DiagSys: network and third-party web-service monitoring from the browser's perspective (industry track)

Loïck Bonniot, Christoph Neumann, François Taïani

DiagSys: network and third-party web-service monitoring from the browser's perspective

(industry track)

Loïck Bonniot[1,2], Christoph Neumann[1], François Taïani[2]
[1]InterDigital, [2]Univ Rennes, Inria, CNRS, IRISA
{firstname.surname}@interdigital.com,{firstname.surname}@irisa.fr

Abstract

For Internet operators, on-line service providers and end-users, representative operational measurements are crucial to monitor and diagnose the performance of networks and on-line services. While numerous approaches have been proposed to measure performance, only a few works fully adopt an end-user perspective by taking measurements from within web browsers.

In this paper we propose and describe DiagSys, a novel crowd-sourced data collection system designed to monitor the performance of network- and web-services from a range of diverse viewpoints. DiagSys leverages the web browsers running on end-user devices to probe dedicated remote measurement points and third-party web services. It uses a JavaScript snippet embedded within webpages and/or a dedicated browser extension to this end, while staying compatible with recent browser capabilities and security restrictions. We also present interesting case studies based on the data already collected in our DiagSys deployment.

1 Introduction

Internet Service Providers, on-line service providers and their end-users need accurate and automated tools to measure and diagnose networks and third-party on-line services on a large scale. To provide insightful reports, such tools should ideally reflect the Quality of Experience (QoE) perceived by end-users when they use on-line services such as websites and web APIs. Because QoE problems are often explained by causes near end users [16, 18, 19], many past measurement approaches have been implemented at the network's edge, by taking the viewpoint of either the home gateway [15, 17, 20], the browser [5, 6, 9, 11], or by using dedicated tools running on end-user devices [8]. (See Table 1 for an overview of some of these approaches.) In this paper, we propose to take stock of these seminal approaches, yet to get one step closer to a *holistic* monitoring of QoE conditions: we combine end-user perspective with infrastructure-side insights in a more systematic monitoring strategy, which is often lacking in the above solutions.

More concretely, we argue that although the location of measuring probes in the network is critical, the device used (PC, smartphone …) and the execution environment are also essential to capture a user's QoE. We therefore advocate that measurements should whenever possible be taken from end-user devices. This implies that any user-side measurement software should be easy to deploy and use, remain non-intrusive and incur a minimal network overhead. Browser-based measurements [5, 11]—the approach we explore in this paper—adhere to the above principles. Most web services used by end-users run within a browser, where measurement scripts can be easily deployed using JavaScript with little to no user interaction. Moreover, web browsers make it easy to target a wide range of devices, from PCs to smartphones through gaming consoles.

Table 1. Comparison of some client-side monitoring tools.

	Headless browser	End-user device	3rd-party service monitoring	Traceroute support
Fathom [5]	✗	≈[2]	✗	✓
NDT [11]	✗	✓	✗	✓
Mirage [17]	≈[1]	✗	✓	✗
DiagSys	✓	✓	✓	✓

[1] No JavaScript emulation [2] No support for recent browsers

Although end-user measurements appear key to reliably assess users' QoE, measurements taken from end-user devices do come with caveats: first, browser-based measurements are more challenging to implement nowadays than in the past [5], due to the many security restrictions added in recent years. Then, end-user devices are not always on, they often change network location (i.e. in the case of smartphones, laptops, and tablets), might not go back to a particular site or service for extended periods of time. We overcome these limits by supplementing user-based measurements with measurements taken from headless browsers running within the infrastructure. We also focus our monitoring effort on a preconfigured set of neutral third-party web services which can be changed at runtime. This allows us to build a consistent and fine-grained set of datapoints obtained from diverse vantage points.

