Quality-of-Experience driven configuration of WebRTC services through automated testing

Antonia Bertolino*, Antonello Calabró*, Guglielmo De Angelis†,
Francisco Gortázar‡, Francesca Lonetti*, Michel Maes‡, and Guiomar Tuñón§

*CNR–ISTI, Pisa, Italy
Email: {antonia.bertolino, antonello.calabro, francesca.lonetti}@isti.cnr.it

†CNR–IASI, Roma, Italy
Email: guglielmo.deangelis@iasi.cnr.it

‡Universidad Rey Juan Carlos, Madrid, Spain
Email: {francisco.gortazar, michel.maes}@urjc.es

§NAEVA TEC, Madrid, Spain
Email: gtunon@naevatec.com

***Abstract*—Quality of Experience (QoE) refers to the end users level of satisfaction with a real-time service, in particular in relation to its audio and video quality. Advances in WebRTC technology have favored the spread of multimedia services through use of any browser. Provision of adequate QoE in such services is of paramount importance. The assessment of QoE is costly and can be done only late in the service lifecycle. In this work we propose a simple approach for QoE-driven non-functional testing of WebRTC services that relies on the ElasTest open-source platform for end-to-end testing of large complex systems. We describe the ElasTest platform, the proposed approach and an experimental study. In this study, we compared qualitatively and quantitatively the effort required in the ElasTest supported scenario with respect to a "traditional" solution, showing great savings in terms of effort and time.**

***Index Terms*—Quality-of-Experience, Non-functional Testing, Configuration Testing, Real-time Services, WebRTC, ElasTest Platform**

I. Introduction

In last years the great advances of WebRTC technology is favoring more and more the use of web services as the preferred place for work meetings, social activities, and entertainment. In fact, WebRTC supports real-time communication across web browsers, thus making it possible the provision of services requiring voice and video communication from any web applications and mobile devices. In contemporary times, in which people from many countries remain locked at home to contrast the insurgence of Covid-19 disease, school lectures, seminars, working meetings as well as social activities continue to happen on-line largely thanks to WebRTC-centred browsers, a solution that could not have been conceived on such a scale a few years ago.

Although peer-to-peer communication is possible in WebRTC, usually some WebRTC server is involved for many reasons: many-to-many communications, recording, or transcoding, among others. Developers of WebRTC servers must ensure not only that the provided services comply with the functional specifications, but also that the end users will be satisfied with the (audio and video) quality of provided services. To this end, the notion of a user's Quality-of-Experience (QoE) has gained momentum. The QoE concept has been defined by the Qualinet consortium as *the degree of delight or annoyance of the user of an application or service.*[1] The definition continues by pointing out how such subjective concept depends on the user's expectations, also in relation with the service utility and the user's personality and state.

Research on how QoE should be modelled, measured and assessed has been very active. As reported in [2], methods have been proposed to evaluate QoE in subjective or objective way. The former are based on end-users evaluation, whereas the latter rely on some parameters that have been shown to correlate well with subjective assessments. In either case, the implicit idea is that the quality of the provided service is assessed as delivered.

However such approaches cannot help while the services are under development: during design and configuration the programmer has to draw decisions about how to structure and empower a server's architecture, and how to quantify the resources to be employed. Such type of decisions are aimed at providing satisfying QoE, but on the other side must also take into account the cost behind a certain configuration. This is a question that arises often with OpenVidu[1], an open source WebRTC project aimed at enabling WebRTC in any web and mobile application. Companies willing to use the project know the number of users they are going to have in WebRTC calls, and they are concerned with the computing resources needed and their cost.

This is the problem we address here. We have recently released the ElasTest open-source platform for automated testing of cloud services, also embedding WebRTC solutions. The platform has been developed within an H2020 European project, now terminated (Dec. 2019). At time of writing,

[1] see https://openvidu.io/

978-1-7281-8913-0/20/$31.00 ©2020 IEEE
DOI 10.1109/QRS51102.2020.00031

ElasTest 2.4.0 has been released. An extensive documentation and the software can be obtained from the project web page[2].

ElasTest supports both functional and non-functional end-to-end testing of large applications. In particular we present here how the platform can facilitate the QoE-driven evaluation and configuration of WebRTC services. By ElasTest, a tester can easily and quickly try different solutions so to identify the best trade-off between delivered QoE and employed resources. Moreover thanks to the online support provided by ElasTest, QoE assessment can be shifted left in the life cycle, thus supporting timely configuration and resource-related decisions. A small experimental study provides preliminary evidence of advantages in using ElasTest vs. a plain ad-hoc approach.

In summary, the contribution of this work includes:

- the ElasTest approach for QoE-driven non-functional testing: to the best of our knowledge this is the first automated approach that supports shift-left QoE assessment to drive WebRTC service configuration;
- an experimental study comparing qualitatively and quantitatively the effort required in the ElasTest supported scenario with respect to a "traditional" solution.

