise it concludes that the problem is in the gateway that is overloaded (given that the problem is associated to the time required to traverse the gateway).

It is worth noticing that the quantities T_{nhop} are not required to be collected towards the nodes that are in the path to the contacted web server, but they can be measured on every path outside the local network: in our case we choose these nodes by performing a Traceroute towards the resolved IP address in a session.

In case Δ^1 results are normal, the algorithm can exclude the overloaded gateway case, and proceeds by checking each middle box detected, if any (e.g., exploiting tools like Tracebox [13]).

The verification of the middle box is based on a process that is very similar to the one used to check the gateway, indeed the algorithm checks the CUSUM applied to T_{nhop} (where n is the middle box). If this is anomalous, it can conclude that the problem is in the middle box, otherwise it checks if any anomaly is present in Δ^n: if not, it excludes the middle box and goes to the next middle box (if present), otherwise it also check Δ^{n+1} and concludes that the problem is in the middle box if the latter is normal, or in the congested network if Δ^{n+1} is anomalous.

This phase can exploit all the information already obtained from the previous phases: if the algorithm cannot locate the problem neither in the gateway and local network, nor in the middle boxes, it concludes that the problem resides in the portion of the backbone network closest to the probe.

Second case. Let us analyze now the case in which none of the other devices of the local network is experiencing any problem. In this case, we can easily exclude the gateway, the local network, and the middle boxes, restricting the causes to either the remote server or the backbone network. Hence, the first check is performed on the remote server (that is assumed to be more probable than the backbone network, given that the only device that is experiencing problems is the one navigating that remote server). To perform such a check, the algorithm verifies if the CUSUM applied to the metric $T_{http} - T_{tcp}$ is anomalous or not: this metric roughly represents the time needed by the remote server to process the http GET request, being T_{tcp} almost independent on the server load. In case it is anomalous the algorithm concludes that the problem is located in the remote server, otherwise that it is located in the backbone network. In fact, given that an anomalous return value on the CUSUM could also be due to the loss and consequent retransmission of the GET packet, the algorithm, before returning that the problem is located in the remote server, "asks" the client to reload the web page and performs that check once again, minimizing the probability of producing a wrong output. It is also important to highlight that, in case we do not want to require the page reload, the

only confusion can be between the remote server and the far portion of the backbone network, that can be still acceptable in most cases.

Third case. The last case still to be analyzed is the one in which some of the local network devices are experiencing some problems and some are not. This case results to be straightforward, given that we can directly exclude all the cases apart from the backbone network problem, hence the algorithm directly concludes that the problem is in the backbone network (probably in a portion of the network close to the local network, given that it is traversed by several local devices).

The number and the type of the operations made by the probe, make it suitable for being used as a background process, without significantly affecting the system performance. Indeed all the checks are just performed by either comparing some passive measurements to a threshold or computing the CUSUM statistics (CUSUM is well-known for being suitable for all kind of real-time applications) and comparing them with a threshold.

Finally, it is also important to highlight that in the case there is not any other device in the local network apart from the one that raised the alarm, the algorithm can still be applied, by checking all the possible locations of the problem (following the order: local host, page size, server too far, DNS, gateway, local network, middle boxes, and remote server).

V. Experimental Results

In this section we describe the experiments carried out to validate the proposed algorithm, by analyzing at first the tuning of the system parameters and then describing the network test-bed considered in the validation process and the obtained results.

A. Tuning of the System Parameters

The proposed algorithm presents several parameters that have to be tuned before launching the probe. Nonetheless, this phase is not so critical: we design the algorithm so that small changes in the different parameters values result in the same diagnosis result. In more detail, we have to determine the following quantities:

- EWMA parameter α: in our settings we have used the value $\alpha = 0.9$, which is "classical" in many network applications (e.g., [14])
- CUSUM parameter c: we have set $c = 0.5$, as in other previous works on CUSUM (e.g., [15])
- Algorithm thresholds: the choice of these thresholds, that usually represents a critical aspect in the application of CUSUM based methods[6] in other fields (e.g., network anomaly detection), has resulted not to be that critical in this application scenario.

[6]Thresholds regarding page sizes are of course domain-dependent, and vary from page to page when browsing real web sites.

