TAPAS-360°: A Tool for the Design and Experimental Evaluation of 360° Video Streaming Systems

https://github.com/c3lab/tapas360

Giuseppe Ribezzo, Luca De Cicco, Vittorio Palmisano, Saverio Mascolo
Politecnico di Bari
Bari, Italy
{name.surname}@poliba.it

ABSTRACT

Video streaming platforms are required to innovate their delivery pipeline to allow new and more immersive video content to be supported. In particular, Omnidirectional videos enable the user to explore a 360° scene by moving their heads using Head Mounted Display devices. Viewport adaptive streaming allows changing dynamically the quality of the video falling in the user's field of view. In this paper, we present TAPAS-360°, an open-source tool that enables designing and experimenting all the components required to build omnidirectional video streaming systems. The tool can be used by researchers focusing on the design of viewport-adaptive algorithms and also to produce video streams to be employed for subjective and objective Quality of Experience evaluations.

CCS CONCEPTS

• **Networks** → **Network experimentation**; • **Computing methodologies** → *Virtual reality*.

KEYWORDS

360-degree video; Adaptive video streaming; DASH

ACM Reference Format:
Giuseppe Ribezzo, Luca De Cicco, Vittorio Palmisano, Saverio Mascolo. 2020. TAPAS-360°: A Tool for the Design and Experimental Evaluation of 360° Video Streaming Systems: https://github.com/c3lab/tapas360. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20), October 12–16, 2020, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3394171.3414541

1 INTRODUCTION

Internet media delivery has evolved to a mature technology that is at the base of popular on-line video streaming services such as YouTube and Netflix. Today, the challenge is to design media delivery systems that are also able to stream immersive video content which comprise Omnidirectional Videos, or 360° videos, and volumetric content, or 6DoF videos. Immersive videos add several dimensions to the classical 2D videos and allow user to explore a scene from different point of views, enhancing the overall viewing experience. Specifically, 360° videos are produced by capturing a scene in all directions simultaneously with a number of video cameras. The user, equipped with a Head Mounted Display (HMD), is free to explore the recorded environment. Due to the inherently larger resolutions entailed by 360° videos, the design of viewport adaptive algorithms which dynamically select which portion of the spherical video to be streamed at the highest quality are now a hot research topic.

Experimental research in this area requires building a full pipeline which starts from immersive content generation and ends at video consumption using a player. All these components must implement the required features to make the interaction with the 360° scene possible. If the research community has proposed several tools for the design and experimental evaluation of Adaptive BitRate (ABR) algorithms [2, 10], the same cannot be said about omnidirectional videos. As a result, it is quite difficult to reproduce the results of different viewport adaptive algorithms and to make fair comparison among such algorithms.

In our previous work [2], we proposed a TAPAS a framework allowing the researcher to only concentrate on the design of the ABR algorithm without the need of implementing a complete player for classic 2D adaptive streaming. Building on the core functionalities of TAPAS, this work presents TAPAS-360°, a tool which significantly extends TAPAS and allows rapid prototyping of viewport adaptive control algorithms used for the distribution of immersive content. The tool has been designed to decrease the computational load required for each video stream generated on the testing machine. In particular, since panoramic video decoding is the process having the greatest impact on the computational load, TAPAS-360° allows to optionally disable the video decoding process while keeping the dynamics of the playout buffer, and therefore of the overall video streaming session, unchanged. Consequently, it is possible to carry out accurate experiments involving a large number of concurrent flows using the same machine. This feature is fundamental for experimentally studying the performance of the video distribution system as the number of streams that insist on the same link changes. Moreover, the tool can be easily used in combination with common network emulation tools such as, f.i., MahiMahi[1] to perform experiments in a controlled network environment, allowing the reproducibility of the obtained results. Additionally, traces of head movement [1, 6] can be used to experimentally evaluate the performances of the viewport adaptive algorithms with respect to