In the next section we present related work. Section III provides an overview of ElasTest functions and architecture, not only to make the paper self-contained, but also because we believe it can be of interest to QRS participants also beyond the specific usage scenario presented in this work. In Section IV we discuss the motivations behind the work, and in Section V we explain our approach. Then, in Section VI we present the study conducted. Finally, we draw conclusions in Section VII.

II. Related Work

The work presented in this paper spans over two main research directions that are: i) the broad area of test execution on large, complex, distributed and heterogeneous systems, with a specific focus on non-functional testing; ii) QoE testing of WebRTC applications.

a) Test execution on distributed systems: Testing research addresses many scientific and technical challenges that arise from the testing of complex distributed systems, and more specifically of cloud-based systems[3]. A recent systematic review [4] provides an overview on the current status of cloud testing. Among the main challenges resulted from the study, the need emerged of efficient approaches to support the execution of complex test scenarios where computational resources are elastically provided on-demand. Indeed, setting up the testing environment is often a time-consuming activity that implies high costs and complexity. In this sense, the need emerges [4] for an effort to link software testing research with the progress in cloud computing so to facilitate the management of the resources for a fast emulation of real-world scenarios, while granting an increased observability at testing-time.

ElasTest addresses several of the challenges emerged in [4]. Specifically, it supports end-to-end testing in different test scenarios and allows for, among other features, simulation of real-world conditions, analysis of test data as well as performance testing combined with elasticity aspects.

Similar to ElasTest platform, several cloud testing tools or infrastructures exist. For example, platforms such as Cloud Crawler [5] allow cloud testers to better control the costs of the cloud configurations and resources allocation; the Vee@Cloud architecture [6] enables the configuration of scalable testbeds by managing virtual resources from a Infrastructure as a Service (IaaS) provider.

The outcome of a technical analysis of the SotA [7] highlights the most important available tools and frameworks that are related to the ElasTest platform and shows the advancement of ElasTest with respect to these tools in many technological areas, including those that are related to the approach proposed in this paper. In detail, ElasTest provides a whole set of functionalities specifically conceived for managing the whole test process, reducing the complexity of the configuration of the Continuous-Integration (CI) servers. Also, the ElasTest platform provides a structured and complete monitoring sub-system that realises an observability layer at testing-time.

Concerning non functional testing, traditional relevant tools [3] such as Apache JMeter, Artillery.io, K6.io, WebLOAD, Load Impact are used to perform load testing and measure the performance of web applications. These tools focus on assessing the load capability but provide limited functionalities regarding the validation of the System Under Test (SUT) in different scenarios whereas ElasTest enables the combination of performance testing with scalability aspects, allowing testers to run different configurations and compare their results.

b) QoE Testing of WebRTC-based applications: QoE assessment of WebRTC applications is nowadays a challenging research topic aiming at identifying the many factors that might influence the user experience. In this context, the authors of [8] distinguish three main types of factors that influence the QoE for WebRTC video calls: human factors referring to human characteristic (e.g., age, gender, educational background, needs, previous experience); context factors describing the physical, temporal, social, economic, and technical user environment; and system factors defining the properties of the content, medium, device, network that determine the technical quality of the applications. Among context factors, task complexity and duration are considered important factors influencing the QoE assessment [9].

A recent systematic review on the assessment of QoE in WebRTC applications [2] presents an updated comprehensive summary of the aspects of Quality of Service (QoS) and QoE covered by the existing works in literature. An outcome of this review is that QoS parameters such as end-to-end delay, jitter, packet loss, bandwidth, throughput, and bit-rate are more used than QoE subjective methods (for instance the Mean

[2]see at https://elastest.io/

[3]see respectively: http://jmeter.apache.org/, https://artillery.io/,https://k6.io/, http://www.radview.com/, and https://loadimpact.com/

Opinion Score (MOS)) for assessing the quality experienced by end users. Moreover, the work in [2] defines a set of KPIs for modelling and assessment of QoE for WebRTC including, among the others, call establishment time, end-to-end delay, audio, video and audiovisual quality. The same authors released an open-source benchmark [10] for assessing WebRTC applications in which different objective metrics for video and audio can be correlated with a subjective analysis of the same recording by means of Mean Opinion Score (MOS).

In the last years, several works are focusing on platforms and tools for testing of WebRTC applications. According to the technical analysis of the SotA [7], most of them, such as TestRTC, Kurento Testing Framework (KTF) [11], [12] or callstats.io [4] aim to test, monitor and analyse WebRTC-based communications. They mainly focus on real-time reporting about the voice/video quality, by collecting and calculating many KPIs related to the voice and video streams. However, the support they provide to the testing of QoE is still limited. For instance, callstats.io provides an Extended Mean Opinion Score (eMOS) metrics to measure QoE for WebRTC voice and video, whereas KTF, developed within the Kurento open source project, allows to capture both QoS and QoE indicators by exercising the SUT in different network scenarios in order to simulate potential real world WebRTC issues at the development stage.

