Toward Operative QoE Models using Virtual Multimedia IP-based Streaming Services

Sofiene Jelassi, Gerardo Rubino

Toward Operative QoE Models using Virtual Multimedia IP-based Streaming Services

Sofiene Jelassi[1] and Gerardo Rubino[2]

[1]University of Rennes1, Rennes, France
Email: sofiene.jelassi@univ-rennes1.fr

[2]Inria Rennes – Bretagne Atlantique, Rennes, France
Email: gerardo.rubino@inria.fr

***Abstract*—The measurement of the Quality of Experience (QoE) of multimedia streaming services (MMSS) over IP networks may be realized thanks to objective QoE models. They are mathematical functions transforming metrics from technical to user domains. An interesting category of QoE models predicts QoE scores of MMSS at runtime, letting use them for the monitoring operation. This requires integrating them inside the production environments, i.e., *where the actual MMSS are consumed by end-users*. This aspect is often neglected by QoE modelers that focus mainly on the accuracy and fitness of their designed models with respect to a given set of settings and conditions. As a consequence, a considerable technical effort should be made in order to bring them from the laboratory to the production environments. This obviously discourages MMSS providers to easily accept and adopt them. For the sake of enhancing QoE models integration, we propose `Mesukal`, a software-layer ensuring portability of QoE models over a variety of underlying MMSS, e.g. YouTube or Netflix. Specifically, `Mesukal` acts as a Java Virtual Machine (JVM) enabling to build portable applications over different OS, e.g. Windows, Linux or MacOS. `Mesukal` can be considered as a virtual MMSS that is able to seamlessly interact with QoE models, on the one hand, and arbitrary real MMSS, on the other hand. Each considered MMSS over IP networks is appropriately virtualized by a dedicated *Mesukal App*. Besides real MMSS, `Mesukal` can be used to instantiate experimental MMSS where the accuracy and portability of QoE models may be inspected and checked under controlled conditions. The inputs needed by the concerned QoE models are fetched from each real MMSS using probes that are tailored following the technology used by the considered multimedia service. In addition, `Mesukal` includes a rich GUI dashboard that enables to inspect QoE results.**

***Index Terms*—IP-based multimedia streaming, Modeling of Quality of Experience, QoE monitoring and reporting**

I. Introduction

QoE measurement is nowadays a well-established research topic. Since its inception, two decades ago, huge steps forward have been made to define, measure, model and standardize it [1]. In contrast to technical metrics, QoE measurement considers human perception and behavior in order to rate a given service. This is actually very insightful to promote a considered technology. A lot of attention has been paid to model QoE using mathematical functions allowing to (a) avoid redundant, costly and time-consuming subjective in-lab testing and (b) extend the application scope of QoE measurement, with some partial successes. We call "in-the-box" a QoS model developed by the concerned service provider, and "out-of-the-box" those developed by third parties (see Fig. 1). Mesukal is concerned basically by the latter. They can be actually segregated into media- and equipment-based. The former category, denoted sometimes as black-box approach, examines the processed media, e.g. audio, image or video solely at the end-points of a considered media processing chain. However, the latter category, denoted sometimes as glass-box approach, examines the effects of each individual physical element, e.g. microphone, encoder, network, playback algorithm, or echo canceler, located in the media processing chain between end-points. Each QoE model defines tailored inputs that should be provided in the right format and margin in order to get faithful outcomes. In this work, we consider QoE models that require only data gathered at the exit point of MMSS given that our focus on monitoring.

Out-of-the-box models are built for baseline "abstracted" MMSS under certain settings, e.g. H.265 or VP9 and conditions, e.g. outdoor or indoor, that may be likely encountered in the production environment. This choice is made for increasing their application scope, and thus improving their added-value. This implies that MMSS developers should select suitable out-of-the-box QoE models before plugging them inside their special development and production environments. This is well-recognized to be tedious and time-consuming processes needing thorough knowledge about the QoE area coupled with a cumbersome technical effort. Moreover, the out-of-the-box QoE models should be maintained and upgraded with respect to each MMSS release that is a supplementary source of troublesome and confusion. These embarrassing features explain, to certain extent, why MMSS developers are more and more reluctant to include out-of-the-box QoE models. Indeed, this is often judged as an accessory incurring an unnecessary significant overhead. That is why, service developers prefer building their own in-the-box QoE models, such as Netflix and YouTube. The resulting in-the-box QoE models can be easily maintained and integrated within the service universe in an agile fashion. Hence, for the sake of supplying hands-on out-of-the-box QoE models, they should be *portable* in the sense that they should be operative over different underlying MMSS service technologies, again such as Netflix and YouTube, to

