Testing Framework for WebRTC Services

Boni García
Universidad Rey Juan Carlos
Camino del Molino S/N, 28943, Spain
boni.garcia@urjc.es

Luis López-Fernández
Universidad Rey Juan Carlos
Camino del Molino S/N, 28943, Spain
luis.lopez@urjc.es

Micael Gallego
Universidad Rey Juan Carlos
Camino del Molino S/N, 28943, Spain
micael.gallego@urjc.es

Francisco Gortázar
Universidad Rey Juan Carlos
Camino del Molino S/N, 28943, Spain
francisco.gortazar@urjc.es

ABSTRACT
WebRTC is the umbrella term for several emergent technologies aimed to exchange real-time media in the Web. WebRTC is gaining the attention of practitioners quickly, and therefore the mechanisms to provide quality assurance for WebRTC services are becoming more and more demanded. WebRTC has been conceived as a peer-to-peer architecture where browsers can directly communicate. This model can be extended using a media server to provide extra features such as group communications, media recording, and so on. In this context, the open source initiative kurento.org provides a WebRTC media server and a set of APIs aimed to simplify the development of advanced WebRTC applications. Among these APIs, Kurento provides a high level testing infrastructure to assess WebRTC services in terms of functionality, performance, and quality-of-experience. This paper presents a detailed description of the testing services provided by this framework.

CCS Concepts
• Software and its engineering→Software creation and management→Software defect analysis→Software testing and debugging.

Keywords
WebRTC; Software Testing; Performance; Quality-of-Experience.

1. INTRODUCTION
Web Real-Time Communications (WebRTC) is the set of emergent technologies and APIs that aim to bring such communications to the Web. The standardization activity for WebRTC is split 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 (Application Programming Interfaces) and the standard HTML5 tags to enable real-time media capabilities to browsers. On the other hand, IETF is defining the underlying communication protocols (SRTP, SDP, ICE, and so on) for the setup and management of a reliable communication channel between browsers [1]. WebRTC is a technological initiative getting considerable worldwide attention nowadays. Therefore, the need for evaluating the quality of WebRTC systems and services becomes urgent.

WebRTC-based applications and services enable human-to-human communication. These systems can be evaluated with respect to their multimedia conversation quality, i.e. whether they enable a good communication of information between the involved peers.

This piece of research presents a testing framework aimed to simplify the testing process of WebRTC services. This framework has been developed within the open source project Kurento [2], and its name is Kurento Testing Framework (KTF).

The remainder of this paper is structured as follows. Section 2 summarizes the state of the art in two main topics of this paper: testing of web applications and quality-of-experience for multimedia. Section 3 presents some approaches for testing WebRTC available in the literature. Section 4 details the developed testing framework for WebRTC services. Section 5 provides implementation details and examples of test cases and results obtained with the proposed testing framework. Section 6 presents a case study in which the testing framework presented in this paper is used to carry out a case study aimed to select the features of an Amazon cloud instance hosting a given WebRTC service. Finally the conclusions and the future work is given on section 7.

2. BACKGROUND
2.1 Web Testing
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 [3].

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 [4].

In order to perform a complete assessment procedure, it is required to evaluate web applications from functional and non-functional perspectives. Functional requirements are actions that a software product must do to be useful to users. These requirements arise from the work that stakeholders need to do. Non-functional requirements (also known as quality attributes) are properties that the product must have. In other words, the functional requirements define what the system should do while non-functional requirements define how the system should behave [5]. According to Di Lucca and Fasolino, the most important non-functional requirements are performance, load, stress, compatibility, accessibility, usability, and security [6].

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

MOBIMEDIA 2016, June 18-19, Xi'an, People's Republic of China

DOI 10.4108/eai.18-6-2016.2264212

2.2 Quality-of-Experience for Multimedia

There is no agreement of what software quality actually means. The quality definition has been improved in 2005 at ISO 9000 standard, stating that quality is the "*degree to which a set of inherent characteristics [...] fulfils requirements*" [7].

In parallel to the re-consideration of the term "quality", the term "Quality of Experience" (QoE) has gained momentum, mainly with respect to media transmission systems and services. This term was born to counter-balance the term Quality of Service (QoS) with something which addresses the user's perceptions and experiences, because those were considered to be more appropriate for designing systems and services with a high acceptance [8].

The most widely way to classify QoE metrics is based on subjective or objective methods. 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 [9].

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 to estimate the mean opinion score (MOS score) [10]. MOS score was first standardized by the International Telecommunication Union (ITU). With this metric, the system quality is assessed subjectively from the mean opinion on a five-point scale: 1 = bad, 2 = poor, 3 = fair, 4 = good and 5 = excellent. The MOS is used typically in subjective tests of audio listening.