A recent state of the art of WebRTC testing presented by García et. al [12] presents a comprehensive summary of the trends in WebRTC testing, investigating: i) research papers of WebRTC testing, ii) WebRTC testing tools and iii) grey literature. Focusing on the tools, the authors find that the Selenium framework provides the main tools for WebRTC testing. The main problem of this framework is the need to have browsers available to use. Some PaaS [5] as Saucelabs, BrowserStack and Nightwatch.js offer commercial solutions to this problem. With respect to these tools, ElasTest provides enhanced WebRTC testing solutions by simulating different WebRTC network topologies and addressing interoperability issues for the SUT or any other device involved in the test. Leveraging ElasTest support, testers are able to plan and to emulate user actions in many heterogeneous GUIs (including web browsers) in the same or different machines, overcoming the limitations of WebRTC testing tools as Selenium. The resulting coordinated support from the underlying platform, allows testers to access integrated features such as the simultaneous display of the execution logs, synchronization between video recording and logs useful for ease detection of failures and management of different SUT environmental conditions (e.g. load, network conditions, failures, etc.).

The experimental study reported in this work shows how ElasTest can improve the QoE testing of WebRTC applications.

[4]see also respectively: https://testrtc.com/, and https://www.callstats.io/

[5]see respectively: https://saucelabs.com/, https://www.browserstack.com/, and https://nightwatchjs.org/

III. The ElasTest Platform

ElasTest [13] is an open source platform specifically conceived for managing the end-to-end testing activities for different kinds of distributed applications. Its main purpose is to support testers, and developers with dedicated feature for the assessment of a distributed SUT in a comprehensive and elastic platform [14]. In this sense, the use cases for ElasTest cover the whole spectrum of activities in the testing life-cycle: deploying and monitoring the SUT, executing the end-to-end tests, gathering logs and metrics from the execution and its elastic environment, exposing the results to software engineers and testers.

From a methodological perspective, planning testing campaigns with the help of ElasTest could led to the application of several testing principles: addressing atomic testing objectives by means of the definition of atomic testing bundles expressed in any language and using any testing framework (i.e., `TJobs`); enabling the sequential and parallel composition of available `TJobs` (i.e., test orchestration [15]); executing the SUT in operational conditions close to the actual real-world context; customising the SUT so to gather relevant information during testing; recommending those testing actions that enable interactive decision-taking during the testing phase.

From a technical perspective the ElasTest platform is conceived as a cloud-based architecture (see Figure 1) exposing its functionalities through REST APIs (both HTTP and WebSocket). Developers and testers can interact with a platform instance by means of web or command-line interfaces. Also, the chaining with CI servers can be realised by means of dedicated plugins (e.g., the ElasTest Jenkins plugin).

The architecture of ElasTest follows the microservices architectural style. It distinguishes between a set of mandatory services, and a set of other pluggable services that provide optional support features or domain-specific/legacy features. Among the mandatory services, the *ElasTest Platform Manager* (`EPM`), is responsible of isolating the ElasTest services from the underlying cloud infrastructure, enabling a seamless deployment and execution of the ElasTest features across a variety of target cloud infrastructures (e.g. Docker, Kubernetes, AWS, OpenStack, etc.).

The *ElasTest Orchestration and Recommendation Manager* (`TORM`) is the mandatory service acting as entry point to all the functionalities in ElasTest. Overall, the main use-case for `TORM` concerns the selection and the sequential/parallel activation of the `TJobs` while a testing session is running. Specifically, the orchestration concept refers to an on-line decision mechanism about which `TJobs` have to be launched next, which can be driven by the exit-status (i.e., *verdict-driven* activation), or by the results (i.e, *data-driven* activation) of the previous testing activities [15].

During the test execution the ElasTest platform handles several kinds of data: outcome of the tests launched, logs from the SUT, metrics about the resources consumed by the SUT, etc. All these data are gathered by managing specific kinds of data sources (e.g., `EMS`). The mandatory service that is

