WebRTC Testing: State of the Art

Boni García, Micael Gallego, Francisco Gortázar and Eduardo Jiménez
Universidad Rey Juan Carlos, Calle Tulipán S/N, 28933 Móstoles, Spain
{boni.garcia, micael.gallego, francisco.gortazar, edu.jg}@urjc.es

Keywords: WebRTC, Software Testing, Software Quality.

Abstract: WebRTC is the umbrella term for a number of emerging technologies that extends the web browsing model to exchange real-time media (Voice over IP, VoIP) with other browsers. The mechanisms to provide quality assurance for WebRTC are key to release this kind of applications to production environments. Nevertheless, testing WebRTC based application, consistently automated fashion is a challenging problem. The aim of this piece of research is to provide a comprehensive summary of the current trends in the domain of WebRTC testing. For the sake of completeness, we have carried out this survey by aggregating the results from three different sources of information: i) Scientific and academia research papers; ii) WebRTC testing tools (both commercial and open source); iii) "Grey literature", that is, materials produced by organizations outside of the traditional commercial or academic publishing and distribution channels.

1. INTRODUCTION

Multimedia applications and services are becoming the main force of the Internet. A recent forecast by Cisco (Index, 2016) shows that IP video traffic will be 82 percent of all consumer Internet traffic by 2020.

Among the diversity and multiplicity of multimedia technologies, in this paper we focus on Web Real-Time Communications (WebRTC), whichis a set of emerging technologies and APIs having the ambition of bringing high-quality RTC to the Web (Loreto and Romano, 2014). WebRTC is a joint standardization effort between the World Wide Web Consortium[1] (W3C) and the Internet Engineering Task Force[2] (IETF). On the one hand, W3C is defining the JavaScript APIs in so-called WebRTC 1.0 and the standard HTML5 tags to enable peer-to-peer (P2P) connections between web-enabled devices. The WebRTC APIs are *getUserMedia*: which gain access to camera, microphone, or screen device; *RTCPeerConnection*: encoding and decoding media, sends it over the network, NAT (Network Address Translation) traversal; and *RTCDataChannel*: send arbitrary data directly between browser with low latency. On the other hand, IETF is defining the underlying communication protocols, such as SRTP (Secure Real-time Transport Protocol), SDP (Session Description Protocol), or ICE (Interactive Connectivity Establishment), for the setup and management of a reliable communication channel between browsers.

WebRTC has come a long way since its inception in May 2011. Among its highlights, we can point out the interoperability between Chrome and Firefox browsers in 2013, and the support for Android mobile in 2014 (Kaul, 2015). Moreover, market momentum is expected to continue growing. A recent analysts report predicts that with Apple and Microsoft supporting WebRTC in their browsers, there might be 7 billion devices compliant WebRTC by 2020 (Sal and Rebbeck, 2014).

Since the beginning of 2014, a new initiative has seen the light in the W3C: the ORTC (Object Real-time Communications) Community Group, which has initially been identified as a clear opponent to WebRTC. Nowadays, ORTC and WebRTC are converging in the so-called "Next Version" of WebRTC (sometimes called WebRTC 1.1 or WebRTC-NV).

Due to this strong growth rate of WebRTC, it is imperative for software engineers and testers to have a strategy in place in order to assess WebRTC

[1] http://www.w3.org/TR/webrtc/
[2] http://tools.ietf.org/wg/rtcweb/

applications efficiently. Nevertheless, testing WebRTC based application in a consistently automated fashion is a challenging problem. When developers use WebRTC for integrating audio and video communications into their web applications, they usually consume a complex media pipeline transporting multimedia information, and therefore this kind of applications cannot be tested using the usual simple comparison-based oracles. For example, validating the functional correctness of a WebRTC application requires the ability of evaluating aspects such as media connectivity (e.g. whether the media bits are being sent end-to-end) or media continuity (e.g. whether the media is decodable).