MM '20, October 12–16, 2020, Seattle, WA, USA

ACM ISBN 978-1-4503-7988-5/20/10…$15.00
https://doi.org/10.1145/3394171.3414541

[1] http://mahimahi.mit.edu/

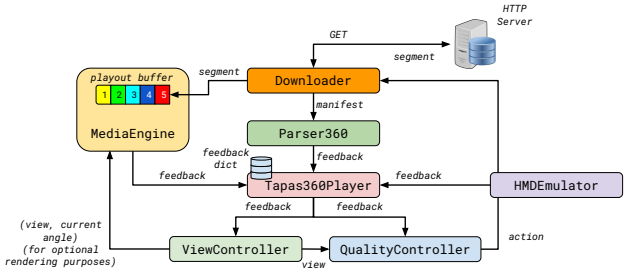

Figure 1: Workflow of the TAPAS-360° tool

different viewing patterns, or to test field-of-view prediction algorithms. At the best of our knowledge, there are no open-source tools available that implement the features described above. TAPAS-360° currently supports only viewport-adaptive schemes that download the whole sphere, whereas supporting schemes that download only portions of the whole sphere is a planned feature to be implemented in the future.

2 BACKGROUND

Classical 2D adaptive video streaming requires the video content to be encoded at different bitrates and resolutions, generally referred to as video *levels* or *representations* which are temporally divided in *segments* of fixed durations. The HTTP server indexes these segments and produces a *manifest* which provides relevant information about the segment such as encoding bitrates, resolution, duration, and a URI pointing to the segment. The client fetches and analyzes the manifest to build a data structure that is used for downloading video segments. For each video segment, a controller – the ABR algorithm – dynamically decides which video level to request to obtain the best possible quality given the bandwidth available with the constraint of avoiding interruptions in playback. Both the HLS and MPEG-DASH standards require that the ABR algorithm is implemented on the client.

Differently from classical 2D videos, immersive video streaming systems manipulate the omnidirectional content where the part of the video currently visualized by the user (the user *viewport*) is roughly one-sixth of the entire video resolution. Streaming such a large resolution video on the Internet entails a remarkable waste of network bandwidth. In an effort to reduce the required bitrate, recent literature has focused on designing *viewport adaptive* strategies mainly relying on the *tiling* technique [3, 5, 9]. *Tiling* technique requires the immersive content to be spatially divided into several portions named *tiles*. Viewport adaptive algorithms aim at fetching the tiles falling into user's viewport at higher quality, while keeping the other at lower quality (or not delivered at all).

3 TAPAS-360°

Figure 1 shows a block diagram of the TAPAS-360° tools highlighting the main components and the corresponding connections between them. In addition to the features of rapid prototyping, flexibility of use, and modularity inherited from the predecessor [2], TAPAS-360° integrates a whole series of new modules allowing the management of immersive content compliant with the MPEG-DASH SRD specifications [4]. In the following, the essential details of the components is provided, whereas specific implementation details are left to the documentation of the project available in the project repository.

3.1 Tapas360Player

`Tapas360Player` is the central module that deals with orchestrating the operations of all TAPAS-360° components. This module implements the player logic and updates the log files that are populated during the experiments and that can be used in the post-processing phase for analyzing the performance of the implemented algorithms. The communication between modules is implemented through the exchange of a `feedback` dictionary. This dictionary contains all the pieces of information that are useful for performing the experiments and to pass data from one module to the other.

The `play` method is used to start the experiment and manages the user interaction, initializes the feedback dictionary, and orchestrates the operational flow of the various modules composing TAPAS-360°.

3.2 Parser360

The main task of this module is to retrieve and store information about the video manifest. It performs the parsing operation required for the particular streaming standard (HLS or MPEG-DASH) in use. `Parser360` populates and keeps updated the `playlists` data structure. In this data structure, each segment is identified by the relative URI. Moreover, the piece of information about the particular *level* (or representation), together with the respective parameters such as resolution and bitrate, are stored to allow the required control actions being performed. In the case of immersive content, additional information about the possibly different viewpoint representations is also stored. At the end of the update process, the `playlists` data structure is passed to the `Tapas360Player` module for updating the `feedback` dictionary: in the case of *live streaming*, the `playlists` data structure is continuously updated, while in the case of *video on demand* (VoD) the data structure is populated once at the startup.