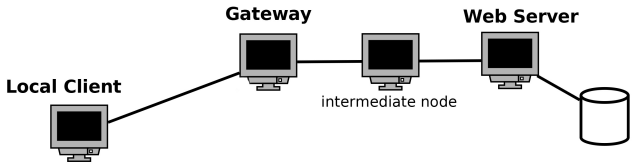

Fig. 3. Schematics of the used test bed.

B. Experimental Results

To validate and verify the behavior of the diagnosis algorithm and the performances of the proposed probe, we have taken into considerations two distinct experimental scenarios: a controlled laboratory testbed to validate the proposed diagnosis algorithm, and a set of browsing sessions into the "wild" Internet, to verify the suitability of the developed probe for real-world applications.

At first, an exhaustive set of experiments has been conducted in a testbed composed of four distinct PCs, configured as depicted in the figure 3, so as to verify the effectiveness of our proposal.

Given the setup of the testbed we have been able to emulate three distinct cases:

- "normal" functioning
- congestion on the local network
- congestion the backbone network

The three cases have been realized by using *netem* [16], which providing us with the ability of automatically adding variable losses and delays on the network, has allowed the realization of a labeled dataset (ground truth).

Note that in this testbed we have not involved any human interaction, meaning that the diagnosis algorithms has been used over all the sessions and not only when a "dissatisfaction signal" was generated. It is important to highlight that this fact could bias the results, in terms of a bigger number of false positives (that could be not relevant in the "real-world" scenario, where the user not necessarily raise an alarm), but not in terms of false negatives. Indeed, the problem normally connected to the choice of the algorithm thresholds is that it has a direct impact on the number of detected anomalies, but also on the number of false positives (events signaled as anomalous that are, in fact, normal events). Nonetheless, in our application scenarios, we can accept a certain number of false positives, without affecting the system performance. This is due to the fact that having a false positive, without the signaling of the problem, does not lead to any conclusion. Hence, from a practical point of view, we have tuned these thresholds to a value that is equal to the mean value of the CUSUM obtained during a normal session plus a corrective factor computed as a function of the CUSUM variance (i.e., scaling). For this reason, in this tests we have also performed a preliminary training phase aimed at computing the threshold values.

Table II shows the obtained results. In more detail, over a total of about 1800 distinct browsing sessions, the algorithm has not produced any false negative, and it has introduced 11

Considered Case	Algorithm Output		
-	"Normal" Functioning	Local Network Congestion	Backbone Network Congestion
"Normal" Functioning	1617	9	2
Local Network Congestion	0	112	0
Backbone Network Congestion	0	0	159

TABLE II. EXPERIMENTAL RESULTS

false positives. Moreover, in case of really anomalous sessions (i.e., very high latencies and packet loss ingested) the algorithm has always correctly identified the cause.

Finally, to conduct a preliminary performance evaluation of the probe, verifying its suitability for real world use, we have conducted experiments into the "wild" Internet. This last scenario is not used to validate the diagnosis algorithm, as we do not have any control on the full path between the probe and the web server, but to verify if the developed system is able to deal with a real operative network scenario. The overall process of browsing a URL and running the diagnosis algorithm for a single session spans from 1 to 3.5 minutes, that we think it is a reasonable time for providing the end user with a diagnosis for a poor QoE. This time span is due to the browsing timing itself, which differentiates between small web sites (e.g., Google front page) and complex web sites (e.g., news web sites with a high number of servers to contact to fetch different objects). Most of the time is spent performing the active measurements: we have to wait for Ping messages and Traceroutes to return their results. As previously mentioned, all the results are stored locally and sent to a central repository for further analysis. We store all the collected data and the diagnosis result in JSON files, growing from less than 20 kB (small sites) to a maximum of 800 kB (very big sites).

VI. CONCLUSION AND FUTURE WORK

We presented in this paper a new probe architecture that makes use of a novel algorithm to perform root cause analysis over poor QoE in web browsing sessions. We described the algorithm in details, underlining the rationale behind it, and presented experimental results from a controlled testbed, for validating the approach.