The objective quality measurement methods can be classified into the following five main categories (see Figure 1) [11][12]:

- Media-layer models: the QoE is calculated using the speech and video signal. This model does not require any information about the system under testing (e.g. codec type, packet-loss rate).
- Parametric packet-layer models: these models predict the QoE from the packet header information and do not analyze media signals.
- Parametric planning models: these models use quality planning parameters for networks and terminals to predict the QoE.
- Bit-stream-layer models: in these models, encoded bit-stream information and packet-layer information are used to measure QoE.
- Hybrid models: these models are a combination of previously mentioned technologies.

The media-layer objective quality measurement methods are subdivided into three types:

- Full Reference (FR): the degraded signal is compared pixel by pixel with the original signal.
- No Reference (NR): stream analysis on receipt without comparing it to the original signal. Quality information is extracted from the degraded signal, as no reference is available. These methods are very small, making them very suitable for analysis in real time.
- Reduced Reference (RR): these methods actually build upon representative parameters (typically statistical values) that allow for the quantification of the change of quality between the original and the distorted version.

The FR and RR are classified into three categories:

- Traditional point-based metrics. For example peak signal-to-noise ratio (PSNR). PSNR [13] is the proportion between the maximum signal power and the corruption noise power. This metric has several limitations and it can only be used as a measure of quality of reconstruction of loss compression codecs (e.g. for image compression).
- Natural visual characteristics oriented metrics. For example structural similarity (SSIM), video quality metric (VQM). SSIM [14] is a metric used for measuring the similarity between two images. The image quality measurement is based on a distortion free image as a reference. VQM [15] has been developed by the Institute for Telecommunication Science (ITS) to provide an objective measurement for perceived video quality.
- Perceptual oriented metrics. For example perceptual evaluation of speech quality (PESQ), perceptual evaluation of video quality (PEVQ). PESQ [16][17] is a method for evaluating speech quality autonomously as the experience of a telephony system user. PESQ is superseded by a listening-only signal based model called Perceptual Objective Listening Quality Assessment (POLQA) [18]. PEVQ [19] provides MOS scores of the video quality for IPTV, streaming video, mobile TV and video telephony. This method analyzes the degraded video signal through the network for evaluating the degradation.

3. RELATED WORK

WebRTC is an emerging technology still in development. Regarding testing and quality of WebRTC applications, there are not many references yet. This section provides a brief summary of the research literature available at writing time.

For instance, Cinar and Melvin use a black-box testing technique to evaluate, via PESQ, the voice quality of WebRTC sessions under varying network delay and jitter [20]. 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.

Amirante et al. presents a study of the performance of an open source WebRTC media server called Janus [21]. This work focuses on assessing the scalability of the media server architecture, by selecting some representative use cases, followed by an analysis of a multi-point audio conferencing scenario.

Vucic and Skorin-Kapov 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 [22].

WebRTCBench is a benchmark tool presented in [23]. This aim of this tool is to measure WebRTC peer connection establishment and communication performance. The authors of this work present and discuss performance evaluation of WebRTC implementations across a range of implementations and devices.

Finally, Singh et al. evaluate the performance of the congestion control Receive-side Real-Time Congestion Control (RRTCC), currently implemented in Google Chrome [24].

4. WEBRTC TESTING FRAMEWORK

WebRTC is a technology that provides Real-Time Communications capabilities to web browsers through JavaScript APIs. It is designed as a peer to peer architecture where browsers can communicate directly without the mediation of any infrastructure. This model is sufficient for creating basic applications, but features such as group communications, streaming media recording, media transcoding and broadcast media are difficult to implement on top of it. For this reason, many applications require the use of a media server.

Kurento[3] is a WebRTC media server and a set of client APIs making simple the development of advanced video applications for the Web and smartphone platforms. Kurento is a Free Open Source Software (FOSS) project (LGPL license). As a differential feature, the Kurento Media Server (KMS) provides advanced media processing capabilities involving computer vision, video indexing, augmented reality and speech analysis. Kurento modular architecture makes simple the integration of third party media processing algorithms (i.e. speech recognition, sentiment analysis, face recognition, etc.), which can be transparently used by application developers as the rest of Kurento built-in features.