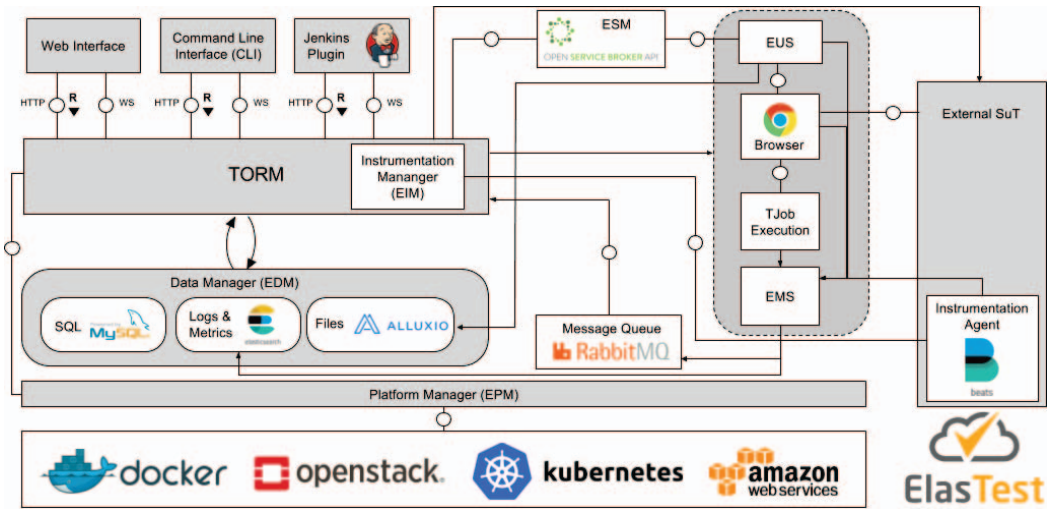

Fig. 1. Architecture of the ElasTest Platform

responsible for the persistence of the data collected during the testing session is the ElasTest Data Manager (EDM). The EDM functioning relies on different persistence paradigms: relational DBs (MySQL), document indexing aggregators (the ELK stack), or distributed storage system (Alluxio)[6].

Another important mandatory service is the *ElasTest Service Manager* (ESM) responsible for the discovery and the enactment of other optional service instances in the platform. For example, the ESM is responsible for the engagement of test support services specifically conceived in order to support the reuse testing harnesses offered by optional components of ElasTest. Among them, the ElasTest User Service (EUS) offers the impersonation of human actors interacting with the SUT by means of an actual instance of some web browser (e.g., Google Chrome). In this sense, EUS provides a mechanism for enabling the emulation of actual users in end-to-end tests of the SUT. From a technological perspective, the EUS is backwards compatible with existing frameworks for functional testing such as Selenium[7].

The EUS component is the one enabling WebRTC testing, lowering the entry barrier and the efforts needed. This component provides browsers as a service that, in collaboration with the EUS, are capable of retrieving and send to the TORM real-time statistics about WebRTC applications, and include setups specific for measuring QoE using different algorithms. When combining with services such as EMS, complex load scenarios in the WebRTC arena can be devised and implemented with a low effort in terms of both time and lines of code.

In broad terms, the ElasTest platform has been conceived so to target any kind of SUT: no specific constraints are imposed to the SUT or to its deployment. Nevertheless, ElasTest includes features for managing and controlling the deployment of SUT instances over dedicated cloud facilities. In this case, the ElasTest Instrumentation Manager (EIM), a dedicated component of the architecture, is responsible for governing potential realistic environmental conditions affecting the test execution. Specifically, EIM can instrument a SUT instance so to enable infrastructure-level failures such as network packet-loss, available bandwidth adjustments, CPU bursting, node failures, etc.

[6]see respectively: https://www.mysql.com/, https://www.elastic.co/, and https://www.alluxio.org/

[7]see at https://www.selenium.dev/

IV. Motivation

Testing is a best-effort activity: the more extensive the testing process, the higher the chances that the tested system will meet the expectations from its stakeholders. However the testing activities are usually quite costly both in terms of resources required and in terms of human effort needed for running and coordinating them. For this reason, the automation of software testing has become a key ingredient in order to grant the proper quality for the considered SUT.

In this section, we briefly introduce the overall context of the ElasTest validation, by which our study related to QoE assessment for WebRTC services is motivated.

A. Context

The ElasTest platform aims at improving the testing of complex distributed software systems; notoriously this is difficult and effort-prone domain. In this sense, several concerns could tackle the acceptance of the ElasTest platform by both testers and software companies. Among the others, its efficiency in the testing process if compared with the existing practices of combining other available testing frameworks.

The study reported in this work is part of a broader evaluation effort that accompanied the whole project life-cycle. Such effort aimed at continuously assessing the usefulness and effectiveness of ElasTest support for test automation and at providing evidence-backed feedback to project management between successive platform releases. Its main objective

was answering the following broad research questions: Can ElasTest contribute to improve the efficiency of the testing process? How? How much?

Among the several validation metrics that had been set since project inception (as described in the project deliverable D7.2 [16]) to address such questions, we also aimed at evaluating whether ElasTest could improve QoE testing.

For this aim, we set up a specific experimental scenario in the context of developing WebRTC applications, as described below.

B. Target Domain

The application domain subject of the case study concerns solutions for real-time communications (RTC) based on the WebRTC Technology. WebRTC[8] is a free, open standard that proposes reference specifications for a technological stack and shared APIs specifically conceived for enabling RTC capabilities among web and mobile applications. Different reference implementations of the WebRTC technology have been released for the most common web/mobile browsers as well as for native clients.

In other words, WebRTC is a modern cross-platform solution that aims to define an independent transmission level for media over the Internet. Indeed, WebRTC clients can be directly implemented within web pages by allowing direct peer-to-peer media communication, and eliminating the need to install plugins or download native apps.

