Analysis of video quality and end-to-end latency in WebRTC

Fifth IEEE International Workshop on
Quality of Experience for Multimedia Communications - QoEMC2016
IEEE GLOBECOM 2016
Washington, DC USA, 8 December, 2016

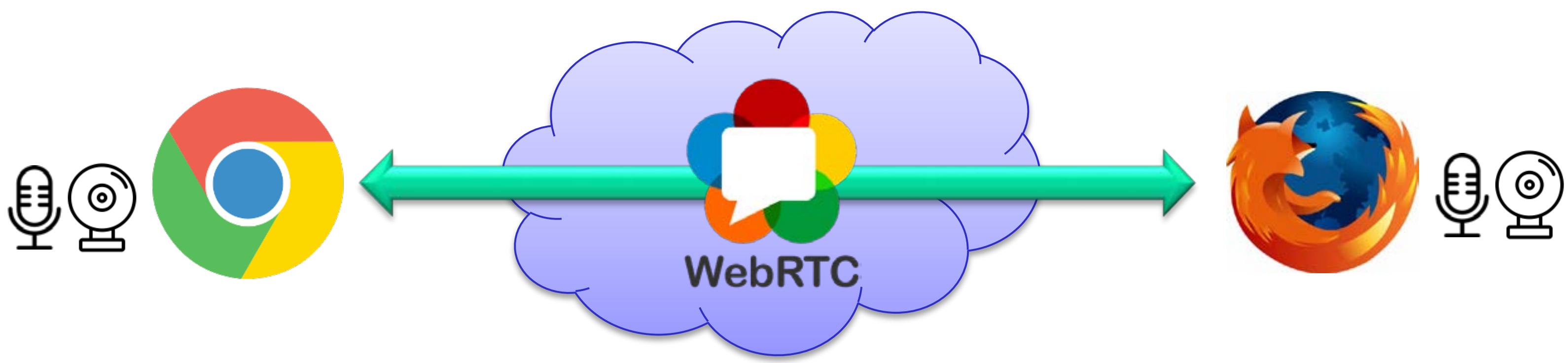
Boni García
Universidad Rey Juan Carlos (Spain)
boni.garcia@urjc.es

Universidad Rey Juan Carlos

Contents

1. Introduction

- **WebRTC** is the set umbrella term for a number of novel technologies having the ambition of bringing high-quality *Real Time Communications* to the Web
  – W3C (JavaScript APIs): *getUserMedia*, *PeerConnection*, *DataChannels*
  – IETF (protocol stack): ICE, SDP, TURN, STUN, DTLS, …



1. Introduction

- **Kurento** is an open source framework for WebRTC aimed to created applications with advance media capabilities (e.g. augmented reality, video content analysis)
- It is composed by a Media Server and a set of APIs
- Kurento has been recently acquired by **Twilio**

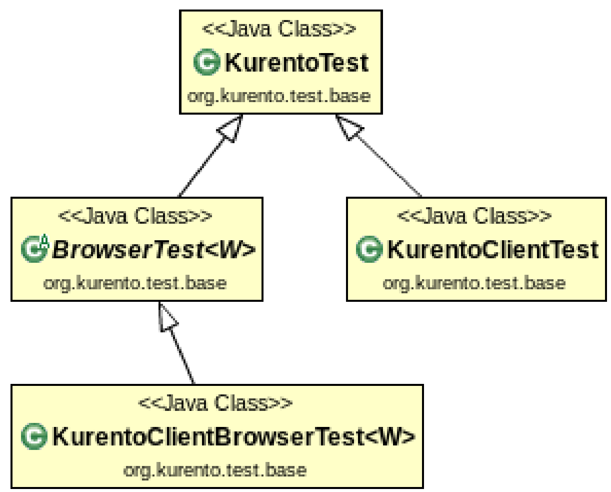
1. Introduction

- Based on our experience, we have created a testing framework for WebRTC applications
- This framework exposes high-level capabilities for testers, supported by a continuous delivery infrastructure for the DevOps team

Kurento Testing Framework

2. Testing Framework for WebRTC

- Kurento Testing Framework has been built upon well-known testing technologies, such as JUnit, Selenium, Jenkins
- It exposes an API for testers with advanced testing capabilities
  1. Seamless browser handling
  2. Functional assessment
  3. Quality of Experience



2. Testing Framework for WebRTC

→ *Seamless browser handling*

- KTF introduces the concept of **test scenario**, which can be seen as the collection of browsers in which a given test case is going to be exercised
- Each browser have a given *scope*:
  - Local machine
  - Remote machine
  - Remote PaaS (Saucelabs)
  - Docker