Kurento also provides a complete testing framework (KTF, Kurento Testing Framework) aimed to simplify the assessment of WebRTC-based applications. KTF is not only for Kurento applications and can be used in general for WebRTC. 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, Internet Explorer, and so on). The well-known open source testing framework Selenium[4] is capable of handling this task. Selenium is composed by three components:

- Selenium IDE: It is a Firefox plugin implementing the record and playback pattern for web applications.
- Selenium WebDriver. It allows to handle local browsers natively as a user would using different programming languages (e.g. Java, C#, Python, Ruby, PHP, Perl, or JavaScript). It can be seen as the evolution of the now deprecated project Selenium RC (Remote Control).
- Selenium Grid: It allows distributing browser execution on parallel remote machines.

KTF provides a high level framework to perform complete automated testing for WebRTC-based applications. In the core of this framework, Selenium is a key component since it allows to handle browsers from the test logic. KTF supports three kind of browsers (scope):

- Local browsers. The host running tests should have installed web browsers in the operating system.
- Remote browsers. The execution of a test can be configured to run in a remote browser. These tests are implemented using Selenium Grid.
- Remote browsers from Saucelabs[5]. Saucelabs is a PaaS (Platform as a Service) cloud solution to support remote testing based on Selenium. It provides a set of cloud instances able to run WebDriver tests. At writing time, Saucelabs supports 700 combinations of platform (e.g. Linux, Windows, Mac OS X, Android, iOS), browser (Chrome, Firefox, Internet Explorer, etc), and browser version.
- Docker[6] browsers. Docker automates the deployment of applications inside software containers. It provides a layer of abstraction of the operating system virtualization on Linux.

4.1 Test Scenario

The first high-level concept introduced by KTF is the **test scenario**. Test scenario can be seen as the client-side infrastructure running a test case or suite. In other words, it is the set of browsers which are going to use the web application under test.

Let's suppose one of the simplest WebRTC application possible, in which two peers establish a real time communication through the Web. A textual description of a test scenario example would be as follows: "peer one uses Chrome and peer two uses Firefox". Another test scenario could be "both peers first use Chrome, and then use Firefox".

KTF allows to setup different test scenarios based on a custom and highly customizable JSON notation. In these JSON files, several test executions can be setup. For each execution, the browser scope can be chosen, i.e. local browsers, remote browsers, or browsers provided by Saucelabs.

For example, Listing 1 shows a test scenario in which two executions are defined. First execution defines one local browser (identified as *peer1*) and another in Docker (*peer2*), Chrome and Firefox respectively. The second execution defines also two browsers, this time browsers are located in the cloud infrastructure provided by Saucelabs (Internet Explorer 11 on Windows 8.1 and Safari 8 on Mac OS X Yosemite).

Listing 1. Example of test scenario in JSON notation.

```
{
   "executions" : [
      {
         "peer1" : {
            "scope" : "local",
            "browser" : "chrome"
         },
         "peer2" : {
            "scope" : "docker",
            "browser" : "firefox"
         }
      },
      {
         "peer1" : {
            "scope" : "saucelabs",
            "browser" : "explorer",
            "version" : "11",
            "platform" : "win8_1"
         },
         "peer2" : {
            "scope" : "saucelabs",
            "browser" : "safari",
            "version" : "36",
            "platform" : "yosemite"
         }
      }
   ]
}
```

Thus, KTF allows in a simple fashion to setup rich test scenarios to perform WebRTC testing using real browsers. But this is not

[3] https://www.kurento.org/
[4] http://www.seleniumhq.org/
[5] https://saucelabs.com/
[6] https://www.docker.com/

enough to perform a complete verification of a WebRTC application. Testing activities are aimed to provide answers about a System Under Test (SUT) to developers and testers [25]. Regarding WebRTC, many questions should be addressed before releasing an application to a production environment. For example: "Are WebRTC peers always receiving media?", "Is my application compatible with different versions of web browsers?" "What happens to my system when the number of WebRTC peers increase?", "Is the quality of the received media perceptible for users?".

In order to answer these questions (in other words, in order to perform different kind of tests), KTF provides further high-levels mechanisms to perform complex assessment activities easily.

4.2 Testing Activities

As introduced before, the aim of web applications testing consists of executing a web application under test using combinations of input in order to ensure quality and reveal failures. KTF provides specific capabilities to perform:

- Functional test. Assessment for WebRTC media capabilities.
- Performance tests. Evaluation of system behavior whilst web application is exercised with many concurrent requests.
- Quality-of-experience tests. These kinds of tests assess the quality of the media received in the browsers using QoE methods as depicted on section 2.2.

Regarding **functional** tests, two out-of-the-box capabilities are provided by KTF. WebRTC media is played in browsers using the HTML5 video element. Several media events are triggered by the video tag, for instance, play, pause, seek, change of volume, mute, or change of the playback rate. KTF allows subscribing to any of these video tags events, asserting if the subscribed event happens in a given time. In addition, the playing time is also captured by KTF. Therefore tests can easily assess whether or not the playback time is as supposed.