This paper presents a survey of the state of the art in the domain of testing for WebRTC-based applications. The aim of this piece of research is to provide a comprehensive summary of the current trends in this domain for researchers, engineers, and practitioners. For the shake of completeness, we have carried out this survey by aggregating the results from three different sources of information. First, we study the most remarkable scientific papers and articles in peer-reviewed journals, magazine, and international conferences. Second, we analyze the features of public available WebRTC testing tools, both commercial and open source. Third, we summarize several contributions available in the so-called “grey literature”, that is, materials produced by organizations outside of the traditional commercial or academic publishing and distribution channels. We find examples of grey literature on technical reports, white papers, newsletters, blogs, among others.

The remainder of this paper is structured as follows. Section 2 provides a brief overview of the background of this work (i.e. software testing and WebRTC). Section 3 presents a collection of scientific and academic contributions on WebRTC testing. Section 4 summarizes the main features of several tools on this domain. Section 5 details several publications of the grey literature. Finally, section 6 concludes the paper summarizing the most remarkable findings as conclusions of this survey.

2. BACKGROUND

Verification and Validation (V&V) is the set of techniques that assess software products and services. Software testing is the most commonly performed activity within V&V. Given a piece of code, software testing consists of observing a sample of executions (test cases), and giving a verdict over them (Bertolino, 2007).

Testing of web applications shares the same objectives of traditional application testing, i.e. to ensure quality and finding defects in the required functionality and services. Due to its heterogeneity, web applications present important challenges for their quality assurance and testing (Li et al., 2014). In order to perform a complete assessment procedure, it is required to evaluate web applications from functional and non-functional perspectives. According to Di Lucca and Fasolino, the most important non-functional requirements for web applications are performance, load, stress, compatibility, accessibility, usability, and security (Di Lucca and Fasolino, 2006).

WebRTC applications and services enable human-to-human communication. The real-time nature of WebRTC traffic makes QoS (Quality of Service) parameters such as network latency, network jitter or packet loss to affect significantly, and in non-trivial ways, the end-user's QoE (Quality of Experience). The most widely accepted way to classify QoE metrics is based on subjective or objective methods (Jain and Scheirer, 2014). Subjective methods are conducted to obtain information on the quality of multimedia services using opinion scores, while objective methods are used to estimate the network performance using models that approximate the results of subjective quality evaluation. Subjective QoE measurement is time consuming, and is not particularly applicable in a production environment. Instead of directly collecting quality information, objective methods can be used, namely:

- Traditional point-based metrics. For example, peak signal-to-noise ratio (PSNR), which is the proportion between the maximum signal power and the corruption noise power (Huynh-Thu and Ghanbari, 2008).
- Natural visual characteristics oriented metrics. For example, structural similarity (SSIM), which is a method for predicting the perceived quality of images and videos based on its similarity (Wang et al., 2004).
- Perceptual oriented metrics. In these metrics, is predicted using Mean Opinion Score (MOS) ratings. The typical MOS scale has five-points: 1 = bad, 2 = poor, 3 = fair, 4 = good and 5 = excellent. For example, perceptual evaluation of speech quality (PESQ) for audio, and perceptual evaluation of video quality (PEVQ) for video (Viswanathan and Viswanathan, 2005).

3. SCIENTIFIC AND ACADEMIC RESEARCH

This section provides a summary of the main contributions found concerning testing of WebRTC applications in peer reviewed contributions in journals, magazines, and international conferences. In order to carry out this study, the following search engines for scientific and academic have been used:

- Google Scholar[3] is a free academic search engine that indexes academic information from various online web resources.
- CiteSeerx[4] is a digital library and an online academic journal that offer information within the field of computer science.
- Microsoft Academic Research[5] is yet another top search engine for academic resources.
- ScienceDirect[6] is a full-text scientific database offering journal articles and books.

Table 1 summarizes the selection of papers. (Sandholm et al., 2013) presents a solution for tunneling WebRTC traffic using JavaScript Session Establishment Protocol (JSEP). Then, the authors carried out some experimentation in order to show the evolution of several QoS parameters such as round-trip-time (RTT) or jitter in different network configurations (WiFi and Ethernet) and number of users.