2. Testing Framework for WebRTC

→ *Seamless browser handling*

- KTF defines a custom **JSON** notation to define the different parameters for test scenarios

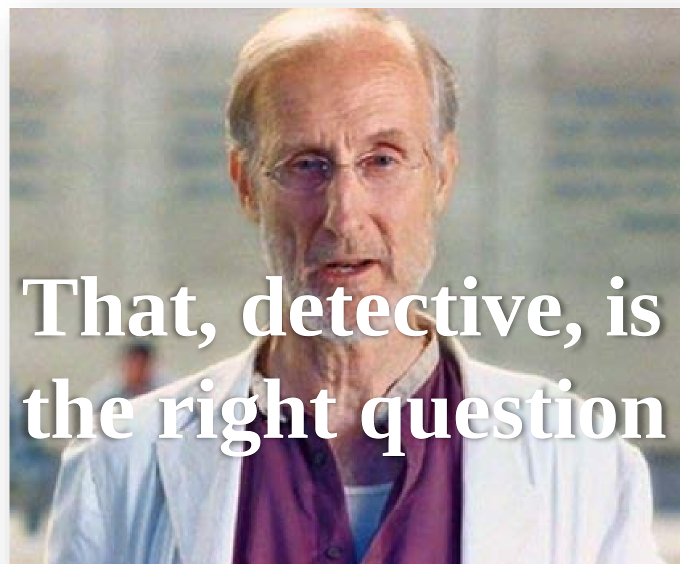

```
{
   "executions" : [
      {
         "peer1" : {
            "scope" : "local",
            "browser" : "chrome"
         },
         "peer2" : {
            "scope" : "docker",
            "browser" : "firefox"
         }
      },
      {
         "peer1" : {
            "scope" : "saucelabs",
            "browser" : "edge",
            "version" : "13",
            "platform" : "win10"
         },
         "peer2" : {
            "scope" : "remote",
            "browser" : "safari"
         }
      }
   ]
}
```

2. Testing Framework for WebRTC

→ *Functional assessment*

- Automated interactions with browsers
- Subscription to media events (such as *playing*)
- Color detection for media in HTML5 video tags

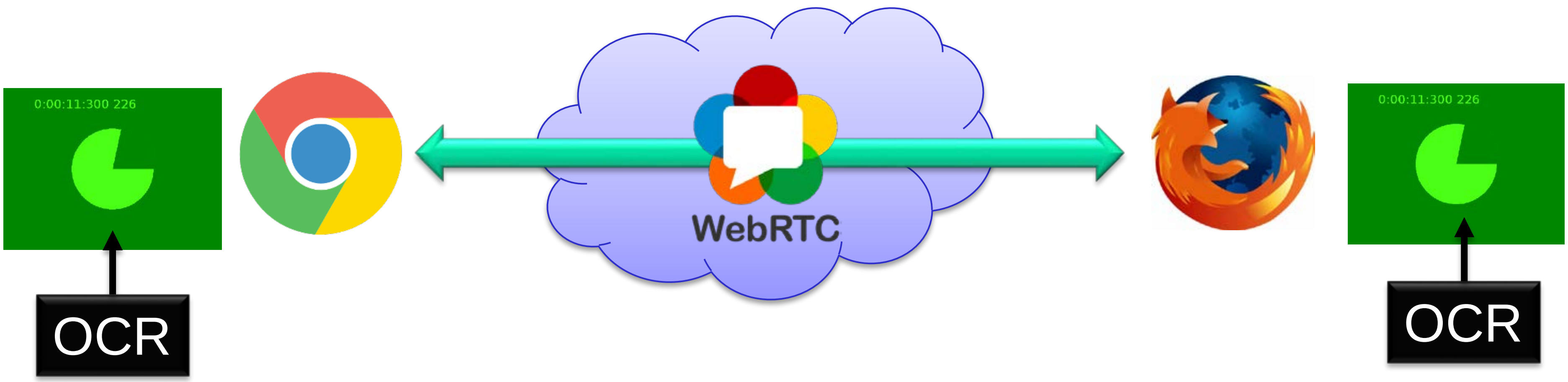
With this capabilities we are able to detect whether or not media is reaching browsers, but:
Is that media as expected?