Another functional feature provided by KTF is the capability of analyzing the color of the received media in HTML5 video tags. Given the Cartesian coordinates within the video tag, KTF is able to get the RGB (Red, Green, Blue) color components. Then, a built-in comparison method based on the calculation of the RGB distance from the expected color and the real color can be used to assess whether or not the received color is similar to the expected one. In other words, the distance (d) calculated using the following equation should be less than a given threshold (th).

$$d = \sqrt{(R_{real} - R_{expected})^2 + (G_{real} - G_{expected})^2 + (B_{real} - B_{expected})^2} < th$$

KTF provides a fluent API to use these testing capabilities. See section 5 for further details an examples.

Regarding **performance** tests, KTF provides two main capabilities. First, a built-in function to handle the rate of the browser interactions with the SUT in form of a ramp (see Figure 2). Given a test scenario composed by N browsers, each browser interacts with the SUT one by one at a rate given by a customizable parameter (*parallel.browsers.rate*). The simultaneous execution of all browsers is maintained a given number of seconds given by another parameter (*parallel.browsers.holdtime*).

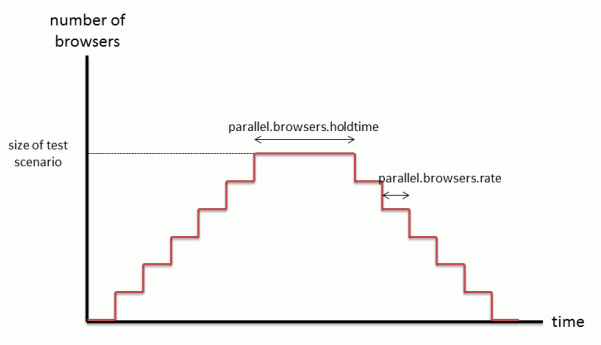

Figure 1. Browser ramp.

Let's see an example of a JSON test scenario for this kind of tests. A typical WebRTC application could be a video room application in which a peer (called presenter) sends its media stream to N peers (called viewers). In order to test this kind of application using the performance ramp described before, we first need to define the test scenario. Listing 2 shows how to define the test scenario for this video room application (with 80 remote viewers).

Listing 2. Example of scenario for performance tests.

```
{
   "executions" : [
      {
         "presenter" : {
            "viewer" : "local",
            "browser" : "chrome"
         },
         "viewer" : {
            "scope" : "remote",
            "browser" : "firefox",
            "instances" : 80
         }
      }
   ]
}
```

KTF provides the capability of monitoring a given machine, typically the sever hosting the SUT. The parameters gathered by the monitor are the following:

- Time (relative to the start of the test).
- Number of incoming clients.
- CPU usage (percentage).
- Memory usage (number of bytes and).
- Swap memory usage (number of bytes and percentage).
- Network interfaces usage (number of sent and received bytes in each of the network interfaces)

This information can be gathered locally, i.e. the SUT is running in the same host that it is executing the performance test or remotely, i.e. the SUT is running in a different host.

In addition, the system monitor also gathers client-side latency measurement. KTF implements a novel procedure to calculate the end to end latency in the WebRTC media transmission. This method is based on color detection of the local (sent by a peer) and remote (received by another peer).

To implement this method, first of all we have to detect the color change in the local and remote stream. Figure 2 shows an example of a video[7] with the required features. This video has a duration of

[7] http://files.kurento.org/

15 seconds. The first 5 seconds in the upper-left part a completely red video is shown. After that, another 5 seconds green video is shown. Finally another 5 seconds blue video is sent. This video is sent in loopback with the source peer in the WebRTC communication. The peer(s) consuming that media receives this video with a given latency.

As depicted before, KTF is able to recognize colors in the WebRTC streams. Thus, the color change is detected simultaneously in the local and remote stream. After a synchronization step, KTF is able to calculate the network latency each 5 seconds simply calculating the color change time from the remote and local stream.

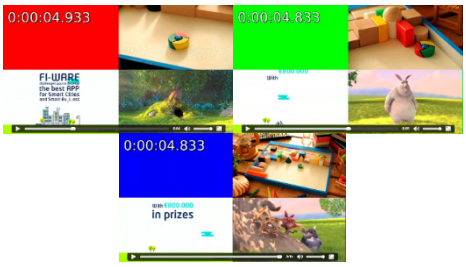

Figure 2. Example of video test used to measure latencies.