Measurements taken from within a browser do bring a lot of information, but might remain difficult to dissect and diagnose without additional insights into the inner workings of the concerned services, or into the topology of the underlying networks. This information is however usually not

accessible, for security or business reasons [21]. We therefore take a tangential approach to estimate network conditions, and rely on a set of *landmark servers* (landmarks for short) that implement specialized measurement services at diverse locations of the infrastructure (e.g. a cloud datacenter, a PoP, a home network). In addition to measurement services, landmarks also host our headless browsers, and allow us to implement network probing mechanisms that are unavailable from browsers, such as traceroute.

In the following, we present DiagSys, a crowd-sourced data collection system targeted at monitoring networks and third party web-services that implement the strategies we have just sketched. DiagSys combines browser-based probes, running both on end-user devices and in headless browsers, and landmark servers hosting measurement services. Our browser-based probes are compatible with the recent security restrictions of modern browsers, and systematically monitor a set of pre-configured services. We describe a first set of case studies based on the data collected so far with DiagSys. Despite a quite recent and therefore limited deployment we can already show that DiagSys can provide insightful data regarding third-party web service behavior and load, CDN behaviors and routing or network load issues.

2 DiagSys system description

An overview of DiagSys is depicted in Figure 1. A set of "landmarks" (reference servers) are deployed in diverse vantage points within the Internet (subsection 2.1). We carefully designed landmarks in order to also measure the effects of CDNs during probing. The landmarks act as measurement points for users, that can directly probe them from unmodified web browsers (subsection 2.2). DiagSys also offers a browser extension as an additional mechanism to probe third-party Internet services from an end-user browser (subsection 2.3). Browser extensions can bypass cross-domain security policies, thereby providing access to additional measurements regarding third-party web services. In addition to real users installing the extension, we run headless browsers in each landmark to *emulate a user's navigation* on our targeted services. For clarity, we refer to user browsers (whether instrumented with an extension or not) as *clients*. A distributed data-store provides up-to-date references to landmarks and services, along with long-term storage for result samples. Since DiagSys has no strong consistency requirements, we use asynchronous multi-master replication to provide a highly-available, low-latency data-store.

2.1 Landmarks as reference points

We rely on a fleet of landmark servers, acting as reference points and providing *relevant features* (or "metrics") to clients. Ideally, one would want to deploy these landmark servers broadly, covering many autonomous systems, datacenters, interconnection points etc. to cover all the paths from users and their service providers to Internet services and their own cloud resources. With DiagSys, landmark servers are self-contained stateless public HTTP servers that can be provided by different ISPs, cloud providers or other third parties in exchange of measurement analytics. (The approach is similar to the global network of Speedtest servers [13], which is an example of practical public landmark servers deployment.)

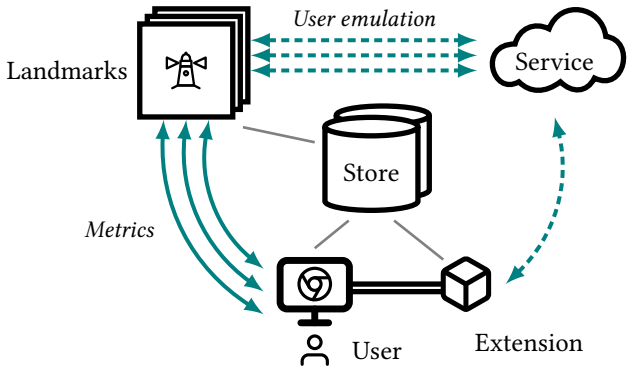

Figure 1. Overview of DiagSys. Landmarks metrics can be directly fetched using user browser, while service health probes can only be executed through a browser extension or client emulation in landmarks (dashed lines). A distributed datastore is used to collect experiment samples.

In our design, a landmark server does not make any assumption about the underlying layers under the HTTP application layer: it is possible to serve clients using legacy HTTP/1 over the TCP transport, to more recent clients requiring HTTP/3 over UDP transport. To avoid data tampering and privacy leaks, landmarks must use TLS. In our prototype, landmark servers are implemented in Go, and provide the following endpoints for metric collection:

`/ping` This endpoint first upgrades the HTTP connection to WebSocket and replies immediately with an empty message for each message sent by a client. The client computes an accurate round-trip time (RTT) [10], without the classic overhead of HTTP requests. Multiple messages allow to estimate the connection jitter between the client and the landmark.