2. Testing Framework for WebRTC

→ *Quality of Experience*

- **End-to-end latency** meter
- Using the fake user media provided out of the box by Google Chrome



$$\text{E2E-latency} = t_{receiver} - t_{sender}$$

2. Testing Framework for WebRTC

→ *Quality of Experience*

- Problem: how to make sampling?
- Solution: synchronize presenter and viewer by means of NTP (Network Time Protocol)
- Implementation:
  - JavaScript logic is injected in each browser
  - This logic gathers the clock from media using HTML5 Canvas
  - After that the set of images is processed using Tesseract OCR

2. Testing Framework for WebRTC

→ *Quality of Experience*

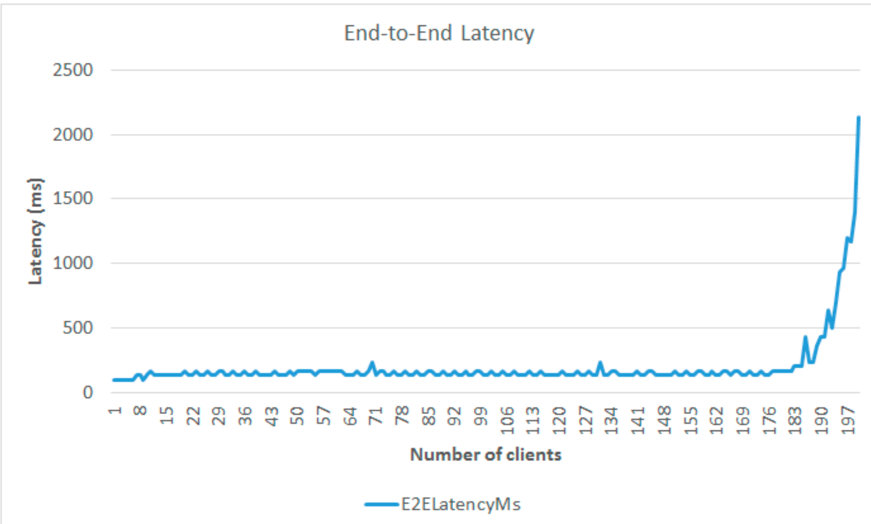
- Moreover, KTF is integrated with existing QoE algorithms
  - **PESQ** (Perceptual Evaluation of Speech Quality) for audio
  - **SSIM** (Structural similarity) for video
  - **PSNR** (Peak Signal-to-Noise Ratio) also for video
- Finally, KTF gathers WebRTC statistics and compile that data as CSV files

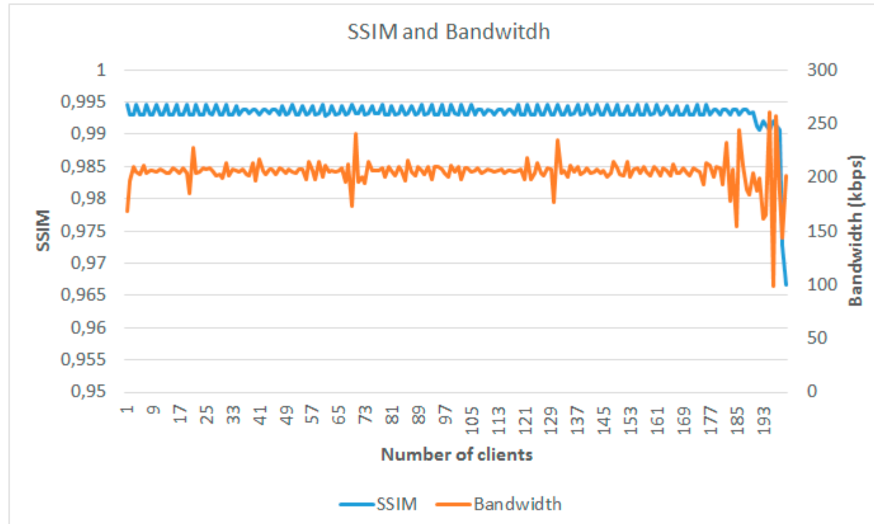
3. Case Study: WebRTC broadcasting

- Test scenario (1 to N video communication)
  - 1 local browser acting as presenter
  - 200 remote browser acting as viewers
  - A new viewer is connected to the broadcasting each second
  - Total test time: 200 seconds
- System features
  - The machine hosting the service is a medium cloud instance (2 VCPU, 4 GB RAM)

3. Case Study: WebRTC broadcasting

- Results





The experiment shows that the system supports up to around 190 concurrent users

4. Conclusions

- Testing WebRTC based application, consistently automated fashion is a cumbersome challenging problem
- We have created a framework to assess WebRTC applications
  - Seamless browser handling (JSON test scenario)
  - Quality of Experience (end-to-end latency, integration with existing algorithms)
- Next step: improve scalability by using *fake browsers*

Thank you

QA?

Boni García

Departamento de Sistemas Telemáticos y Computación (GSyC)

Universidad Rey Juan Carlos

boni.garcia@urjc.es

Universidad Rey Juan Carlos