(Cinar and Melvin, 2014) uses a black-box testing technique to evaluate, via PESQ, the voice quality of WebRTC sessions under varying network delay and jitter. In this paper, network emulators are employed to implement the delay and jitter variations. The results highlight the dangers of black-box testing, whereby test-bed issues can result in very misleading results.

(Vucic and Skorin-Kapov, 2015) study QoE for mobile video conferencing focusing on the impact of different smartphone configurations (CPU, display size, and resolution). They conduct subjective studies involving interactive three-party audiovisual conversations based on WebRTC technology in a natural environment over a Wi-Fi network with symmetric and asymmetric bandwidths. The finding of this work shows that different device factors impact clearly on user QoE.

(Amirante et al., 2016) present Jattack, a general-purpose WebRTC stressing tool capable to simulate the activities of multiple WebRTC sessions. This tool is based on the Janus media server, which is able to create a big number of WebRTC PeerConnection to stress the SUT. Jattack monitors de physical parameter of the SUT (CPU. Memory) and gather the number of negative acknowledgments (NACKs) to estimate the QoE of the system.

(Taheri et al., 2015) introduces WebRTCBench, an open source tool for performance assessment of WebRTC implementations which allows testing applications making use of video and audio through WebRTC standards and collects performance indicators. It consists of a Node.js application with a HTML5 client, supporting Chrome and Firefox browsers.

(Spoiala et al., 2016) compares the performance of the WebRTC media server Kurento hosted in two different platforms: virtual machines and Docker containers. The authors of this work concluded that the Docker performance is better than Kernel-based Virtual Machine (KVM), especially for latency, which is critical metric for real-time applications.

(García et al., 2016a) presents the Kurento Testing Framework (KTF), a high-level framework aimed to carry out different kind of testing activities for WebRTC services. KTF provides several mechanisms for assessing the functional parameters (media communication events, detection of color), performance (monitor system latency measurement based on the color comparison means sent and received), and QoE (evaluation audio quality through PESQ). KTF uses Selenium WebDriver/Grid for automatic interaction with WebRTC applications. This work is continued in (García et al., 2016a), in which the framework is extended QoE indicators for video, concretely structural similarity (SSIM) and peak signal-to-noise ratio (PSNR). Moreover, the framework proposed in this work is able to calculate the end-to-end latency in a WebRTC connection using an Optical Character Recognition (OCR) applied to the media sent by a browser and received by other one. Finally, the framework proposed is completed by adding the capability of creating fake browsers in order to generate a huge load of WebRTC traffic in the SUT (García et al., 2017). Moreover, KTF supports the use of Docker[7] containers in order to configure complex networking scenarios where NAT traversal, firewalls or different traffic loss, latency, and bandwidths are enforced.

[3] http://scholar.google.com/
[4] http://citeseerx.ist.psu.edu/
[5] http://academic.research.microsoft.com/
[6] http://www.sciencedirect.com/
[7] https://www.docker.com/

Table 1: Selection of WebRTC testing research papers.

Title	Keywords	Reference
On-Demand WebRTC Tunneling in Restricted Networks	Black-box testing, QoS, networking	(Sandholm et al., 2013)
WebRTC quality assessment: Dangers of black-box testing	Black-box testing, QoS, objective QoE	(Cinar and Melvin, 2014)
The impact of mobile device factors on QoE for multi-party video conferencing via WebRTC	Subjective QoE	(Vucic and Skorin-Kapov, 2015)
WebRTCbench: a benchmark for performance assessment of WebRTC implementations	Performance testing, framework, open source	(Taheri et al., 2015)
Jattack: a WebRTC load testing tool	Load testing, QoS, framework	(Amirante et al., 2016)
Performance comparison of a WebRTC server on Docker versus virtual machine	Load testing, QoS	(Spoiala et al., 2016)
Testing Framework for WebRTC Services	Black-box testing, QoE, QoS, framework, open source	(García et al., 2016a)
Analysis of Video Quality and End-to-End Latency in WebRTC	Load testing, QoS, objective QoE, framework, open source	(García et al., 2016b)
WebRTC Testing: Challenges and Practical Solutions	Load testing, QoS, objective QoE, framework, networking	(García et al., 2017)

4. WEBRTC TESTING TOOLS

Traditionally, tools such as Apache JMeter[8] have been used to analyze and measure the performance of a variety of services, with a focus on Web applications (Halili, 2008). Nevertheless, these kinds of tools are not valid to test WebRTC applications because it is required to use browsers implementing the WebRTC stack.