`/download` Clients can download uncompressed random binary data with a single GET query. We use a multithreaded pseudo-random number generator to provide the maximum possible throughput server-side and actually measure the network limit. A waiting queue is also used to limit to only one download at a time and avoid client concurrency. This design allows us to reliably measure download speeds up to 8 Gb/s with recent commodity hardware. Clients discard the first chunk of data to avoid counting the queuing delay, and can download chunks for a maximum of five seconds. The client is responsible for measuring the download time.

`/upload` Clients can upload random binary data using a single POST query. The main challenge is to limit the overhead imposed by JavaScript clients browser-side. In practice, we only generate 1024 bytes of pseudo-random data from the client and repeat them until we reach the desired sample size. Again, we rely on a waiting queue to avoid client contention. Since it is not possible to send data chunk by chunk using JavaScript APIs, the server is the one responsible for measuring received chunks' lengths and delays. It sends its report to the client after having received all the data or encountering a timeout of five seconds.

`/conn` While a client cannot extract transport layer statistics from the available JavaScript functions, the server can provide its own transport statistics to the client. If the HTTP connection is supported by a TCP socket, we use the `getsockopt` Linux syscall on the server to obtain raw TCP statistics, containing among others the number of retransmissions and the minimum round-trip time measured by TCP. When available, we also return the congestion control algorithm used by the server, along with the set of statistics for supported algorithms. Thanks to the HTTP/1.1 Keepalive feature, TCP connections are not reset between HTTP calls. One client can thereby retrieve the full TCP statistics after having performed the download and upload tests for in-depth insights on its connectivity towards the landmark. We plan to also support QUIC statistics for HTTP/3 support.

`/traceroute` We propose two endpoints to 1) start a traceroute from the landmark server to the client public IP and 2) retrieve the result of this traceroute a few seconds later. (This allows a client to start a traceroute asynchronously without blocking while waiting for the response.) To detect NATs and ECMP routes, we use dublin-traceroute, a variant of the recognized paris-traceroute [1]; and we complete found intermediate hops with their DNS PTR record ("reverse DNS").

Landmark servers also periodically probe other landmarks and third-party web services, by emulating real users' navigation in a headless Chrome browser, and saving response times of services (home page and all the associated resources).

The special case of CDNs. Content Delivery Networks (CDNs) are widely used as the public-facing component of many web service [24]: they cache static resources and relay requests to "origin" servers. The main advantages of such architecture are two-fold. First, a CDN can redirect clients to the closest Points of Presence (PoP) thus lowering latencies. Second, many CDNs propose security features to protect the origin server from abnormal traffic, like Distributed Denial of Service attacks or ill-formed requests.

DiagSys covers common CDN PoPs by leveraging the caching mechanisms of CDNs to serve *degraded* landmarks. The basic idea is to host two files on a controlled origin server: an empty file (for degraded latency measurement) and a random file of known size (we use 8MB, for throughput measurement). A CDN can be configured to cache the files indefinitely: any client accessing one file will obtain it from one PoP. Our assumption is that the chosen PoP only depends on a client's location, and will be the same when downloading a resource from a landmark as when using an actual web service. We deployed this strategy in Cloudflare and get the name of the selected PoP from the `CF-RAY` header.

2.2 Browser-based measurements

DiagSys implements browser-based probing in JavaScript, which can be incorporated into any webpage. We assume that JavaScript is enabled in user browsers to allow custom logic to be executed (this is the default, but some users might want to disable JavaScript). We also note that this analysis is based on Firefox version 76 and Chromium version 83.