Regarding **quality** tests, currently PESQ (Perceptual Evaluation of Speech Quality) is supported to evaluate the received audio quality. To achieve this, first the PESQ algorithms needs to compare an original audio source to a degraded version. For this reason, in the JSON test scenario description, a new field called audio can be added. This parameter points to a WAV file that will be merged in the WebRTC peer acting as source of the communication by means of the Web Audio API[8]. In the WebRTC peer consuming the media, the audio is recorded by means of FFMPEG[9]. Finally, the quality test infrastructure executes PESQ with the source and the received audio file. The result is a MOS score from 1 to 5 as described on section 2.2. This figure can be used to assert when a test fails when the resulting MOS is lower than a given results (e.g. when resulting MOS is 3 or less).

4.3 WebRTC Statistics

WebRTC streams (audio, video, or data) can be lost, and experience varying amounts of network delay. In order to assess the performance of WebRTC applications, it could be required to be able to monitor the WebRTC features of the underlying network and media pipeline.

To that aim, KTC provides WebRTC statistics gathering for the server-side (KMS) and also for the client-side (*peerConnection*). The implementation of this capability follows the guidelines provided in the W3C WebRTC's Statistics API[10].

WebRTC statistics are read as a map. Each entry of this collection has a key and a value, in which the key is the specific statistic, with a given value at the reading time. The most relevant statistics are listed below:

- *ssrc*: The synchronized source (SSRC).
- *firCount*: Count the total number of full intra request (FIR) packets received by the sender. This metric is only valid for video and is sent by receiver.
- *pliCount*: Count the total number of packet loss indication (PLI) packets received by the sender and is sent by receiver.
- *nackCount*: Count the total number of negative acknowledgement (NACK) packets received by the sender and is sent by receiver.
- *sliCount*: Count the total number of Slice Loss Indication (SLI) packets received by the sender. This metric is only valid for video and is sent by receiver.
- *remb*: The Receiver Estimated Maximum Bitrate (REMB). This metric is only valid for video.
- *packetsLost*: Total number of RTP packets lost for this SSRC.
- *bytesReceived*: Total number of bytes received for this SSRC.
- *jitter*: Packet Jitter measured in seconds for this SSRC.
- *bytesSent*: Total number of bytes sent for this SSRC.
- *targetBitrate*: Presently configured bitrate target of this SSRC, in bits per second.
- *roundTripTime*: Estimated round trip time.

5. KURENTO TESTING FRAMEWORK IN ACTION

KTF has been implemented in Java. Moreover, KTF has been designed to use JUnit framework to implement tests. KTF dependency can be imported using Maven by means of the project coordinates described on Listing 3.

Listing 3. Kurento Testing Framework Maven Dependency.

```
<dependency>
  <groupId>org.kurento</groupId>
  <artifactId>kurento-test</artifactId>
   <version>6.4.0</version>
   <scope>test</scope>
</dependency>
```

In order to create new JUnit tests, these tests should extend the parent test class KurentoTest (located in the package `org.kurento.test.base.KurentoTest`). Listing 4 shows an example of JUnit Functional test. This listing provides important details to understand how tests which uses KTF works. These tests should use the parameterized feature of JUnit. This parameter accepts the test scenario which is created from the JSON file (in the snippet below this file is called *browsers.json*). This JSON file follows the notation depicted in section 4.2 and should be available in the project classpath.

Listing 4. JUnit Test Example.

```
import org.junit.Test;
import org.junit.runners.Parameterized.Parameters;
import org.kurento.test.base.KurentoTest;
import org.kurento.test.config.TestScenario;

public class MyTest extends KurentoTest {

   @Parameters(name = "{index}: {0}")
   public static Collection<Object[]> data() {
      return TestScenario.json("browsers.json");

   @Test
   public void test() {
      // Test logic
   }
}
```

[8] https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html
[9] https://www.ffmpeg.org/
[10] http://www.w3.org/TR/webrtc-stats/

5.1 Functional Tests

Let's suppose the JSON file with the test scenario description is the same as the depicted on Listing 1. In this case, in the test logic implementation, the browsers can be accessed by means of its identifiers, i.e. *peer1* and *peer2*, as illustrated in Listing 5. This snippet shows an example of subscription and assertion to the *playing* media event in the HTML5 video tag with id *video* in browser peer1. Then, for the same video tag, the media in the position x=0, y=0 is read and another assertion is made. Finally the current time for that video tag is also read.