In order to perform tests for WebRTC applications, it is a must to be able to automate test execution using real web browsers (Chrome, Firefox, and so on). The well-known open source testing framework Selenium[9] is capable of drive a browser automatically using different programming languages (Java, C#, Python, Ruby, PHP, Perl, or JavaScript). To that aim, the following Selenium projects are very useful (Avasarala, 2014):

- Selenium Remote Control (RC): This piece of software injected a JavaScript library (called Selenium Core) in the SUT. This library was controlled with an intermediate component called Selenium RC Server which receives requests from the test code (see Figure 1-i). Selenium RC had important security problems due to Same-Origin Policy. For that reason, it was deprecated on 2016 in favor of Selenium WebDriver.
- Selenium WebDriver makes direct calls to the browser using each browser's native support for automation. The language bindings provided by Selenium WebDriver (labeled as *Test* in Figure 1-ii) communicates with and a browser-specific binary which acts as a bridge between real browser (Liang and Collins, 2016). For instance, this binary is called chromedriver[10] for Chrome and geckodriver[11] for Firefox. The communication between the *Test* and the driver is done with JSON messages over HTTP using the so-called JSON Wire Protocol. This mechanism, originally proposed by the WebDriver team is standardized in the W3C WebDriver[12] API.
- Selenium Grid: It allows distributing browser execution on remote machines. There are a number of *Nodes*, each running on different operating systems and with different browsers. The *Hub* server keeps a track of the nodes and proxies requests to them (Figure 1-iii).

The inconvenient of Selenium WebDriver is the need of browser instances installed in the local machine running the test. A common workaround to solve this problem in operative systems without graphical user interface is using Xvfb[13] (X virtual framebuffer) as display server implementing the X11 server protocol or *dockerized* browsers (i.e. browser running inside a Docker container).

[8] http://jmeter.apache.org/
[9] http://www.seleniumhq.org/
[10] https://sites.google.com/a/chromium.org/chromedriver/
[11] https://github.com/mozilla/geckodriver
[12] https://www.w3.org/TR/webdriver/
[13] https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml

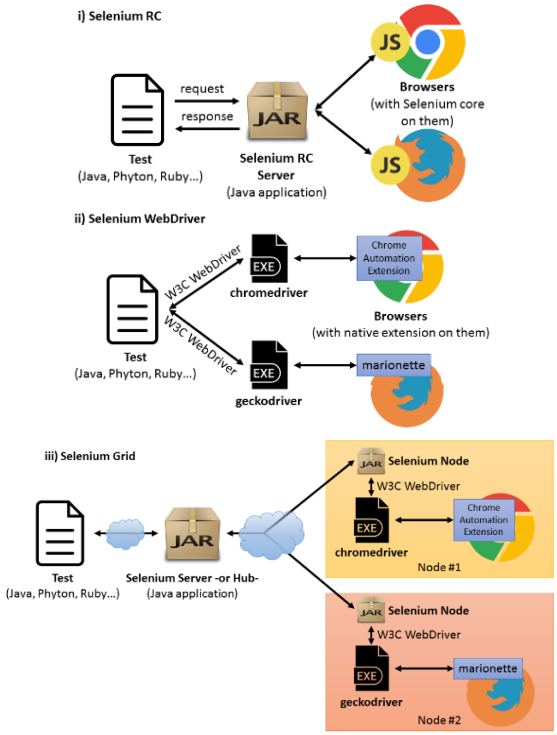

Figure 1: i) Selenium RC; ii) Selenium WebDriver; iii) Selenium Grid

Moreover, nowadays there are several cloud testing providers which allows to run Selenium Grid tests in different browsers. These providers are:

- Saucelabs[14] is a commercial PaaS (Platform as a Service) cloud solution to support remote testing based on supporting many combinations of platform (Linux, Windows, Mac OS X, Android, iOS), browser (Chrome, Firefox, Opera, etc.), and browser versions (including beta and development releases).
- BrowserStack[15] is another commercial PaaS which provides instant access to mobile and desktop browsers for live and automated testing across different browser (Chrome, Firefox, Edge, Opera, etc.), operating systems (Windows and Mac OS X) and mobile real devices (iOS, Android, Windows Phone).