Traditional testing for RTC applications focuses on validating if the system behaves as expected under an established set of defined scenarios (e.g. the transmitted packages are accepted/skipped according to the prescribed deadlines). Nevertheless, in many practical cases such tests do not validate if the experience of the participants during the communication sessions (i.e. the Quality of Experience – QoE) is satisfying. Among the main challenges for automation testing is that QoE also involves subjective judgements. To circumvent the challenges posed by testing with final users, QoE testing often refers to indirect objective parameters (e.g., transmission rate, delay, jitter, bit error rate, packet loss rate, etc.) in order to estimate the perceived QoE of WebRTC sessions. Specifically, each parameter is associated with a threshold denoting the criteria for a successful communication. However the measurement of these parameters are not easily accessible by automatic testing machinery, and it is nearly impossible to use them as oracles for judging the results of an acceptance testing level.

Moreover, the evaluation of end-users experience (both subjective and objective) can only take place after the service is released, which might be late in the lifecycle. Developers need to have a reliable prediction of QoE while a WebRTC service is being configured, so to be able to assess the resources to be employed to guarantee sufficient quality at proper costs. ElasTest supports developers of WebRTC services in this step, by allowing a quick and easy evaluation of different configurations that may impact on the audio and video quality of delivered solutions.

[8]see https://webrtc.org/

V. Testing for QoE Assessment

Assessment of WebRTC applications is a difficult task. A common question that arises when planning the infrastructure for a WebRTC application is, given a number of participants, what is the smallest server that will provide the desired quality. This question seeks to find a server size that fits the needs with the lowest cost.

As introduced before, quality in WebRTC applications is usually measured in terms of Quality of Experience, with either a subjective or objective approach. Obviously, the subjective approach is unfeasible in many situations. For a continuous testing approach, where tests are to be run on continuous integration systems, it is not practical to rely on human assessment of the quality of the videos produced. Hence, the objective approach seems a better approach, but it has two main problems: a) some methods are offline, and they require the test to finish before being able to provide the quality estimation; b) others require a reference video, i.e., the original video sent by a participant, that is then compared to the one received by another participant.

In addition to these limitations, there is nothing on the QoE algorithms that is aimed towards deciding about a feasible infrastructure for a given QoE threshold. Developers are responsible for preparing different scenarios, collecting the QoE measurements, and then use the data to drive a decision.

One of the contributions of this paper is proposing a quick way to assist decision-makers in choosing the best cost-effective solution that fits their needs in terms of number of users and QoE in the WebRTC arena. The method is not as accurate when estimating QoE as the above-mentioned algorithms, however, the development cost is much lower, as many tasks are automated within ElasTest.

The proposed method consists on: a) implementing a test that exercises a WebRTC application emulating a real scenario; b) defining different configurations for the test; c) collecting jitter and delay metrics from the different browsers involved in the communication, and use them to estimate QoE; d) plotting the results of the different scenarios in a graph for a better assessment of the cost-effective solution that complies with the desirec QoE.

How video-conference sessions are initiated, i.e, the video-conference software to use, or how many participants are involved in each session is up to the test. The rest of it is automated by ElasTest. Once the test is in place, it is executed against the configurations defined in step b). During each of these executions jitter and delay metrics are collected. As a result, it is possible to estimate the resources needed for a given WebRTC load, something that, to the best of our knowledge, is not possible with any other WebRTC testing tool.

VI. Experimental Study

The case study on which we evaluated the ElasTest support in assessing QoE refers to OpenVidu[9]: a specific video-conference system that is able to manage WebRTC sessions (e.g. enabling communication channels, or media transmission). OpenVidu is an open-source project that wraps and hides all the low-level operations foreseen by the WebRTC technology so to provide a simple, effective and easy-to-use API.

The proposed case study envisions the dimensioning of an online video-conferencing PAAS. On-demand, the service offers a WebRTC server that can host up to 5 independent video-conference sessions with 7 users each; all these sessions are supposed to run in a single OpenVidu instance. For the purpose of this study we assume that three different Virtual Machine (VM) configurations are available (i.e., t3.medium, t3.large, t3.xlarge[10]). The testers of the video-conferencing PAAS aim at identifying the best VM configuration that should be instantiated. Clearly, the more powerful a VM the higher its costs. This scenario aims at investigating the best trade-off between costs and potential QoE offered by the video-conferencing service. The objective of the testing campaign is to identify a VM configuration fitting with the purpose, which is as small as possible while still providing good video and audio quality.

A. Methodology

To evaluate the validity and efficiency of ElasTest support in QoE assessment, we conducted a controlled study. The hypothesis we test is that test automation can make early assessment of QoE more efficient. Although the size of the study we report here was quite small that calling it an "experiment" may be too pretentious, nevertheless we applied every care to make it in rigorous and unbiased manner.

Two practitioners from the consortium volunteered to perform the study. The two practitioners had similar coding expertise and testing skills.