Latency measurement. Recent browsers expose a standardized JavaScript API (Resource Timing interface [23]) to extract each HTTP request's delays. This makes it possible to retrieve the connection, wait and download delays with millisecond precision. However, for privacy considerations, the W3C recommendation states that these delays can only be available programmatically if the request resource is on the same origin (subdomain) or a suitable `Timing-Allow-Origin` response header is provided. For raw network round-trip measurement, another option is to rely on WebSockets [7, 10]. A simple HTTP request is usually accompanied by a text header of more than 100 bytes, forcing the server to download and parse it. Compared to this scheme, an empty WebSocket message has only an overhead of 6 bytes [2]. A third option would be to use WebRTC data channels to measure round-trip times without the TCP overhead. The main issue with that last option is that data channels require strong permissions from users, such as microphone or webcam access: this would be questionable for a latency-measurement tool to request such permissions. We rely on multiple empty ping/pong messages through a WebSocket connection to estimate latency to landmarks.

Bandwidth measurement. To estimate available network bandwidth, we only send one HTTP request and measure its throughput. This minimizes the overhead mentioned in the last paragraph. Using the JavaScript AJAX API, we retrieve the result of an HTTP request chunk by chunk. Chunk sizes are unpredictable, but are usually a few kilobytes worth of data. We store each chunk's size and the absolute time at which it was received. Then, we aggregate these chunks in a fixed number of "meta-chunks" (8) and compute the total time taken to download each meta-chunk. This method captures the potential variations of measured throughput and avoid being biased with slow-start and bursts caused by browser or system buffers.

Cross-origin security restrictions. To protect users from cross-site scripting (XSS) vulnerabilities, recent browsers block requests to third-party origins by default. Cross-origin

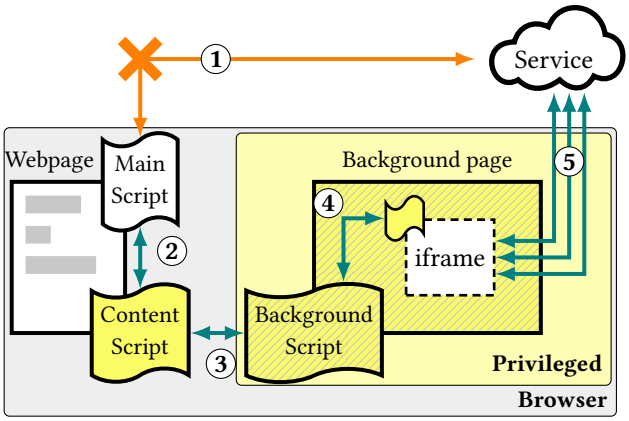

Figure 2. Third-party health check via browser extension: due to the default cross-origin policy ①, a webpage cannot directly fetch a third-party service's resources. However, it is possible to communicate with the background script of an extension *via* injected content scripts (② and ③). The background script can create background iframes ④ that are allowed to load the resources of any service ⑤.

requests are still possible by using the cross-origin resource sharing (CORS) mechanism: third-parties accepting such requests can add special headers to their HTTP responses to disable some browsers restrictions. This makes it difficult for a webpage to probe third-party services' health: every response not having CORS headers (the default) will be blocked. Similarly, one webpage can create tabs and iframes pointing to third-party services, but accessing the properties of these resources is restricted by browsers.

2.3 Browser measurements using extensions

The cross-origin security restrictions motivate the design of a browser extension for third-party service health check. DiagSys' browser extension relies on the *WebExtension API* [22]. (At the time of writing, this is the standard method for building extensions for Mozilla Firefox, Google Chrome and Microsoft Edge.) The installation of this extension is optional: browsers can also request a service health check *as seen from landmarks* through client emulation (Figure 1). With the appropriate permissions set, a WebExtension can *intercept* and *modify* any web request made to any third-party service. A first solution to disable cross-origin security would be to insert the CORS headers in every response. This would open a major security hole in the browser security model, as there is no standard method to add these headers only for requests originating from trusted sources. This is also not sufficient to verify the global health of a third-party, as it would require to know the full list of needed web resources in advance.