Listing 5. KTF Functional Snippets.

```
// Media events
getBrowser("peer1").getVideoTag("video").
    subscribeEvents("playing");
boolean playing = getBrowser("peer1").
    getVideoTag("video").waitForEvent("playing");
Assert.assertTrue(playing);

// Color
Color realColor =
getBrowser("peer1").getVideoTag("video").
    getColorAt(0,0);
Assert.assertTrue(similarColor(realColor,
    expectedColor));

// Time
double currentTime = getBrowser("peer1").
    getVideoTag("video").getTime();
Assert.assertTrue(compareTime(currentTime,
    expectedTime));
```

5.2 Performance Tests

KTF provides an API to perform the browser ramp as depicted in Figure 1. Listing 6 shows an example of this method, used in a test scenario as defined in Listing 2 (1 presenter and 80 viewers).

Listing 6. Usage of KTF API to Carry Out Performance Testing.

```
private SystemMonitorManager monitor;

@Before
public void setup() {
   String host = "127.0.0.1, login = "user",
        key = "/path/to/key.pem";
   monitor = new SystemMonitorManager(host,
       login, key);
   monitor.start();
}

@After
public void teardown() {
   monitor.stop();
   monitor.writeResults("results.csv");
   monitor.destroy();
}

@Test
public void test() {
   Map<String, BrowserClient> browsers =
      getTestScenario().getBrowserMap("viewer");

   // Test logic for presenter

   ParallelBrowsers.ramp(browsers, monitor, new
      BrowserRunner() {
           public void run(BrowserClient browser)
              throws Exception {
           // Test logic for viewers
         }
      });
}
```

The class `SystemMonitorManager` performs the analysis on a host (typically the machine hosting the SUT). The results is tabulated in a comma-separated values (CSV) file, useful to get charts of the system performance with respect to the time or number of concurrent users.

5.3 Quality Tests

Snippet shown in Listing 7 shows the usage of the API provided by KTC to carry out QoE evaluation based on PESQ. When a custom audio file is specified in the test scenario, automatically KTF records the audio received by WebRTC. Then, using the class `Recorder`, the test logic is able to compare the recording with the original source by means of an integrated implementation of the PESQ algorithm. As a result, a MOS score is obtained, which can be used to feed a test assertion.

Listing 7. Usage of KTF API to Perform Quality Testing.

```
int sampleRate = 16000; // samples per second
float minPesqMos = 3; // PESQ MOS [1..5]
String audioUrl = "http://path-to/audio-test.wav";

float realPesqMos = Recorder.getPesqMos(audioUrl,
sampleRate);
Assert.assertTrue(realPesqMos >= minPesqMos);
```

6. CASE STUDY

This section provides a case study in which KTF is used as a benchmarking tool. Our service under test is basically a videoconferencing room application, in which a WebRTC peer acts as a presenter, and N WebRTC peers (let's call them viewers) receive media from the presenter. In order to implement this service, a WebRTC server infrastructure is needed. We use Kurento Media Server for the implementation of the server-side login of this application. Finally, we have established two non-functional requirements for our service:

- Performance: maximum number of concurrent viewers is 50.
- End-to-end latency: the maximum latency for the viewers for our service should be around 200 milliseconds.

We are going to install our media server in a cloud infrastructure (IaaS, Infrastructure as a Service), concretely the Amazon Web Services[11] (AWS). The question driving this case study is the following: What are the features of the AWS instance needed to meet the requirements of our service?

We use KTC to find a proper answer to this question. First, we use the ramp function as depicted in Figure 1 to model how the viewers are connected to the presenter in our real-time videoconference service. Thus, the features of this ramp of clients are the following:

- Number of clients: 50 clients. This figure determines the size of the test scenario.
- Rate of incoming clients: 1 second. This means that a new viewer (WebRTC peer) is connected to the presenter each second.

[11] https://aws.amazon.com/

- Simultaneous exaction time: 30 seconds. This means that the total amount of viewers (50) are connected to the presenter during 30 seconds.

Another important feature provided by KTC that is going to allow us to carry out this case study is the system monitor. We need to analyze the behavior of our WebRTC server in terms of CPU usage and end-to-end latency. Both figures can be obtained in the CSV report generated by KTF. All in all, we carry out our benchmarking tests against two types of AWS instances:

- M3-medium: VCPU 1 (Intel Xeon E5-2670 v2), RAM 3.75 GB.
- M3-large: VCPU 2 (Intel Xeon E5-2670 v2), RAM 7.5 GB.