In detail, they have been assigned the same task of exploring which VM configuration provides the best trade-off between costs and potential QoE offered. The cost was measured by the time required for testing, whereas the potential QoE was measured by delay and jitter [17].

They have been asked to complete the assigned task by using differing methodology. Precisely:

- one of them, denoted as the traditional tester (TT in the following), was asked to follow the testing practices, current tools, frameworks, and facilities in place at her company;
- the other one, denoted as the ElasTest tester (ET), had to conduct the same checks but managing the testing activities with the support of the ElasTest platform.

More specifically, both testers (TT and ET) have been informed about the objective of the testing campaign (see Sec. IV-B); the OpenVidu documentation and its testing harnesses have been provided, as well as the information about the target infrastructure for hosting the deployment of the testbed.

The assessment of quality during the WebRTC sessions is achieved by observing interactions among the OpenVidu instance and actual browsers from emulated clients deployed in the testbed. Each test estimated the QoE by relying on two metrics provided by each of the browsers: delay and jitter. If at any moment the delay or jitter exceeded a given threshold, the test failed so that the specific configuration of VM used for hosting OpenVidu should be discarded.

Each the testers applied the testing plan by taking notes of each configuration launched, the indirect objective parameters measured at the client-side (i.e., delay, jitter), their final evaluation about the result of each testing session, and the time they spent for each activity performed in order to setup and launch each test.

B. Set-up

Both testers had to set-up a testbed enabling an OpenVidu instance running on Amazon WebServices (AWS) resources. Specifically, they defined a test checking if the OpenVidu instance was able to serve 5 parallel video-conference sessions, each of them connecting 7 different users. Each WebRTC session lasted for 1 minute.

The test was run three times on three instances of the testbed, one per VM configuration:

- t3.medium: 2 vCPUs, 4Gb memory, up to 5Gb bandwidth.
- t3.large: 2 vCPUs, 8Gb memory, up to 5Gb bandwidth.
- t3.xlarge: 4 vCPUs, 16Gb memory, up to 5Gb bandwidth.

In detail, both testers referred three different deployments of the OpenVidu instance, each one corresponding to a VM configuration accessible by means of a given IP address.

Users in the video-conference sessions have been emulated using browsers running in AWS VMs, as well. Specifically both testers were provided with a VM image for AWS equipped with the Chrome Browser. The VM image was ready to be instantiated and to provide browsers emulation on-demand.

About the measurement fetched from the browsers: both the considered metrics has been sampled every second; for each metric the average across the samples from all the available browser instances is computed. The estimated QoE for a user (i.e., a browser instance) is considered too low if the reported average exceeds the reference thresholds: 150 msec for the delay, and 100 msec for the jitter [17][18]. In this case, the test is considered failed. It is important to note here that these metrics and averages are automatically retrieved and provided to the ET by ElasTest, but the TT will have to implement the retrieval of this information by hand.

In a first step, both TT and ET contributed in developing an abstract set of test cases for the considered scenario. Then, they had to individually adjust these implementations by

[9] see at https://openvidu.io/

[10] see at https://aws.amazon.com/ec2/instance-types/

adding adaptations taking into account each specific execution context.

In this sense, TT had to set-up all the required infrastructure and machinery necessary for enabling the test to work. For example the TT tester had to implement 2 jobs in Jenkins [11] in order to properly activate/deactivate the VMs required by the scenario. Also, the refinement of the tests required for the implementation of a software layer (i.e. ad-hoc infrastractural scripts) for aligning the averages form the fetched metrics, asserting the exit status of the test, and reporting the observed outcomes.

ET implemented the testbed by defining a configurable `TJob` were a collection of parameters were referred. Among the others, parametric data included the IP addresses of the three VMs running the SUT (i.e. referring the VM hosting the OpenVidu instance), and the security keys to be used during the interactions. All the required VMs emulating the active browsers were deployed and controlled by ElasTest. This way, ElasTest was able to reproduce the exact environmental conditions for three runs each time against a different SUT. Furthermore, its native dashboard for collecting and plotting information has been easily exploited in order to aggregate and compare the data notified by the managed VMs. Still ET had to implement the logic for asserting the exit status of the test, but the task was considerably simplified because of the shared time-stamping system impressed by the coordination of the activities by the ElasTest platform.

The artefacts developed by both TT and ET are available as replication package of the experimental study in: [19]

C. Results

The reported study aimed to assess if ElasTest could contribute to improve the efficiency of the WebRTC configuration by means of test process automation. In this sense, we observed the verdict and the testing time of two testers (i.e., TT and ET) assessing the best trade-off in terms of potential QoE and costs of the configuration for a given scenario.

As expected, the achieved results in terms of enabled users/sessions/vm-instance were similar for both TT, and ET. Both the testers tested the same system, thus the testing campaigns should observe similar outcomes for delay and jitter for each configuration. Indeed, the metrics should not be affected by the referred testing method.