The obtained results give us further questions to be answered and investigated. Future works include to perform additional experimental tests in the "wild" Internet, a more detailed analysis of the loss rate impact at different steps in the path, and the exploitation of multiple vantage points geographically distributed, to drill down in the "generic network" result case (e.g., investigate at the autonomous system level). Furthermore, we are currently applying the ITU QoE model G.1030 [17] to increase the accuracy in our diagnosis algorithm and we are defining a methodology to overcome the domain knowledge requisites in setting the parameters of the algorithm, and to compute and adjust the thresholds in a dynamic way.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union under the FP7 Grant Agreement n. 318627 (Integrated Project "mPlane") and under the FP7 Grant Agreement n. 607019 (Collaborative Project "SCOUT").

REFERENCES

[1] A. Corporation, "The state of the internet, q2, 2014." http://www.akamai.com/dl/whitepapers/akamai-soti-a4-q214.pdf. Accessed: 2014-12-3.

[2] "Fiddler." http://www.telerik.com/fiddler. Accessed: 2015-02-18.

[3] H. Cui and E. W. Biersack, "Troubleshooting slow webpage downloads," in *TMA 2013, 5th IEEE International Traffic Monitoring and Analysis Workshop, in conjunction with INFOCOM 2013, 14-19 April 2013, Turin, Italy*, (Turin, ITALY), 04 2013.

[4] O. Hohlfeld, E. Pujol, F. Ciucu, A. Feldmann, and P. Barford, "A qoe perspective on sizing network buffers," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, (New York, NY, USA), pp. 333–346, ACM, 2014.

[5] "Firebug." http://getfirebug.com/. Accessed: 2015-02-18.

[6] M. Butkiewicz, H. V. Madhyastha, and V. Sekar, "Understanding website complexity: Measurements, metrics, and implications," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, (New York, NY, USA), pp. 313–328, ACM, 2011.

[7] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, (New York, NY, USA), pp. 295–312, ACM, 2011.

[8] H. Cui and E. W. Biersack, "On the relationship between QoS and QoE for web sessions," Tech. Rep. EURECOM+3608, Eurecom, 01 2012.

[9] D. Joumblatt, J. Chandrashekar, B. Kveton, N. Taft, and R. Teixeira, "Predicting user dissatisfaction with internet application performance at end-hosts," in *INFOCOM, 2013 Proceedings IEEE*, pp. 235–239, April 2013.

[10] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson, "Fathom: a browser-based network measurement platform.," in *Internet Measurement Conference* (J. W. Byers, J. Kurose, R. Mahajan, and A. C. Snoeren, eds.), pp. 73–86, ACM, 2012.

[11] C. Callegari, A. Coluccia, A. D'Alconzo, W. Ellens, S. Giordano, M. Mandjes, M. Pagano, T. Pepe, F. Ricciato, and P. Zuraniewski, "A methodological overview on anomaly detection," in *Data Traffic Monitoring and Analysis*, pp. 148–183, Springer Berlin Heidelberg, 2013.

[12] A. G. Tartakovsky, B. L. Rozovskii, R. B. Blaek, and H. Kim, "Detection of intrusions in information systems by sequential change-point methods," *Statistical Methodology*, vol. 3, no. 3, pp. 252 – 293, 2006.

[13] G. Detal, B. Hesmans, O. Bonaventure, Y. Vanaubel, and B. Donnet, "Revealing middlebox interference with tracebox," in *Proceedings of the 2013 ACM SIGCOMM conference on Internet measurement conference*, ACM, October 2013.

[14] O. Salem, S. Vaton, and A. Gravey, "A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice," *Int. J. Netw. Manag.*, vol. 20, pp. 271–293, September 2010.

[15] C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Detecting anomalies in backbone network traffic: a performance comparison among several change detection methods.," *IJSNet*, vol. 11, no. 4, pp. 205–214, 2012.

[16] S. Hemminger, "Network emulation with NetEm," in *LCA 2005, Australia's 6th national Linux conference (linux.conf.au)* (M. Pool, ed.), (Sydney NSW, Australia), Linux Australia, Linux Australia, Apr. 2005.

[17] ITU-T, "Estimating end-to-end performance in ip networks for data application. g.1030."