Testing Infrastructure for WebRTC applications

Boni García

boni.garcia@urjc.es

Contents

1. Introduction

- Web Real-Time Communications (**WebRTC**) is the umbrella term for several 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 (W3C) and the Internet Engineering Task Force (IETF):
  - W3C is defining the JavaScript APIs (Application Programming Interfaces) and the standard HTML5 tags to enable real-time media capabilities to browsers.
  - 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. Introduction

- Although still in its infancy, WebRTC is a technological initiative getting considerable worldwide attention
- WebRTC-based applications can be evaluated with respect to their multimedia conversation quality
- This work presents a **testing framework** aimed to simplify the testing process of WebRTC applications

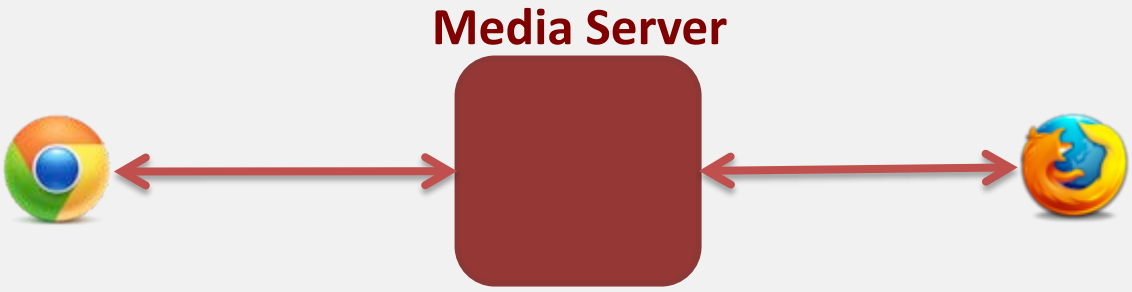
1. Introduction

- This work has been done by **Universidad Rey Juan Carlos** in the context of the projects
  - **FIWARE** FP7-2011-ICT-FI, GA-285248
  - **NUBOMEDIA** FP7-ICT-2013-1.6, GA-610576