Another framework aimed to create automated end-to-end tests for web applications is Nightwatch.js[16]. It has been written in Node.js and use W3C WebDriver API to consume browsers. It can be integrated with cloud services support such as SauceLabs and BrowserStack. At the time of this writing the Nightwatch.js is developing its own cloud-based platform called Nightcloud[17].

Finally, we find a commercial tool specifically designed to carry out testing of WebRTC applications called TestRTC[18]. It can be seen as an integrated platform aimed to test, monitor and analyze WebRTC-based communications. The main features of TestRTC are:

- Use of real browsers. This is one the major strategic decisions of TestRTC: support only real web browsers as agents to assess the SUT instead of building something on top of WebRTC directly. At the time of this writing, TestRTC supports Chrome and Firefox browsers (stable, previous stable, beta, unstable). Thanks to its own global cloud infrastructure, TestRTC allows to choose these browsers from different locations (east US, west US, Europe, and Asia).
- JavaScript API. Developers using TestRTC can write test scripts using an API built on the top of Nightwatch.js. These tests can be directly uploaded to the web dashboard.
- Network awareness. TestRTC allows to configure the underlying network with custom setups, including different firewall and NAT configurations, different bitrates (static and dynamic ones), and different packet loss. To simplify this task, it provides preconfigured scenarios of the typical access networks, such as 3G, 4G, DSL, or WiFi among others.
- Signaling protocol agnostic. TestRTC can test standards-based signaling protocols (such as SIP over WebSocket, XMPP over WebSocket, or BOSH) and proprietary (based on either WebSocket or HTTP(S) and REST).
- WebRTC tests at scale. TestRTC allows to run different media sessions at the same time. In addition, it allows to leverage the number of concurrent users per session
- TestRTC allows to monitor Key Performance Indicators (KPIs) such as channel types, bitrate, timing, packet loss, and jitter.
- WebRTC-internals analyzer. Chrome allows to download the PeerConnection updates and stats data (as defined in the W3C WebRTC's statistic API[19]) in JSON notation. This data can be uploaded with drag-and-drop to the TestRTC web client and provided a detailed report about the data.

[14] https://saucelabs.com/
[15] https://www.browserstack.com/
[16] http://nightwatchjs.org/
[17] https://nightcloud.io/
[18] http://testrtc.com/
[19] https://www.w3.org/TR/webrtc-stats/

Table 2: Selection of WebRTC testing grey literature.

Title	Keywords	Reference
WebRTC Audio Quality Testing	Black-box testing, objective QoE	(Höglund, 2013a)
Automated Video Quality Measurements	Black-box testing, objective QoE	(Höglund, 2013b)
Chrome-Firefox WebRTC Interop Test	Interoperability testing	(Höglund, 2014)
Audio Testing - Automatic Gain Control	Black-box testing	(Höglund, 2015)
The WebRTC Troubleshooter: test.webrtc.org	Black-box testing, QoS	(Pascual, 2015)
Overcoming the Challenges in Testing WebRTC Services	Testing methodology	(Levent-levi, 2015)
Quality Assurance for WebRTC Services	Testing methodology	(Levent-levi, 2016)

- Live preview of the remote browser tests by means of VNC (Virtual Network Computing) connections.
- Use customizable user media for WebRTC sessions, namely VGA, HD, Full-HD, only-audio, among others.
- Reporting. The TestRTC dashboard shows reports by collecting and calculating KPIs related to the voice and video streams.
- Test history. TestRTC stores the full test execution history, allowing developers to keep tracking the evolution of the SUT in time.
- Layer of synchronization across browsers, so a user can indicate in the script tasks that wait for events to occur on other browsers that are running in parallel.