978-1-7281-7705-2/21/$31.00 ©2021 European Union

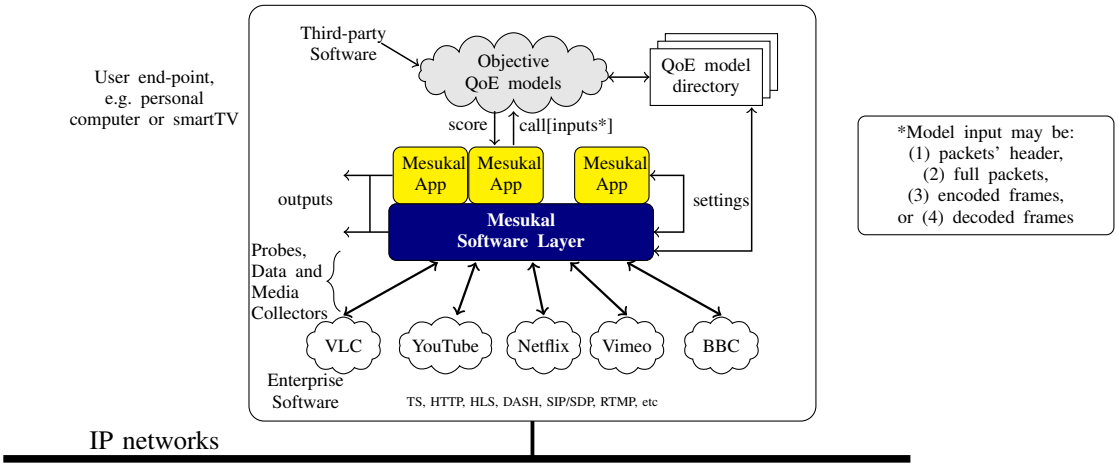

Fig. 1. Mesukal overview

keep the same illustrative examples. Moreover, they should be *pluggable* in the sense that they can be changed, updated or replaced without interfering with the underlying MMSS. This is the main goal of our work, and our focus in this demo.

This work addresses the issue of making portable and pluggable out-of-the-box QoE models over distinguished underlying MMSS. We actually used directive ideas followed by the popular JVM (Java Virtual Machine) built in order to execute a unique source code over different computing machines. This is basically performed by inserting a specialized virtualization software layer, denoted to as `Mesukal`, that abstracts the underlying real MMSS over IP (see Fig. 1). The goals of our demo are then twofold: (1) to illustrate for beginners in which way `Mesukal` should be setup, configured and used through two case studies and (2) demystify `Mesukal` for proficients by going through its internal components and their interactions. We strongly believe that `Mesukal` will be a useful instrument enabling continuous integration, a.k.a. DevOps, of QoE models inside production environments. Please note that as previous text suggests, Mesukal should be installed like a JVM by the user, but obviously a service provider should probably be included in the development of a Mesukal software layer.

The rest of the paper is organized as follows. In Section 2, we give an overview about workflow and architecture of `Mesukal`. In Section 3, we present two QoE case studies using `Mesukal`. We conclude in Section 4.

II. Mesukal Overview

The schema plotted in Fig. 1 highlights the general `Mesukal` workflow and its basic interactions with QoE models and real MMSS. As we can see, the virtual MMSS instantiated by `Mesukal` fetch appropriate QoE model inputs by calling real MMSS using their provisioned API, e.g. YouTube or Vimeo API, and/or by inspecting suitable transport ports and media channels. The QoE models to invoke and their parametrization are specified in the settings that may be given in a manual or scripted way. Please note here that one or several QoE models may be invoked with respect to the intended goals; let's say comparing the accuracy of two QoE models or the alpha and beta versions of the same one. Once inputs are received and their correctness is validated, virtual MMSS transmit them, after an optional supplementary processing, toward the concerned models. The latter compute QoE scores and send them to the virtual MMSS. Typically, the received QoE scores are sent to the real-time plotting and/or saving components according to the specified settings.

To handle the technological variety of MMSS, we created one *Mesukal App* per enclosed MMSS that instantiates the corresponding virtual MMSS. Basically, a *Mesukal App* can be seen as an orchestrator that consolidates many software components, which may be classified into data and control planes. The data plane includes all software components required to actively accede and retrieve locally media content delivered by a given MMSS. Moreover, it includes software components offering VCR-like commands, which enable to play and/or record fetched media content for further processing. Needless to say, software components in the data plane are highly dependent on the technology used by the MMSS. On the other hand, the control plane includes all software components required to fetch the QoE model inputs and to deliver the computed QoE scores to the appropriate programs, namely the plotter and the logger. The fetching process relies on a set of probes customized with respect to the parametrization used by each MMSS, e.g. media format, transport protocol or port numbers. The included probes search information from three main sources: statistics and meta data returned by the MMSS API, captured NDU (Network Data Unit) and captured ADU (Application Data Unit). The available probes are activated according to the required inputs of each QoE model. Two case studies of *Mesukal App* will be discussed in Section III.

III. Mesukal Case Studies

The prevailing *Mesukal App* supported by `Mesukal` may be coarsely split into three categories: Experimental, Live TV channels, and social and commercial VoD services. We

note that Mesukal has been mainly developed in `Python` and `Tkinter`. The following case studies discuss relevant operations that may be achieved with our tool.