Our solution is depicted in Figure 2. We first inject a WebExtension content script into trusted webpages. The content script registers itself with the main webpage script ② and relays messages from the main script to the extension's background script ③. In this setup, the communication between each script is secured by safe *Messaging APIs* provided by browsers. When the main script requests a health check to a specific service, the background script creates an iframe in the extension background page ④ (this operation is invisible to users). Iframes are used to fully load a service, from the initial HTML document to the very last resource load. Another content script is injected in background iframes to obtain the resources timings and send them back to the main script (via the reverse path ④ → ③ → ②). The requests originating from our extension's background iframes can be identified using their unforgeable `originUrl`. We can thereby safely update CORS HTTP headers of responses corresponding to these requests ⑤. More specifically, we remove the `X-Frame-Options` and `Content-Security-Policy` headers to allow loading the third-party service from iframes, and we set `Timing-Allow-Origin` to `*` to enable precise resource timings measurements. It is thereby possible to estimate third-party service QoE through page load time measurement. Some services detect that they are being loaded from iframes and decide to stop loading or to take ownership of the parent frame (i.e. the extension background page). Thankfully, iframes can be *sandboxed* with a limited set of features which avoids losing control of the background page.

2.4 Privacy considerations

Given recent data protection regulations, some rules must be enforced to protect personally identifying information (PII) and sensitive personal information (SPI). While the only PII that could be useful in our case is the public IP address of an user, we were careful to *not collect any user browsing history*. After internal legal validation for GDPR compliance, we provide two modes of operation for DiagSys. In the default mode, all data is anonymized and no PII/SPI is collected (even the client's IP address is removed). The second mode requires the explicit written consent of the end-user and enables the collection of the client's IP address. In particular, participants who contributed to the data discussed in the next section were recruited through community mailing lists, with the guarantee that their data would be deleted after 2 years.

3 Results

DiagSys has been running continuously since October 2019, with more than one million measurement samples collected over 26 landmark servers and 20 third-party services. Around 170 unique end-users are providing measurements, and among them 32 have installed our browser extension and enabled background measurements. We recall that landmarks emulate additional users to provide supplementary and more continuous measurement samples. In this section, we present some case studies extracted from this early dataset.

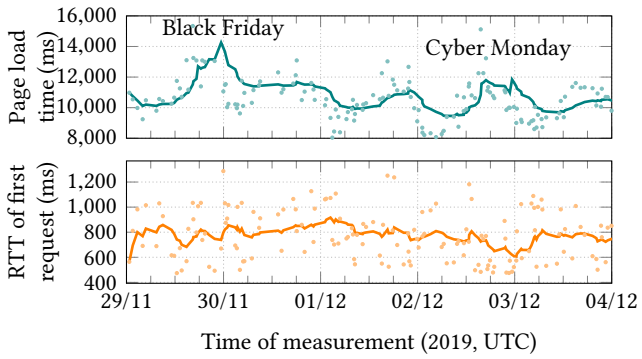

Figure 3. `walmart.com` page load time and RTT of first HTTP request, as measured by DiagSys. We observe increases in page load time during Black Friday and Cyber Monday, in contrast with first request timings.

Monitoring page load times. Users who run the DiagSys extension allow to regularly measure the page load time (PLT) of selected third-party services. This can be used to estimate a web service QoE and detect local and global perturbations. As an example, Figure 3 plots the PLT of `walmart.com` during Black Friday with visible slowdown periods during expected traffic peaks. (The measurements were taken by a landmark in Paris.) We find that measuring PLT is more insightful than just measuring the first request's RTT, as depicted in the lower part of Figure 3: PLT accounts for *every* remote resource, including scripts and medias from other third-parties. Similar highly-correlated patterns have been observed for different landmarks and users, with different amplitudes. This demonstrates the need for full browser emulation, as provided by DiagSys.

Highlighting regional differences. The user diversity of DiagSys makes it possible to spot differences in content served by third-party services to different visitors. Table 2 shows the number of unique resources fetched by 3 landmarks around the globe with identical configuration measuring `cnn.com` around the same time. We notice that the European landmark loads *far fewer resources* than its peers, despite receiving the same HTML page (assumed by identical uncompressed body size). When we look at the difference in loaded resources, we find than non-European visitors load more content related to analytics and ad tracking.