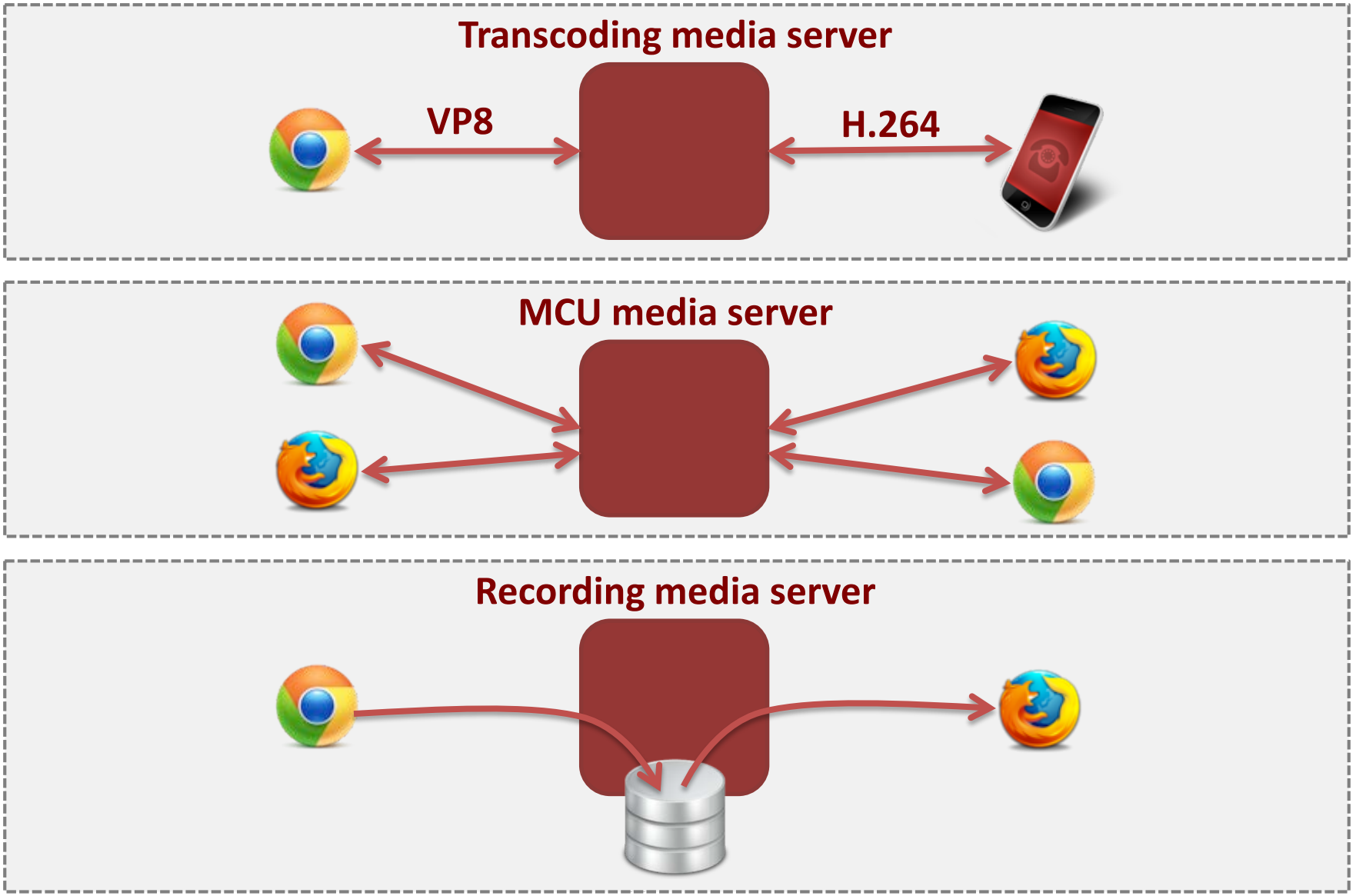
2. WebRTC server infrastructure

Peer-to-Peer WebRTC Application (without media infrastructure)