Nevertheless by analysing the feedback from the testers, there was an evident difference in terms of effort and complexity in conducting the study, as we show in Figure 2. Notice that main differences come from the coding and setting-up, whereas test execution times remain similar. This comes as expected, because both testers were running tests that were performing the same actions. However, differences arise when we look into the developing of the tests and setting-up of the infrastructure in need for running them. TT who directly composed the features of JUnit, Selenium and Maven (plus their manual reviews) recorded that about 73 hours had been

[11] see at:https://jenkins.io/

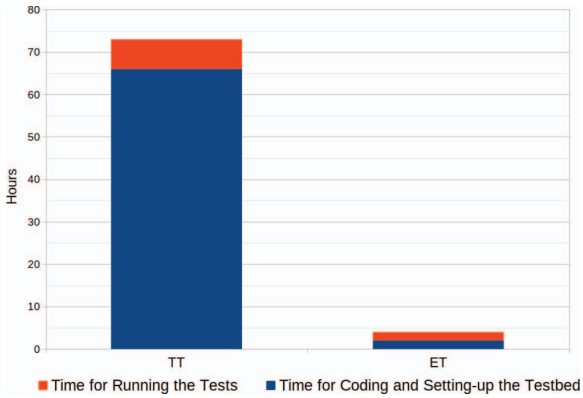

Fig. 2. Results of the experimental study about QoE

required in order to assess the QoE. Specifically, from the records it results that greatest part of their time (i.e., almost 66 hours) have been employed in refining the tests, coding the infastructural scripts, and properly setting up the testbed. Specifically, TT had to code the logic to instruct the browsers to send the jitter and delay, collect these metrics, align them and process them. After running the tests, TT also had to manually compare the results of each tests in terms of QoE to provide a final answer as to which VM was providing better trade-off between QoE and cost. On the other side, ET recorded that all the assessment of the QoE took almost 4 hours, where 2 hours were spent in order to adapt the abstract set of tests.

Such an impressive difference can be justified by considering two major factors: on the one hand the TT tester spent a lot of effort in preparing ad-hoc code able to specifically record the metrics of interest within the testbed and to redirect them to the specific log produced during the testing process; on the other hand logs produced at each run should have been aligned and processed in order to retrieve measured values that could be properly compared. In other words, the direct composition of available testing framework required to develop a specific system observability layer running at testing-time [20].

ET had this concept available within the ElasTest platform at no cost. In this way they leveraged it easing the understanding of the system as well as the detection of potential issues or undesired behaviour. Specifically in the experimental study the ET tester used the services from ElasTest in order to deploy, launch, and handle both the testbed and the test programs. In addition this tester could also access an analytics dashboard enabling a goal-driven inspection, visualisation, and comparison of both logs and metrics collected during the tests for the three VMs considered.

The factors were also confirmed by the qualitative feedback that has been collected from an ex-post meeting with the two testers. Specifically, the discussion led to identify those facilitators that contributed to the efficiency of ElasTest while assessing the QoE [16]. Among the others, the following major facilitators were identified:

- a lower number of test sessions to be executed, thanks to the multi-configuration browsers;
- a more comfortable way to save and inspect logs of non-functional data;
- the end-user service (EUS) from ElasTest emulates actions from a potential user within a browser, and reports all relevant information about the web session (including jitter/delay) without any extra coding;
- ET tester had more time for configuring a custom solution, tailored on the desired performance and quality, reducing the costs in instances;
- ET tester could more easily correlate the video with the data and the graphs that provide hints about what and where was the problem (in the browser, or in OpenVidu for example).

The above quantitative and qualitative results have of course to be considered in view of the many limitations and threats to validity of the study. Indeed, although we put in place any care to ensure a fair comparison, the experience is limited to one case study involving two testers only. The results achieved have to be taken as indicating a promising direction rather than an actual measurement of ElasTest performance.

VII. Conclusions and Future Research

Provisioning adequate QoE for WebRCT services is of paramount importance, but still nowadays its assessment remains a costly and complex activity. Existing solutions foresee either human evaluation or the ad-hoc composition of several frameworks, by aligning the results they provide. Easing how QoE of WebRTC application is assessed would clearly affect the way decisions about deployment and configurations are taken, but it could also impact the whole development process, and reduce the costs of the considered application.

In this work we present an integrated approach that could support software engineers in drawing decisions about configurations, deployments, and resources to be employed in developing WebRTC servers. The ElasTest approach leverages automated testing for the shift-left assessment of QoE on a collection of alternative configurations. A preliminary validation of the solution has been addressed by proposing an experimental study that compares both qualitatively and quantitatively the effort required while evaluating the QoE of WebRTC services with or without the ElasTest platform.

The results highlight that ElasTest improves the simplicity to set-up the testbed, to coordinate the execution of the modules of the SUT, to take and compare measures/logs collected during the execution of the test cases. Although the study is small and has no statistical significance, we believe that its results bring to the table a whole set of new possibilities for evaluating a SUT that relies on WebRTC communications driven by tested QoE. Automated testing of QoE configurations will bring the ability to reduce the time-to-market for WebRTC systems while increasing their overall quality. Indeed, engineering effort would be narrowed only to those scenario acceptable with respect to the constraining non-functional requirements.