Impact of user mobility. Many users are mobile and use multiple methods to connect their devices to Internet (wired, cellular, Wi-Fi, ...) [12]. As a result, measurement samples from one user can be very diverse across time. We evaluated this diversity by using mobility ground truth for a specific volunteer that used both Wi-Fi and wired connections as provided by their ISP. As expected, we observed clear differences in measured throughput between wired and wireless modes.

Table 2. Resources fetched by cnn.com for different regions (18/05/2020 16:35 UTC)

Region	Europe	USA	Japan
HTML Body Size (bytes)			
Compressed	156'908	156'911	156'910
Uncompressed	1'132'658	1'132'658	1'132'658
Number of loaded resources			
style	20	19	19
script	28	61	61
query	20	53	54
iframe	3	13	14
media	7	48	57
total	**78**	**194**	**205**

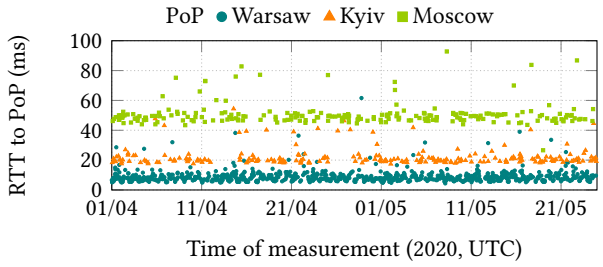

Figure 4. RTT between a landmark in Warsaw and Cloudflare. Three different PoP are regularly serving traffic with up to 5× more latency from Moscow than from Warsaw.

More surprisingly, we noted that some landmarks needed to retransmit *around 10% of packets* with wired connection, compared to zero retransmissions with Wi-Fi. We use BBR as the default TCP congestion control algorithm in landmarks, and this is certainly the reason why we are observing this behavior, as previously studied by Cao et al. [3].

Monitoring CDN performance. We measured the diversity of Cloudflare PoPs chosen for each user, and found that most users always reach the CDN network from the same PoP (we recall that the selected PoP is added in every HTTP response's header). However, we found that PoPs were much more dynamic for some regions, and we take as an example one landmark located in Warsaw's OVH datacenter. While most (66%) of HTTP responses were served from Warsaw's PoP (Figure 4) with a median latency of 8ms, the remaining responses were served from either Kyiv with twice that latency and even Moscow with a median latency of 50ms. Because this observation spans over two months of measurements taken from a static landmark, it is possible that this behavior is due to some load-balancing mechanism or non-optimal configuration.

Effects of network load and routing. DiagSys does not have any information about network topology and BGP announcements. Yet, the collective knowledge gathered from users and landmarks is sufficient to detect and analyze changes

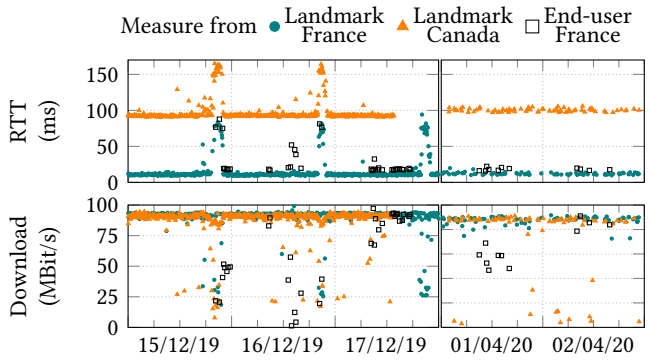

Figure 5. Evolution of RTT and download throughput from two landmarks to a home-network landmark in France. There is a pattern of anomalies during evenings in the first time frame, probably due to congested link.