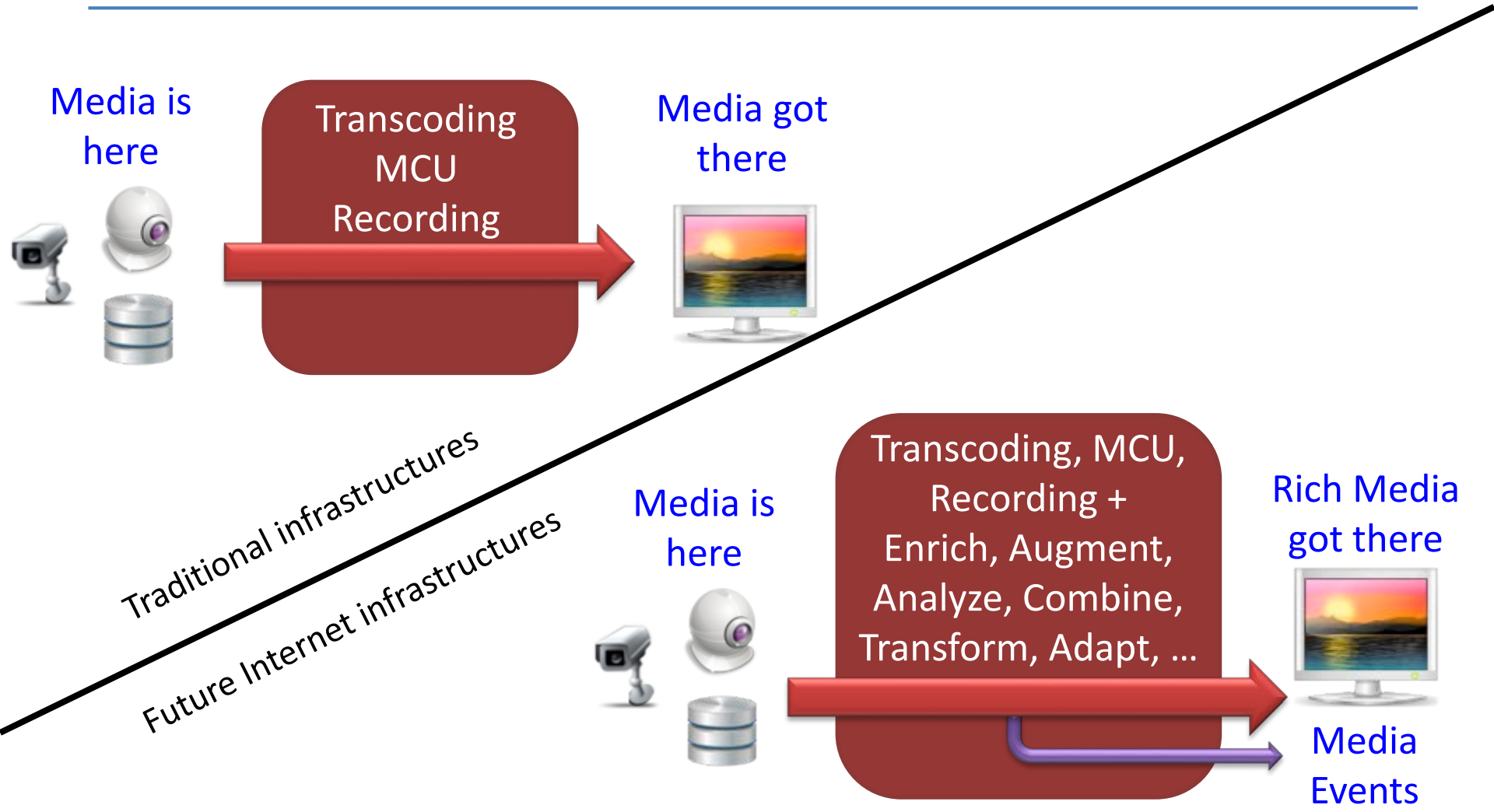
WebRTC Application with media server



2. WebRTC server infrastructure



2. WebRTC server infrastructure

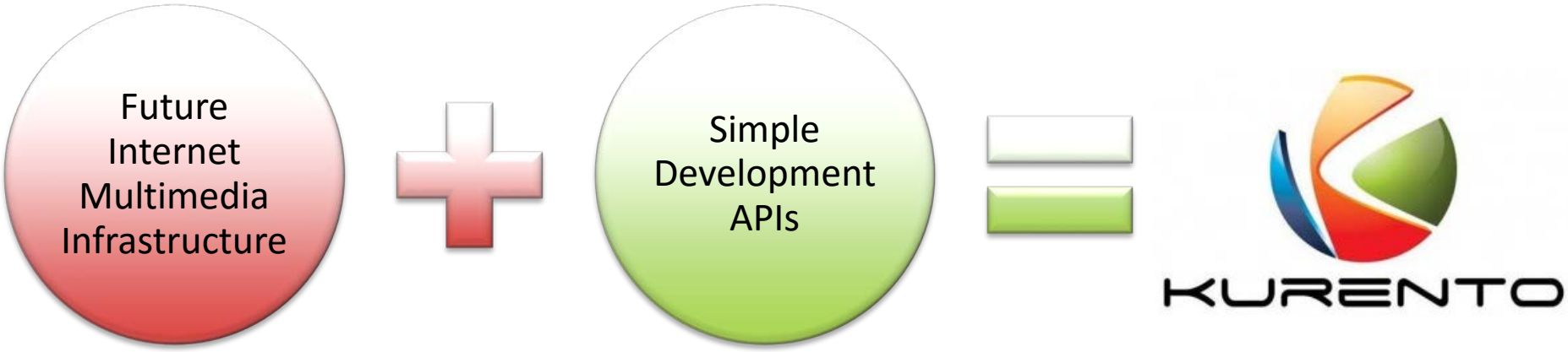


2. WebRTC server infrastructure

Complexity

2. WebRTC server infrastructure

Kurento: the equation

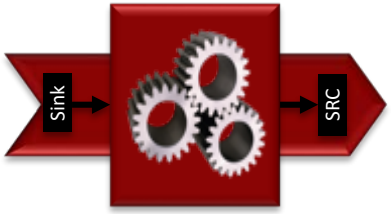


2. WebRTC server infrastructure

Key concepts: media elements and pipelines

- Media Element
  - Provides a specific media functionality
  - Ready to be used
  - New media elements can be added