Case Study A: This case study seeks to compare and examine the behavior and the accuracy of three QoE models of Digital Video Broadcasting (DVB) sessions over IP networks. The first one is specified in the ITU-T Rec. P1201.2 [2], and the second and third ones are proposed in [3]. Please note that we manipulated our implementation of those QoE models that may be slightly different than the original ones. All considered QoE models use as input a list of NDU and need several meta-data about the configuration of the receiver end-point. These specifications led us to design a DVB *Mesukal App* as follows: The data plane instantiates an experimental MMSS composed of a VLC-based streaming server and client sides. They are created and managed programmatically using `python-vlc` library. The streaming server is configured in order to actively transmit MPEG-2 video content using Transport Stream (TS) over RTP/UDP to a set of predefined streaming clients. The time-varying network impairment conditions are applied using the Traffic Control (TC) utility. They are specified using packet loss ratio, one way delay and jitter incurred during a DVB session. The control plane includes one probe that captures NDU including TS packets. A second probe is used in order to dissect tables belonging to program-specific information (PSI), namely Program Association Table (PAT) and Program Mapping Table (PMT). The computed QoE scores are sent to the plotter that displays them at runtime. For sake of clarity, we give in Fig. 2 a snapshot of GUI implemented by this *Mesukal App*.

Case Study B: This case study seeks to check the behavior and accuracy of the QoE model proposed in [4], denoted as PAL QoE model, which has been designed to automatically assess QoE of services using Google VP9 Codec, e.g. YouTube or Netflix. To do that, we need to establish an interaction between PAL QoE model and the real service. This is actually what for instance YouTube *Mesukal App* provides. It has been designed as follows: the data plane includes components enabling to acceding and retrieving arbitrary YouTube video content at runtime. The user can manually specify the URL where YouTube video content is located. Moreover, it enables selecting a specific video format among available ones. The loading process of media content relies basically on `ffmpeg`, which pulls video segments and buffer them in a dedicated pipe. In order to play received video content at runtime, we implement a simple streaming video player using `OpenCV` library. It follows a static playback policy where video frames are fetched at a regular frequency. On the other hand, the control plane includes three kinds of probes. The first one relies on the public provisioned YouTube API accessible via Restful API. It returns stats about displayed video content, such as numbers of likes and views, video duration and size, etc. The second probe collects needful meta-data found at the application-layer using `ffprobe`, which are bitrate, resolution and frame rate. Finally, the third probe intercepts NDU including video content and puts them in a PCAP file.

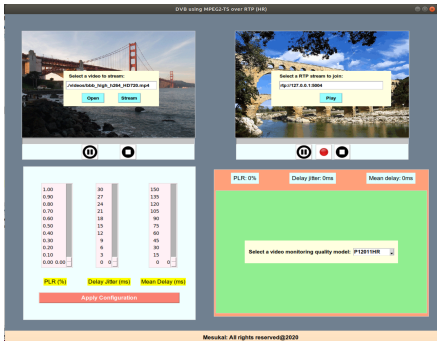

Fig. 2. A controlled streaming experiment

This operation relies essentially on `Tshark`. Our preliminary inspection shows that the PAL QoE model achieves reasonably good results, but we believe that more profound experiments are needed for a better judgment.

IV. Demo Equipment and Organization

A Beta `Mesukal` version is maintained by our research team *Dionysos* at Inria. We also maintain a ready-to-use VM where a stable version of `Mesukal` is configured and installed. A link will be given to the participants in order to download `Mesukal` VM that runs over `VirtualBox`. A detailed guideline will be provided in order to reproduce various showcases, explained remotely, on their personnel computing machine.

V. Conclusion

`Mesukal` is a step toward building operative and portable QoE models over a variety of MMSS. It consolidates a collection of software components inside *Mesukal App* in order to virtualize experimental or existing MMSS. It will help in improving and extending the use of QoE models inside MMSS in production environments. Moreover, it offers a way of analyze QoE models as they appear and evolve.

Of course, it is required a full understanding of the service in order to extract the suitable inputs required by a set of models. Moreover, a Mesukal App should follow the evolution of the service to keep capturing those input parameters. Moreover, mechanisms for protecting user privacy should be be considered.

References

[1] N. Barman and M. G. Martini, "QoE Modeling for HTTP Adaptive Video Streaming A Survey and Open Challenges," *IEEE Access*, vol. 7, pp. 30 831 – 30 859, March 25, 2019.

[2] H. T. Cruz, D. Pal, and T. Triyason, "A survey of standardized approaches towards the quality of experience evaluation for video services: An itu perspective," *International Journal of Digital Multimedia Broadcasting*, vol. 2018, 2018.

[3] D. Hernando, J. E. L. de Vergara, D. Madrigal, and F. Mata, "Evaluating quality of experience in iptv services using mpeg frame loss rate," in *Proc. of 2013 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, Paris, France, 17-19 June 2013.

[4] C. Kotropoulos, D. Pal, and V. Vanijja, "A no-reference modular video quality prediction model for h.265/hevc and vp9 codecs on a mobile device," *Advances in Multimedia*, vol. 2017, 2017.