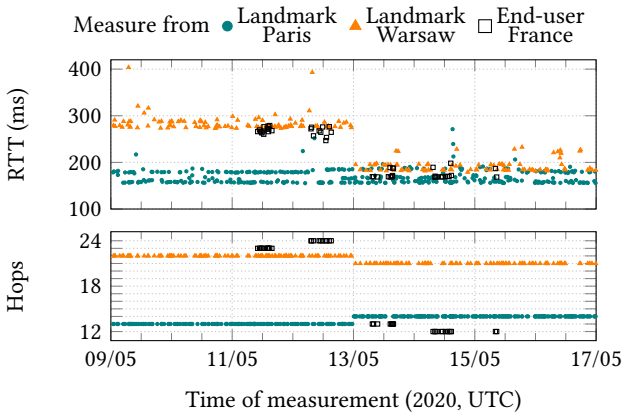

Figure 6. RTT and number of hops in traceroute to a landmark hosted in Vultr Singapore region. We can see a change in routing strategy on 13/05/2020 at midnight with an immediate decrease of RTT for some regions.

in Internet paths and links, overcoming the opacity of ISPs networks. As a first example, we study the performance of a landmark hosted in a home network served by the French ISP "Free". Figure 5 shows the RTT and download throughput of this landmark as measured from two other landmarks in France and Canada and one end-user from France. Measures from end-users are sparser and noisier: this is expected, as their devices are not powered continuously and may have less reliable network connections. During the first time frame, we clearly see anomalies during evenings: the landmark's host confirmed that he encountered QoE degradation, which suggests that the root cause came from an overloaded link in the Free network. After a few months, the anomalies disappeared (second time frame in Figure 5). In a second example (Figure 6), we detected an important routing change between some users and a landmark in Singapore. The RTT to Singapore measured by our landmark in Warsaw dropped by 30%, with one less hop in the reverse traceroute. When looking at the traceroute details, we can assume that the traffic was redirected on May 13 from NTT (AS 2914) to GTT (AS 3257)—two Tier 1 networks. We used BGPlay [14], a routing history visualizer and confirmed this finding. The observations are similar for an end-user in France, but no change is noticeable for another landmark in Paris with already good performance before May 13. It would have been difficult to detect this routing change from within browsers using BGP announcements alone.

4 Related work

Several studies have taken the browser perspective to collect network measurements [5, 6, 9, 11]. Netalyzr [9] runs a wide set of network tests and collects the corresponding network metrics, using a java applet that runs within the browser and a set of dedicated measurement servers. Fathom [5] is also a browser-based measurement framework with support for low-level socket primitives, but only supports Firefox up to version 57. In our work, we collect metrics to multiple landmarks using JavaScript only. We also propose an optional WebExtension for background third-party monitoring from end-users' browsers. Furthermore, we enrich our third-party measurements dataset with samples from landmark servers running a headless Chrome browser. Advanced browser-based techniques that give more accurate estimations of QoE could also be used by DiagSys [4].

Outside the browser, Sundaresan et al. [15, 17, 20] use home gateways to measure and assess the broadband internet performance. These approaches work well to monitor and analyze the last mile broadband connection of internet users. Only [17] measures web page load times. It relies on router-based Web measurement tools to deconstruct Web page load time. Running in a gateway, it does not reflect the actual performance as observed by an end-user using a browser.

5 Conclusion

In this paper we proposed and described DiagSys, a crowd-sourced data collection system targeted at monitoring networks and third party web-services. DiagSys relies on measurement implemented in JavaScript running in browsers as a dedicated extension or embedded within webpages, while being compatible with the recent security restrictions of modern browsers. DiagSys also uses opportunistically-deployed *landmark servers* that act as reference points for measures. These servers also run headless browsers and execute the same JavaScript code to continuously provide measurement samples. Despite a quite recent and therefore limited deployment we showed that DiagSys can provide sharp insights regarding third-party web service and CDN behavior, routing and network load issues.

References

[1] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. 2006. Avoiding traceroute anomalies with Paris traceroute. In *IMC*.