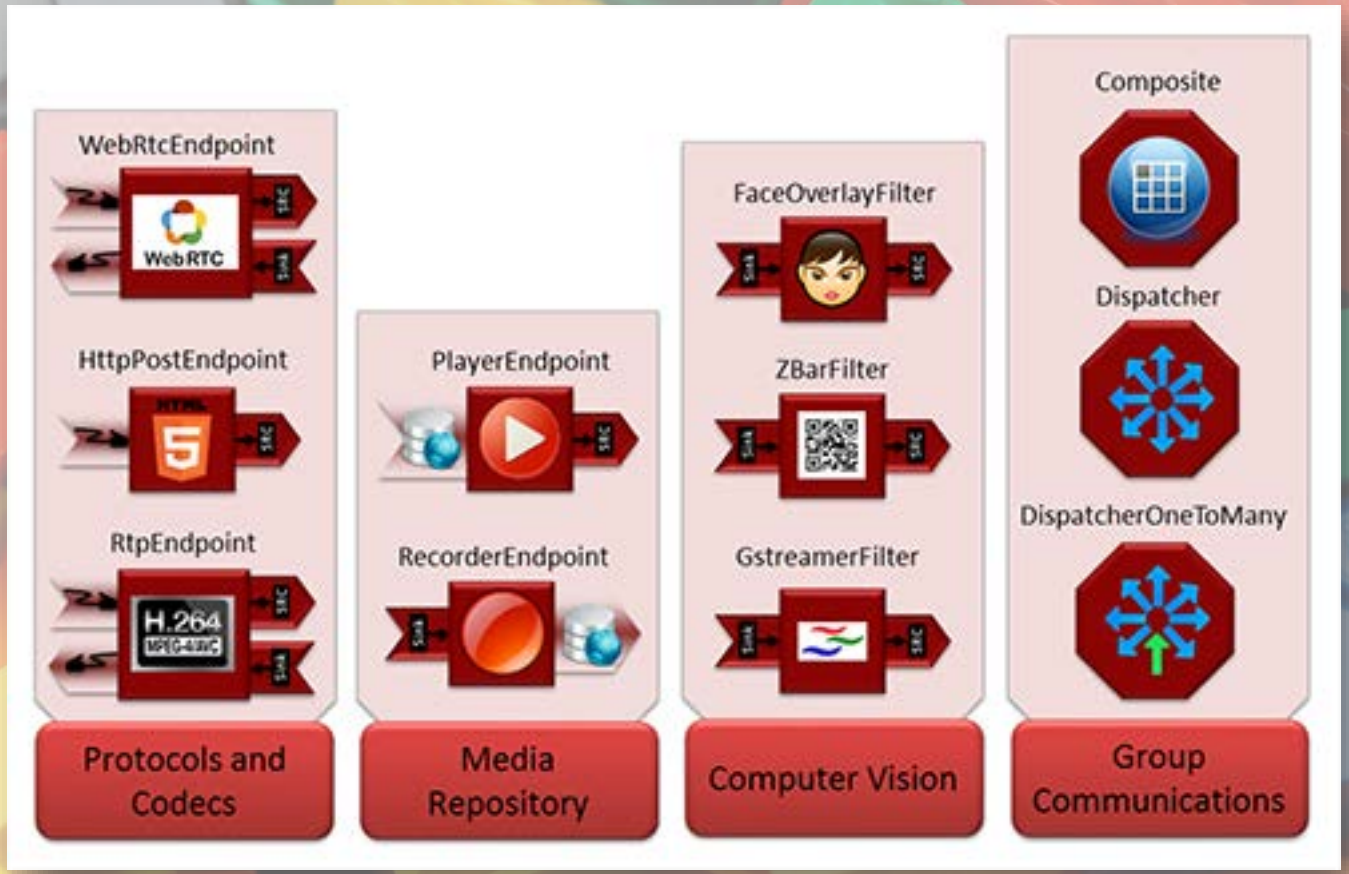
Media Element



Media Pipeline

- Media pipeline
  - Chain of media elements implementing the desired media logic
  - The Media Server provides the capability of creating media pipelines by joining media elements of the toolbox

2. WebRTC server infrastructure



3. WebRTC testing framework

- Kurento also provides a high level 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

3. WebRTC testing framework

- KTF is built on the top of two open source testing frameworks:
  - **Selenium** (automation of web testing)
  - **JUnit** (unit testing for Java)