The implementation of a parser requires the extension of the `BaseParser360` class and the definition of two methods: `start()`, which retrieves and analyzes the manifest to populate the `playlists` data structure and `updateSegmentsList()`, which keeps updated the `playlists` structure.

3.3 MediaEngine

`MediaEngine` is the module dealing with the management of the playback operation. In details, it is responsible for maintaining the playout buffer, providing optional features for the possible decoding and rendering of the video stream. `BaseMediaEngine` provides the skeleton class defining the following methods: `start()`, `stop()`, `pushData()` and `getQueuedTime()`.

Going into detail, the `start()` method initializes the playout buffer and the other components required by the specific `MediaEngine` implementation. `MediaEngine` allows the configuration of the parameter `min_queue_time` that is the minimum duration of video stored in the buffer to start the video playback. Each time a video segment download is completed, the `pushData()` callback is called and the segment is pushed to the playout buffer.

The `getQueuedTime()` method allows to read the length of the playout buffer measured in seconds. This information is useful for

the ABR controller. The `Tapas360Player` module uses this method to allow updating the feedback dictionary.

The specific implementation of `MediaEngine` must extend `BaseMediaEngine`. This way, different logics for draining the buffer, decoding and playing the video stream can be defined. To allow different degrees of simulation details, two multimedia engines have been implemented in TAPAS-360°: `GstMediaEngine` and `FakeMediaEngine`. `FakeMediaEngine` emulates the player status by tracking the length of the playout buffer based on the information contained in the incoming segments without demuxing nor decoding the received video stream. Instead, `GstMediaEngine` provides a complete multimedia engine. Based on *GStreamer 1.0*[2] multimedia framework, it is able to manage both *fMP4* and *ts* media formats, granting compatibility with HLS and MPEG-DASH streaming standards. Moreover, it can work into two modes: `nodec` mode, only demuxing the incoming stream flow; `dec` mode, with video stream decoding and rendering capabilities.

Both `FakeMediaEngine` and `GstMediaEngine` modules allow to disable the video decoding process to keep CPU and memory usage low while perfectly emulating the dynamics of the playout buffer, therefore having the same overall system dynamics as in the case where the received video stream is decoded and rendered. This is a key feature that enables to experimentally study the performance of the video distribution system as the number of streams that share the same bottleneck varies. Moreover, `GstMediaEngine` in `dec` mode can decode, render and possibly store the rendered video stream on the filesystem (see Section 4.3).

3.4 QualityController

The `QualityController` is the module responsible for implementing the ABR algorithm. Its goal is to decide, based on feedback information such as the estimated bandwidth, the length of the playout buffer, and the status of the player, which video representation to download from those listed in the *manifest*.

`BaseQualityController` provides the interface class that a `QualityController` must inherit. Important methods that have to be implemented are: 1) `calcControlAction()`, that implements the control logic by calculating the maximum bitrate value that should be downloaded; 2) `isBuffering()`, that checks if the player is currently into downloading (buffering phase) or in *idle* phase (used to insert OFF pauses between the download of two consecutive segments).

To clarify how the `QualityController` module works, the salient logical sequence of operations that `Tapas360Player` implements is reported. `Tapas360Player` maintains a `feedback` dictionary which stores various information such as the length of the playout buffer and the estimated bandwidth. At the end of the download of each segment, `Tapas360Player`, using the `updateSegmentsList()` method exposed by `Parser360` class, updates the `feedback` dictionary and executes `calcControlAction()` to obtain the video level to be used for the download of the next segment and sets the period of inactivity by using `setIdleDuration()`. In particular, `calcControlAction()` returns the maximum bitrate value that can be downloaded based on the information contained in the `feedback` dictionary. This value is then passed to `quantizeRate()` that selects the highest video level index from the possible values contained in the `feedback` dictionary. In its default implementation, the `quantizeRate()` method selects the highest video level lower the bitrate calculated by `calcControlAction()`. Finally, the `isBuffering()` method checks if the system is either buffering or idle. This is a useful method to keep track of the player state and manage rebuffering events. `BaseQualityController` provides a default implementation for this method, returning `True` if the length of the playout buffer is less than a certain threshold, but more advanced mechanisms can be implemented by overloading this method.