Future research will target to replicate the experience reported in this paper, so to extend the assessment of the proposed approach in a broader context, and also in other domains.

VIII. Acknowledgements

This paper has been supported by the the European Project H2020 731535: ElasTest, and partially by the Italian MIUR PRIN 2017 Project: SISMA (Contract 201752ENYB).

References

[1] K. Brunnström, S. A. Beker, K. De Moor, A. Dooms, S. Egger, M.-N. Garcia, T. Hossfeld, S. Jumisko-Pyykkö *et al.*, "Qualinet white paper on definitions of quality of experience," https://hal.archives-ouvertes.fr/hal-00977812/document, 2013 (accessed 13/03/2020).

[2] B. García, M. Gallego, F. Gortázar, and A. Bertolino, "Understanding and estimating quality of experience in WebRTC applications," *Computing*, vol. 101, no. 11, pp. 1585–1607, 2019.

[3] A. Bertolino, G. De Angelis, and F. Lonetti, "Governing regression testing in systems of systems," in *Int. Work. on Governing Adaptive and Unplanned Systems of Systems*. IEEE, 2019, pp. 144–148.

[4] A. Bertolino, G. De Angelis, M. Gallego, B. García, F. Gortázar, F. Lonetti, and E. Marchetti, "A systematic review on cloud testing," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 93:1–93:42, 2019.

[5] M. Cunha, N. Mendonça, and A. Sampaio, "Cloud crawler: a declarative performance evaluation environment for infrastructure-as-a-service clouds," *Concurrency & Computation: Practice and Experience*, vol. 29, no. 1, 2017.

[6] X. Bai, M. Li, X. Huang, W. T. Tsai, and J. Gao, "Vee@cloud: The virtual test lab on the cloud," in *Proc. of 8th Int. Work. on Automation of Software Test*, 2013, pp. 15–18.

[7] F. Lonetti, Ed., *SotA Revision Document – Ver.2*. The ElasTest Consortium, 2019, no. Del. D2.4.

[8] J. B. Husić, S. Barakovic, and A. Veispahic, "What factors influence the quality of experience for WebRTC video calls?" in *Proc. of 40th Int. Conv. MIPRO*, May 2017, pp. 428–433.

[9] J. B. Husić, E. Alagić, S. Baraković, and M. Mrkaja, "The Influence of Task Complexity and Duration when Testing QoE in WebRTC," in *Proc. of 18th Int. Symp. INFOTEH-JAHORINA*. IEEE, 2019, pp. 1–6.

[10] B. García, F. Gortázar, M. Gallego, and A. Hines, "Assessment of QoE for Video and Audio in WebRTC Applications Using Full-Reference Models," *Electronics*, vol. 9, no. 3, p. 462, 2020.

[11] B. García, L. López-Fernández, M. Gallego, and F. Gortázar, "Testing framework for WebRTC services," in *Proc. of the 9th EAI International Conference on Mobile Multimedia Communications*, 2016, pp. 40–47.

[12] B. Garcia, F. Gortazar, L. Lopez-Fernandez, M. Gallego, and M. Paris, "WebRTC testing: challenges and practical solutions," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 36–42, 2017.

[13] A. Bertolino, A. Calabrò, G. De Angelis, M. Gallego, B. García, and F. Gortázar, "When the testing gets tough, the tough get elastest," in *Proc. of the 40th International Conference on Software Engineering: Companion, (ICSE 2018)*. ACM, Jun. 2018, pp. 17–20.

[14] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *Proc. of the 10th International Conference on Autonomic Computing*, 2013, pp. 23–27.

[15] B. García, F. Lonetti, M. Gallego, B. Miranda, E. Jiménez, G. De Angelis, C. Moreira, and E. Marchetti, "A proposal to orchestrate test cases," in *Proc. of the 11th Int. Conf. on the Quality of Information and Communications Technology*. IEEE-CS, Sept. 2018, pp. 38–46.

[16] A. Bertolino and E. Marchetti, Eds., *ElasTest Validation Methodology and its Results – Ver.2*. The ElasTest Consortium, 2019, no. Del. D7.2.

[17] M. Baldi and Y. Ofek, "End-to-end delay analysis of videoconferencing over packet-switched networks," *IEEE/ACM Transactions On Networking*, vol. 8, no. 4, pp. 479–492, 2000.

[18] G. Karlsson, "Asynchronous transfer of video," *IEEE Communications magazine*, vol. 34, no. 8, pp. 118–126, 1996.

[19] A. Calabró and G. Tuñón, "Replication package for ElasTest QoE experiment," Apr. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3739091

[20] F. Gortázar, "Observability in testing," 2018, QA3C Workshop: Invited Talk.