5. WEBRTC TESTING GREY LITERATURE

Grey literature is an important source of information for complete review syntheses. There are many kinds of grey literature, such as dissertations, technical reports, white papers, government websites, newsletters, blogs and other social networking sites are examples of grey literature.

In order to narrow the problem of seeking the grey literature of WebRTC testing, in this work we are going to limit the spectrum to a set of important websites and initiatives focused on WebRTC and/or testing, namely:

- Google Testing Blog[20]. Official Google blog for testing activities.
- Google Test Automation Conference[21] (GTAC). Annual test automation conference hosted by Google.
- WebRTC Conference[22]. Global summit aimed to bring together everyone in the WebRTC ecosystem.
- BlogGeek.me[23]. Blog of Tsahi Levent-Levi, independent analyst and consultant for WebRTC and a co-founder at TestRTC.
- WebRtcHacks.com[24]. One of most remarkable blogs in the WebRTC arena, maintained by Chad Hart, Victor Pascual, Tsahi Levent-Levi, and Philipp Hancke.

As a result, several contributions were selected and summarized in Table 2.

(Höglund, 2013a) describes a test aimed to analyze the audio quality of a WebRTC one-to-one video call. The audio is recorded in the receiver side directly recording what the audio system sends to default audio out (like speakers or headphones). In order to compare the sender audio to the audio recorded in the receiver, the algorithm PESQ is used. As a result, a MOS score is assessed (it should be at least 4 out of 5).

(Höglund, 2013b) illustrates a test aimed to measure the video degradation in a WebRTC session. This test was run continuously with regression monitoring in the Chromium testbed. In order to synchronize the media sent and received by a pair of browsers, each frame was identified single a single barcode. Finally, two different algorithms were used to evaluate the video quality: PSNR and SSIM.

(Höglund, 2014) describes an interoperability automated test between Chrome and Firefox. To carry out the test, a Chromium browser is launched as part of every Google Chromium browser test. Then, a Python library called *mozrunner* is used to launch Firefox. In order to feed the WebRTC media session, the test use the convenient Chrome `--use-fake-device-for-media-stream` flag that feeds the user media which is a spinning green ball plus a timestamp. In order to establish the WebRTC connection between peer, first it is needed to

[20] https://testing.googleblog.com/

[21] https://developers.google.com/google-test-automation-conference/

[22] https://webrtc-conference.com/

[23] https://bloggeek.me/

[24] https://webrtchacks.com/

exchange SDP signaling messages. Once the media session is established, in order to verify that the audio is playing, the WebRTC stats are read to measure the audio track energy level. To assess video, the test verifies the CSS opacity property of the HTML5 video tag. Moreover, a built-in JavaScript function is used to control the color change ratio of the received media.

Automatic Gain Control (AGC) is a mechanism provided out of the box by WebRTC aimed to adjust the audio of a WebRTC stream in order to make it louder and clearer for the receiver side. (Höglund, 2015) describes an automated end-to-end test to verify AGC. This test uses the Chrome flag `--use-file-for-fake-audio-capture` to inject a custom audio file in the WebRTC stream. The audio level difference is calculated as the subtraction of the audio level in the receiver less the audio level in the sender side. In order to avoid biases due to packet loss and clock drifts (i.e. audio differences due to sample clocks on the sending and receiving sound cards are not perfectly synced) the reference audio file had several small silences among the real audio. The audio was trimmed using these silences and the resulting parts are computed.