[2]https://gstreamer.freedesktop.org/

3.5 ViewController

The `ViewController` is a new component that immersive video streaming systems are required to implement. Its goal is to select the best viewpoint representation according to the position of the user's head which is reported by the HMD device.

The implementation of the `ViewController` needs to extend the `BaseViewController` class and to implement the `getView()` method, which actually defines the viewpoint selection algorithm.

An example implementation of view controller, named `ConvetionalViewController`, is included in the code base which provides a simple control logic that takes as input the current position of the user's head. The `ConventionalViewController` implements the View Selection Algorithm (VSA) described in [7]. In that paper, different viewpoint representations are prepared server-side, each one consisting in a different *Region of Interest* (RoI), the particular region of the video where the visual quality is higher with respect to the other regions. Each viewpoint representation is identified by a URL and correlated to a tuple storing the *identifier* and the yaw angle pointing to the corresponding RoI. The VSA goal is to select the best viewpoint representation based on the current user view direction.

`ConventionalViewController`, in its current implementation, assumes RoIs are centered at 0°, 120° and 240° with a dihedral angle of 120°, resulting in three different *tiles set*. Nevertheless, *Saliency maps* could be used to tailor the selection of the number and position of the RoIs [6]. Notice that we plan to add support for saliency maps to be integrated in the base view controller class soon so that view controllers will be able to readily access this optional information.

The VSA algorithm workflow is described briefly in the following. The `getView()` method returns to `Tapas360Player` the viewpoint representation that the user is currently viewing. At the end of the download of each segment, `Tapas360Player` – through the `getHMDStatus()` method exposed by the `HMDEmulator` class – updates the `feedback` dictionary which also stores the current viewpoint and the angles representing the position of the user's head. Next, it executes the `getView()` method which returns the viewpoint representation to be selected. At this point, the downloader automatically downloads the correct viewpoint representation and the current bitrate representation selected by the `QualityController`.

3.6 HMDEmulator

In TAPAS-360, the design of `ViewController` requires to receive in input the current angular position from an HMD to perform the viewpoint selection strategy. `HMDEmulator` is the module that

emulates the reading of the angles of the user's head position which normally are provided by the HMD device.

`HMDEmulator` implements the `getCurrentViewAngle()` method, which accepts the playback timestamp and returns the angular data of the current position of the user's head. Such information can be also exploited for viewport adaptive control algorithms based on saliency maps. The emulation of the user's head movement is obtained by reading a Comma-Separated Values (CSV) file which contains the angular data relating to the user's head movement at each timestamp of playback. In this way, publicly available datasets such as [1, 6, 8] can be easily used to allow result reproducibility.

4 USE CASES

In this section we describe some of the use cases for which TAPAS-360° has been designed for.

4.1 2D video streaming

The most common use case is the design and the development of new ABR strategies[3]. To this end, only the `BaseController.py` class has to be extended, implementing the control logic in the `calcControlAction()` method. The new ABR algorithm can be added to `play.py` by simply importing it. The command line for testing the algorithm is:

```
$python3 play.py -a [CONTROLLER] -u [URL]
```

where `[CONTROLLER]` is the name of the class containing the algorithm being tested and `[URL]` is the URL indicating the manifest of the testing video. TAPAS-360° will fetch the video segments indicated in `[VIDEO-URL]` as a regular video player would do under the bitrate adaptation algorithm implemented. The `logs/` folder contains a list of subfolders, indexed for streaming session, with all the logs useful for postprocessing and performance evaluation.