[2] Maximilian Bachl, Renata Teixeira, Timur Friedman, Claudio Sacchi, and Anna-Kaisa Pietilainen. 2016. Collaborative Home Network Troubleshooting. *CoRR* (2016). https://hal.inria.fr/hal-01415767

[3] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. 2019. When to use and when not to use BBR: An empirical analysis and evaluation study. In *IMC*.

[4] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. 2018. Narrowing the Gap Between QoS Metrics and Web QoE Using Above-the-fold Metrics. In *PAM*.

[5] Mohan Dhawan, Justin Samuel, Renata Teixeira, Christian Kreibich, Mark Allman, Nicholas Weaver, and Vern Paxson. 2012. Fathom: a browser-based network measurement platform. In *IMC*.

[6] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P Gummadi, Ratul Mahajan, and Stefan Saroiu. 2010. Glasnost : Enabling End Users to Detect Traffic Differentiation. In *NSDI*.

[7] IETF. 2011. *The WebSocket Protocol*. https://tools.ietf.org/html/rfc6455

[8] Diana Joumblatt, Renata Teixeira, Jaideep Chandrashekar, and Nina Taft. 2011. HostView: Annotating end-host performance measurements with user feedback. *ACM SIGMETRICS Performance Evaluation Review* 38, 3 (2011).

[9] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. 2010. Netalyzr: Illuminating The Edge Network. In *IMC*.

[10] Weichao Li, Ricky K.P. Mok, Rocky K.C. Chang, and Waiting W.T. Fok. 2013. Appraising the delay accuracy in browser-based network measurement. In *IMC*.

[11] Measurement Lab. 2020. NDT (Network Diagnostic Tool). https://www.measurementlab.net/tests/ndt

[12] Vikas Mishra, Pierre Laperdrix, Antoine Vastel, Walter Rudametkin, and Martin Lopatka. 2020. Don't count me out: On the relevance of IP addresses in the tracking ecosystem. In *The Web Conference*.

[13] Ookla. 2020. *Speedtest Servers*. https://www.speedtest.net/speedtest-servers

[14] RIPE NCC. 2020. BGPlay. https://stat.ripe.net/widget/bgplay

[15] Srikanth Sundaresan, Sam Burnett, Nick Feamster, and Walter De Donato. 2014. Bismark: A testbed for deploying measurements and applications in broadband access networks. In *ATC*.

[16] Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. 2016. Home network or access link? Locating last-mile downstream throughput bottlenecks. In *PAM*.

[17] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, and Nazanin Magharei. 2013. Measuring and mitigating web performance bottlenecks in broadband access networks. In *IMC*.

[18] Srikanth Sundaresan, Nick Feamster, Renata Teixeira, Srikanth Sundaresan, Nick Feamster, and Renata Teixeira. 2015. Measuring the Performance of User Traffic in Home Wireless Networks. In *PAM*.

[19] Srikanth Sundaresan, Yan Grunenberger, Nick Feamster, Dina Papagiannaki, Dave Levin, and Renata Teixeira. 2013. WTF? Locating Performance Problems in Home Networks. *CoRR* (2013). http://hdl.handle.net/1853/46991

[20] Srikanth Sundaresan, Renata Teixeira, Georgia Tech, Nick Feamster, Antonio Pescapè, and Sam Crawford. 2011. Broadband Internet Performance : A View From the Gateway. In *SIGCOMM*.

[21] Yves Vanaubel, Jean-Romain Luttringer, Pascal Merindol, Jean-Jacques Pansiot, and Benoit Donnet. 2019. TNT, Watch me Explode: A Light in the Dark for Revealing MPLS Tunnels. In *TMA*.

[22] W3C. 2020. *Browser Extensions (Draft)*. https://browserext.github.io/browserext/

[23] W3C. 2020. Resource Timing Level 2. https://www.w3.org/TR/resource-timing-2/

[24] W3Techs. 2020. *Usage statistics of reverse proxy services for websites*. https://w3techs.com/technologies/overview/proxy