(Pascual, 2015) describes the WebRTC Troubleshooter[25], which is a website that provides a set of tests that can be easily run by a user to help diagnose WebRTC related issues. These tests are focused on verify: i) Microphone: audio capture; ii) Camera: check resolution; iii) Network: UDP/TCP and IPv6 connectivity; iv) Connectivity: relay (verifies connections can be established between peers through a TURN server), reflexive (verifies connections can be established between peers through NAT), host (verifies connections can be established between peers with the same IP address); v) Throughput: throughput and video bandwidth

(Levent-levi, 2015) illustrates 5 challenges in testing WebRTC services: i) browser version changes: different versions of browsers (stable, beta, dev, canary) should be consistently testes to avoid regressions and service breaks; ii) NAT traversal: WebRTC is not always P2P since between 5-50% of WebRTC sessions are relayed via TURN servers, and this should be properly tested by providing browsers in remote locations; iii) test at scale should be taking into account (in other words, the question "is my SUT able to scale to thousands of user?" should be properly addressed); iv) service uptime: WebRTC involves heterogenous infrastructure that should be constantly tested in order to verify that our SUT is working every time; v) orchestration: test scripts can be designed to be simple and should be executed in parallel to simulate the real conditions of the system.

(Levent-levi, 2015) provides 7 suggestions to create an WebRTC testbed: i) use real browsers, which can be easily driven by Selenium WebDriver; ii) handle the media feed using the Chrome support for faking media device and FFMPEG[26] to create custom video files in the required format by Chrome (Y4M); iii) use heterogenous setup: with network impairments (bitrate limitation, packet loss, jitter and latency), firewall configuration, and multiple browser locations; iv) handle synchronization of the multiple possible use cases (caller/callee, meeting point, conference, etc.); v) visualize the result: create meaningful reports as a result of the test execution; vi) establish the expectation: define how the test outcome should be in a precise way; vii) collect everything: chrome internal dump, browser console logs, media recordings, machine performance.

6. CONCLUSIONS

WebRTC is a set of technologies aimed to provide RTC media capabilities in an easy way to web applications. Despite the fact that it is still in its infancy, WebRTC services are more and more demanded by practitioners. In order to release these kinds of services to production, software engineers and testers demand testing mechanisms to assess the functional correctness of the SUT, but also the stability, scalability, compatibility, and security of these services. Moreover, WebRTC services involve complex, distributed and heterogeneous network topologies where failures or inefficiencies on any of the comprising components may prevent the service to operate offering a successful user experience.

This paper presents a comprehensive state of the art on WebRTC testing. We have made this survey by aggregating different sources of information: i) academia research papers; ii) WebRTC testing tools; iii) other sources (the so-called "grey literature"). Our findings show that there is an increasing interest in the evaluation of WebRTC services, both in the academic and research arena (integrating QoS and QoE in WebRTC testbeds) and the commercial domain (for example with complete WebRTC testing platforms such as TestRTC).

[25] https://test.webrtc.org/

[26] https://ffmpeg.org/

ACKNOWLEDGEMENTS

This work has been supported by the European Commission under project ElasTest (H2020-ICT-10-2016, GA-731535); by the Regional Government of Madrid (CM) under project Cloud4BigData (S2013/ICE-2894) cofunded by FSE & FEDER; and Spanish Government under project LERNIM (RTC-2016-4674-7) cofunded by the Ministry of Economy and Competitiveness, FEDER & AEI.

We would like to thank to Tsahi Levent-levi, author of BlogGeek.me and cofounder of TestRTC, for his valuable comments on this piece of research.

REFERENCES

Amirante, A., Castaldi, T., Miniero, L. and Romanoy, S. *Jattack: a WebRTC load testing tool*. IEEE, 2016.

Avasarala, S., 2014. *Selenium WebDriver practical guide*. Packt Publishing Ltd.

Bertolino, A., 2007, May. Software testing research: Achievements, challenges, dreams. In *2007 Future of Software Engineering* (pp. 85-103). IEEE Computer Society.

Cinar, Y. and Melvin, H. WebRTC quality assessment: Dangers of black-box testing. Digital Technologies, 2014 10th Int. Conference on, 2014. IEEE, 32-35.

Di Lucca, G.A. and Fasolino, A.R., 2006. Testing Web-based applications: The state of the art and future trends. *Information and Software Technology*, *48*(12), pp.1172-1186.

García, B., López-Fernández, L., Gallego, M., and Gortázar, F., 2016, Testing Framework for WebRTC Services. In *9th EAI International Conference on Mobile Multimedia Communications* (pp. 40-47).