After executing our tests, KTC generates a set of CSV files containing the monitor results. First of all, we compare the CPU usage in a M3-medium against a M3-large AWS instance. These results are shown in Figures 3 and 4:

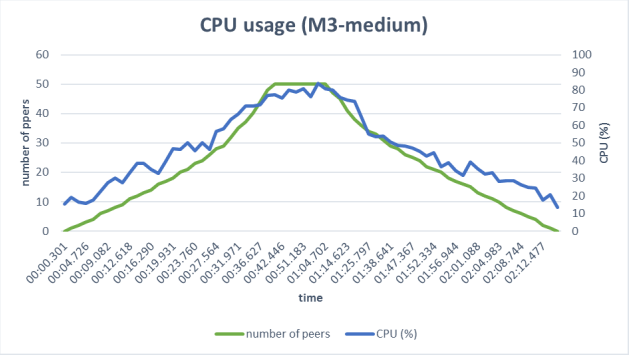

Figure 3. CPU usage in a M3-medium AWS instance.

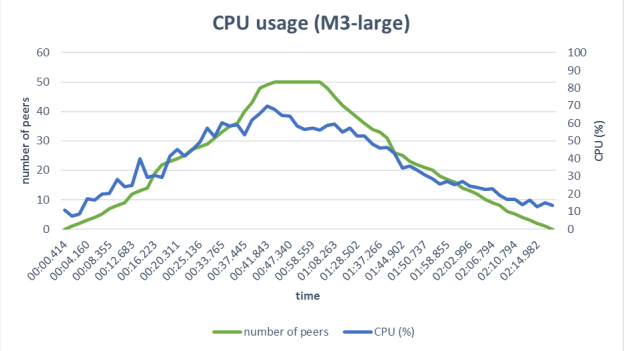

Figure 4. CPU usage in a M3-large AWS instance.

These diagrams shows the evolution of the number of viewers (WebRTC peers) connected to the presenter. The CPU usage is also displayed in both figures. In M3-medium the CPU percentage follows a distribution between the 10 to 50 percent approximately, whereas in the M3-large the distribution is around 10 to 70 percent approximately. This means than both types of instances are valid to host our service, due to the fact that there is still available margin in terms of CPU to add further viewers. For the moment and only considering the CPU consumption needs, both M3-medium and M3-large are valid instances to host our service under test.

Let's move now to the results in terms of end-to-end latency. These results are displayed in the Figures 5 and 6:

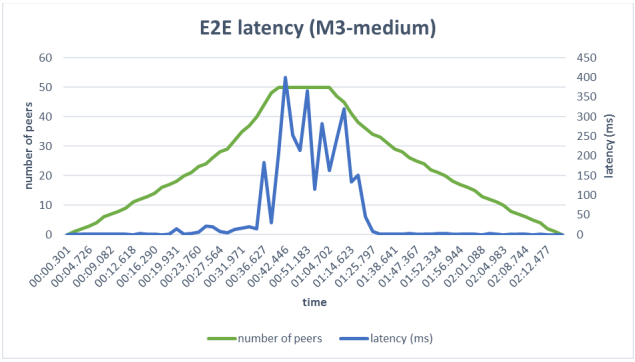

Figure 5. End-to-end latency in a M3-medium AWS instance.

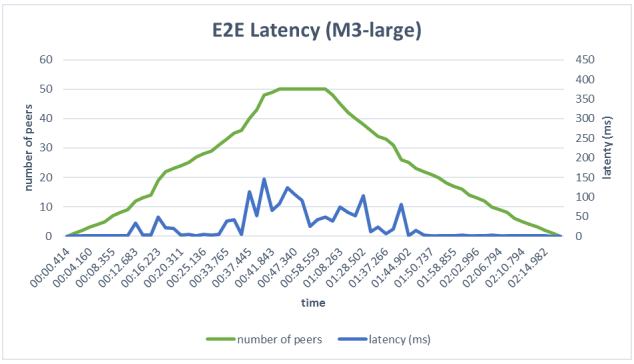

Figure 6. End-to-end latency in a M3-large AWS instance.

These diagrams show an important difference between both types of instances. On the one hand, we see that the end-to-end latency in a M3-medium instance reaches values around 300-400 milliseconds during the maximum occupation of viewers (50 clients, from second 00:36 to 01:14 in Figure 5). On the other hand, the values of latency obtained using a M3-large instance are better, since the maximum values are around 150 milliseconds, which is more suitable for our real-time videoconference service. This is measure is directly related with the CPU usage depicted in Figure 3. Due to the fact that media server is reaching the CPU limits of the underlying hardware, the end-to-end latency in increasingly higher.

In the light of these results, we can conclude that M3-large is a better choice to host our service due to the fact that the perceived latency for final users meets our requirements in terms of maximum concurrent users and latency.