- 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, which is a PaaS (Platform as a Service) cloud solution to support remote testing based on Selenium

3. WebRTC testing framework

- The configuration of browsers is called **test scenario**
- This scenario can be specified by means of a custom JSON notation

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

3. WebRTC testing framework

- 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

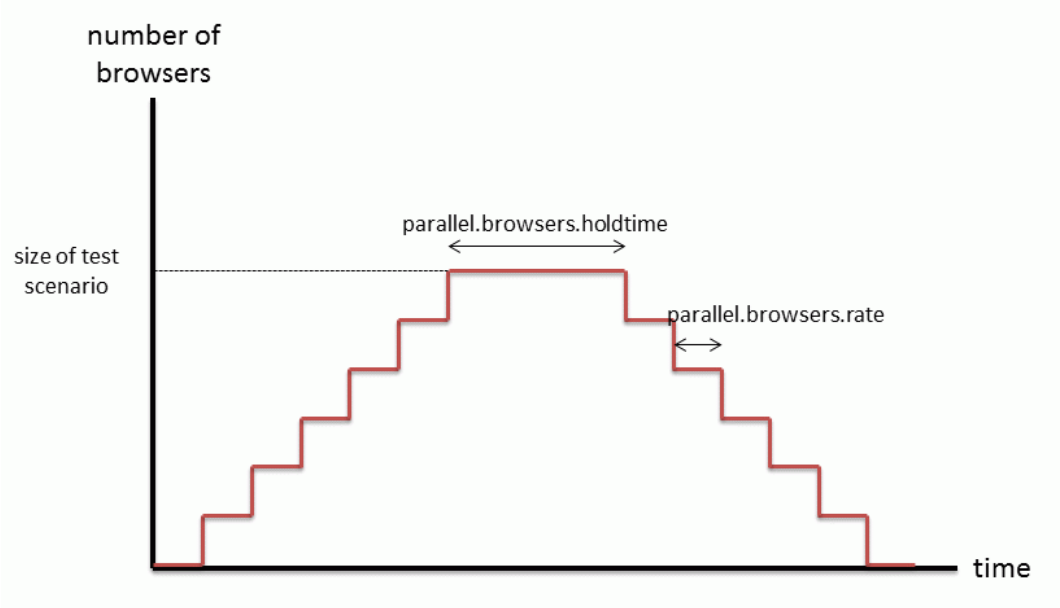
3. WebRTC testing framework

- **Functional** test capabilities:
  - Management of video tag events (subscription and assessment)
  - Analysis of media in video tag based in color detection and comparison (RGB)

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

3. WebRTC testing framework

- **Performance** test capabilities:
  - Ramp of browsers



3. WebRTC testing framework

- **Performance** test capabilities:
  - Monitoring of the system under test
    - Time (relative to the start of the test)
    - Number of incoming clients
    - CPU usage (percentage)
    - Memory usage (number of bytes and percentage out of the total)
    - Swap memory usage (number of bytes and percentage out of the total)
    - Network interfaces usage (number of sent and received bytes in each of the network interfaces)

3. WebRTC testing framework

- **Performance** test capabilities:
  - Latency analysis based on color comparison

- **Quality** test capabilities:
  - PESQ (Perceptual Evaluation of Speech Quality) is supported to evaluate the received audio quality

4. KTC code examples

- KTC can be used as a **Maven** dependency

```
<dependency>
   <groupId>org.kurento</groupId>
   <artifactId>kurento-test</artifactId>
   <version>6.0.0</version>
   <scope>test</scope>
</dependency>
```

4. KTC code examples

- The structure of a KTC JUnit test case is:

```
public class MyTest extends KurentoTest {

        public MyTest(TestScenario testScenario) {
                super(testScenario);
        }

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

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

4. KTC code examples

- Snippets for **functional** assessment:

```
// 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));
```

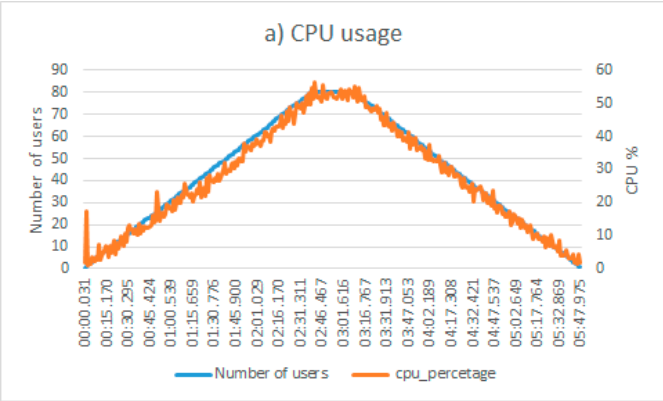
4. KTC code examples

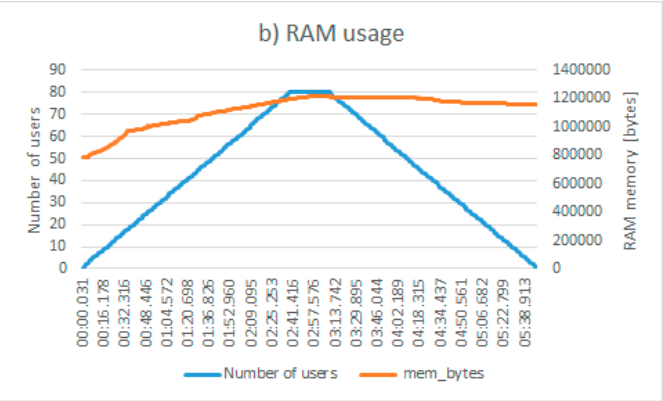
- Snippets for **performance** assessment:

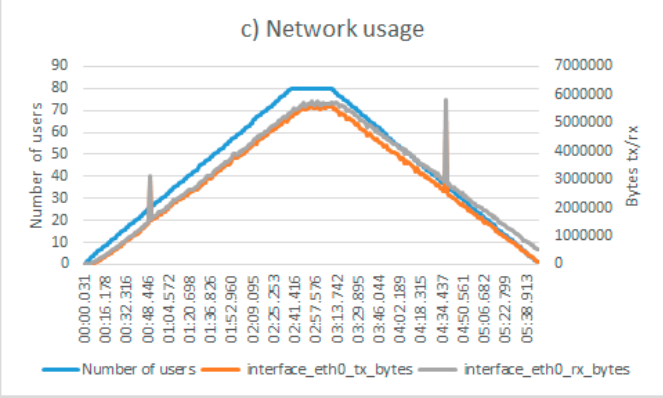
```
private SystemMonitorManager monitor;

@Before
public void setup() {
            String host = "127.0.0.1";
            String login = "user";
            String 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
                        }
            });
}
```

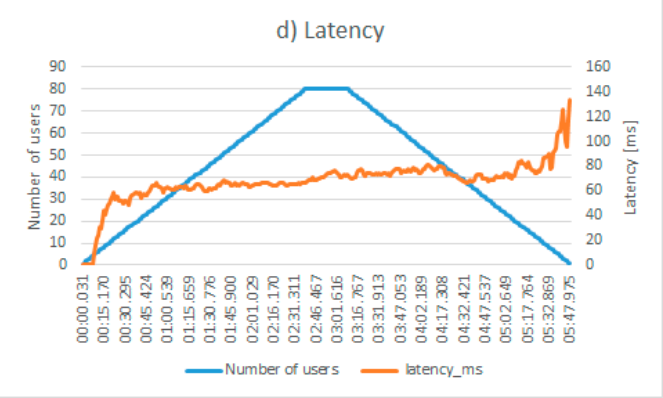
4. KTC code examples

- **Performance** results:









4. KTC code examples

- Snippets for **quality** assessment:

```
int sampleRate = 16000; // samples per second
float minPesqMos = 3; // PESQ MOS [1..5]
String audioUrl =
"http://files.kurento.org/audio/10sec/fiware_mono_16khz.wav";

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

5. Conclusions and future work

- Testing of WebRTC applications presents important challenges for practitioners
- This research presents a high-level testing framework to perform complete assessment of WebRTC-based applications: Kurento Testing Framework (KTF)
- Future work
  - Video QoE assessment (e.g. PEVQ, VQM, or SSIM)
  - Use of **Docker** as new scope for browsers