Garcia, B., López-Fernández, L., Gortázar, F., and Gallego, M., 2016, Analysis of Video Quality and End-to-End Latency in WebRTC. In *Globecom Workshops (GC Wkshps), 2016 IEEE* (pp. 1-6). IEEE.

Garcia, B., Gortázar, F., López, L., Gallego, M., and París, M., 2017, WebRTC Testing: Challenges and Practical Solutions. IEEE Communication Standards. IEEE.

Halili, E.H., 2008. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing Ltd.

Höglund, P., 2013. WebRTC Audio Quality Testing. Available at: https://testing.googleblog.com/2013/11/webrtc-audio-quality-testing.html. Accessed on 23 March 2017

Höglund, P., 2013. Automated Video Quality Measurements. Available at: https://www.youtube.com/watch?v=IbLNm3LsMaw. Accessed on 23 March 2017

Höglund, P., 2014. Chrome-Firefox WebRTC Interop: https://testing.googleblog.com/2014/08/chrome-firefox-webrtc-interop-test-pt-1.html. Accessed on 23 March 2017.

Huynh-Thu, Q., and Ghanbari, M., 2008. Scope of validity of PSNR in image/video quality assessment. *Electronics letters*, *44*(13), 800-801.

Index, C.V.N., 2016. Forecast and methodology, 2015-2020 white paper. *Technical Report, Cisco, Tech. Rep.*

Jain, L.P., Scheirer, W.J. and Boult, T.E., 2004. Quality of experience. In *IEEE multimedia*.

Kaul, N., 2015. WebRTC and Its Impact on Testing. Available at: http://blog.smartbear.com/user-experience/webrtc-and-its-impact-on-testing/. Accessed on 23 March 2017

Levent-levi, T., 2015. Overcoming the Challenges in Testing WebRTC Services. Available at: https://www.slideshare.net/tsahil/overcoming-the-challenges-in-testing-webrtc-services. Accessed on 23 March 2017

Loreto, S. and Romano, S.P., 2014. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. " O'Reilly Media, Inc.".

Li, Y.F., Das, P.K. and Dowe, D.L., 2014. Two decades of Web application testing—A survey of recent advances. *Information Systems*, *43*, pp.20-54.

Pascual, V., 2015, The WebRTC Troubleshooter: test.webrtc.org. Available at: https://webrtchacks.com/webrtc-troubleshooter/. Accessed on 23 March 2017

Sal, S. and Rebbeck, and T., 2014. Operators need to engage with WebRTC and the opportunities it presents. In *October 2014 Analysis Mason*.

Sandholm, T., Magnusson, B., and Johnsson, B. A., 2013, On-Demand WebRTC Tunneling in Restricted Networks. *arXiv preprint arXiv:1312.6501*.

Spoiala, C. C., Calinciuc, A., Turcu, C. O., & Filote, C., 2016, Performance comparison of a WebRTC server on Docker versus virtual machine. In *Development and Application Systems (DAS), 2016 International Conference on* (pp. 295-298). IEEE.

Taheri, S., Beni, L.A., Veidenbaum, A.V., Nicolau, A., Cammarota, R., Qiu, J., Lu, Q. and Haghighat, M.R., 2015, October. WebRTCbench: a benchmark for performance assessment of webRTC implementations. In *Embedded Systems For Real-time Multimedia, 2015 13th IEEE Symposium on* (pp. 1-7). IEEE.

Viswanathan, M., and Viswanathan, M., 2005. Measuring speech quality for text-to-speech systems: development and assessment of a modified mean opinion score (MOS) scale. *Computer Speech & Language*, *19*(1), 55-83.

Vucic, D., and Skorin-Kapov, L., 2015. The impact of mobile device factors on QoE for multi-party video conferencing via WebRTC. In *Telecommunications (ConTEL), 2015 13th International Conference on* (pp. 1-8). IEEE.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P., 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, *13*(4), 600-612.

Liang, Y. and Collins A., 2016. *Selenium WebDriver. From Foundations to Framework*. Leanpub.