7. CONCLUSIONS AND FUTURE WORK

The assessment of WebRTC services presents important challenges. Due to the nature of human to human of WebRTC, the QoE and latency measurements provides a key tool for evaluating perceived quality in WebRTC applications. This research presents a high level framework aimed to carry out different kind of V&V activities for WebRTC services. This framework has been created within the Kurento open source project, and it is called Kurento Testing Framework (KTF).

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 quality of experience (evaluation audio quality through PESQ). KTF uses Selenium WebDriver/Grid for automatic interaction with WebRTC applications. It handles a custom JSON notation to

specify the test scenario. This scenario describes how and where to locate the browsers used for testing. The possibilities are: local, remote, or *dockerized* browsers. This JSON allows compatibility testing with ease, just different specific browsers on different platforms. It is also very useful for setting different test runs in continuous integration environments.

The main shortcoming of KTF so far is that it only supports QoE assessment for audio by means of PESQ analysis. The next feature to be added to KTF is extending this analysis, supporting further methods to assess QoS for video (e.g. PEVQ, VQM, or SSIM).

8. ACKNOWLEDGMENTS

This work has been supported by the European Commission under projects NUBOMEDIA FP7-ICT-2013-1.6 (GA-610576) and FI-CORE FP7-2013-ICT-FI (GA-632893); by Spanish Ministerio de Educación under project Reactiv Media (TIN2013-41819-R); and by the Regional Government of Madrid (CM) under project Cloud4BigData (S2013/ICE-2894) cofunded by FSE & FEDER.

9. REFERENCES

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

[2] Lopez Fernandez, L., Paris Diaz, M., Benitez Mejias, R., Lopez, F. J., and Santos, J. A. Kurento: a media server technology for convergent www/mobile real-time multimedia communications supporting webrtc. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a* (2013), IEEE, pp. 1-6.

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

[4] Li, Y.-F., Das, P. K., and Dowe, D. L. Two decades of web application testing: a survey of recent advances. *Information Systems 43* (2014), 20-54.

[5] Robertson, S., and Robertson, J. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.

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

[7] ISO, B. 9000: 2005 quality management systems: fundamentals and vocabulary. *British Standards Institution* (2005).

[8] Möller, S., and Raake, A. *Quality of Experience*. Springer, 2014.

[9] Tran, H. A., Hoceini, S., Mellouk, A., Perez, J., and Zeadally, S. Qoe-based server selection for content distribution networks. Computers, IEEE Transactions on 63, 11 (2014), 2803-2815.

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

[11] Chikkerur, S., Sundaram, V., Reisslein, M., and Karam, L. J. Objective video quality assessment methods: A classification, review, and performance comparison. *Broadcasting, IEEE Transactions on 57*, 2 (2011), 165-182.

[12] Hands, D., Barriac, O. V., and Telecom, F. Standardization activities in the itu for a qoe assessment of iptv. *IEEE Communications Magazine 46*, 2 (2008), 78-84.

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

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

[15] Pinson, M. H., and Wolf, S. A new standardized method for objectively measuring video quality. *Broadcasting, IEEE Transactions on 50*, 3 (2004), 312-322.

[16] Rix, A. W., Hollier, M. P., Hekstra, A. P., and Beerends, J. G. Perceptual evaluation of speech quality (pesq) the new itu standard for end-to-end speech quality assessment part i-time-delay compensation. *Journal of the Audio Engineering Society 50*, 10 (2002), 755-764.

[17] Rix, A. W., Beerends, J. G., Hollier, M. P., and Hekstra, A. P. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on* (2001), vol. 2, IEEE, pp. 749-752.

[18] Rec, I. P. 863," perceptual objective listening quality assessment (polqa). *International Telecommunication Union, CH-Geneva* (2011).

[19] You, J., Reiter, U., Hannuksela, M. M., Gabbouj, M., and Perkis, A. Perceptual-based quality assessment for audio-visual services: A survey. *Signal Processing: Image Communication 25*, 7 (2010), 482-501.

[20] Cinar, Y., and Melvin, H. Webrtc quality assessment: Dangers of black-box testing. In *Digital Technologies (DT), 2014 10th International Conference on* (2014), IEEE, pp. 32-35.

[21] Amirante, A., Castaldi, T., Miniero, L., and Romano, S. P. Performance analysis of the janus webrtc gateway. In *Proceedings of the 1st Workshop on All-Web Real-Time Systems* (2015), ACM, p. 4.

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

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

[24] Singh, V., Lozano, A. A., and Ott, J. Performance analysis of receive-side real-time congestion control for webrtc. In *Packet Video Workshop (PV), 2013 20th International* (2013), IEEE, pp. 1-8.

[25] Weinberg, G. M. *Perfect Software: And Other Illusions about Testing*. Dorset House Publishing Co., Inc., 2008.