4.2 Viewport-adaptive streaming

The most interesting use case is the design and experimental evaluation of *viewport adaptive* algorithms. To this end, the developer can extend only the `BaseViewController.py` class. The `getView()` method implements the viewport adaptive strategy. Similarly to the `QualityController` algorithm, `play.py` has to be modified importing the new class and adding a custom entry into the flags list (if needed). To test the newly implemented algorithm the following command can be used:

```
$python3 play.py -r True -b [VIEWCONTROLLER] -u [URL]
```

where `[VIEWCONTROLLER]` is the name of the class implementing the viewport adaptive algorithm and `[URL]` is the URL indicating the manifest of the testing video. Similarly to the ABR case, TAPAS-360° will perform the viewport adaptation strategy in the same way as a 360° video player would do. This is possible because of the `HMDEmulator` is fed with a trace representing head movement[4] having the format `[time, alpha, beta, gamma]`, where `time` is a timestamp and `alpha`, `beta`, `gamma` are the three components of the Euler angles. Custom HMD traces can be used in TAPAS-360° by employing the `--hmd_trace` flag. Also in this case useful logs are available into the `logs/` folder.

4.3 Subjective and Objective Quality of Experience evaluations

TAPAS-360° allows to store the fetched segments by simply adding the option `--save_chunks True`. The list of the segments is stored in the subfolder corresponding to the streaming session under the `logs/` folder. This allows to produce video streams that can be used, together with the video streaming log, to run subjective and objective QoE evaluations. To this end, TAPAS-360° could be used to produce a number of "distorted" videos in response to both time-varying network bandwidths (implemented with tools such as Mahimahi) and head movements, by feeding TAPAS-360° traces from datasets such as [1, 6, 8].

5 CONCLUSIONS

This work presents TAPAS-360°, an open-source tool that enables designing and experimenting omnidirectional video streaming algorithms. The tool allows a fine grained control of the decoding process to significantly decrease the CPU load and enable experimenting with several flows being consumed on a single machine. TAPAS-360° includes extensible modules to experiment with viewport adaptive algorithms and to emulate HMD devices using head movements datasets. The ambition is to attract the research community to contribute with their algorithms and make TAPAS-360° an open platform facilitating results reproducibility within the multimedia community.

ACKNOWLEDGMENT

This work has been partially supported by the Italian Ministry of Economic Development (MISE) through the CLIPS project (no. F/050136/01/X32). Any opinions, findings, conclusions or recommendations expressed in this work are the authors' and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Xavier Corbillon, Francesca De Simone, and Gwendal Simon. 2017. 360-Degree Video Head Movement Dataset. In *Proc. ACM MMSys '17*. 199–204.
[2] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. 2014. TAPAS: a Tool for rApid Prototyping of Adaptive Streaming algorithms. In *Proc. Workshop on Design, Quality and Deployment of Adaptive Video Streaming*. 1–6.
[3] J. He, M. Adnan Qureshi, L. Qiu, J. Li, F. Li, and L. Han. 2018. Rubiks: Practical 360-degree streaming for smartphones. In *Proc. ACM MobiSys '18*. 482–494.
[4] O. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S.Y. Lim. 2016. MPEG DASH SRD: spatial relationship description. In *Proc. ACM MMSys '16*. 1–8.
[5] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proc. of ACM MobiCom '18*. 99–114.
[6] Y. Rai, J. Gutiérrez, and P. Le Callet. 2017. A dataset of head and eye movements for 360 degree images. In *Proc. ACM MMSys '17*. 205–210.
[7] Giuseppe Ribezzo, Giuseppe Samela, Vittorio Palmisano, Luca De Cicco, and Saverio Mascolo. 2018. A DASH Video Streaming System for Immersive Contents. In *Proc. ACM MMSys '18*. 525–528.
[8] Silvia Rossi, Cagri Ozcinar, Aljosa Smolic, and Laura Toni. 2020. Do Users Behave Similarly in VR? Investigation of the User Influence on the System Design. *ACM Trans. Multimedia Comput. Commun. Appl.* 16, 2, Article 46 (May 2020), 26 pages.
[9] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. 2017. 360probdash: Improving QoE of 360 video streaming using tile-based http adaptive streaming. In *Pro. ACM MM '17*. 315–323.
[10] A. Zabrovskiy, E. Kuzmin, E. Petrov, C. Timmerer, and C. Mueller. 2017. Advise: Adaptive video streaming evaluation framework for the automated testing of media players. In *Proc. ACM MMSys '17*. 217–220.

[3] Notice that this use case was already possible with TAPAS [2], but we mention it for the sake of completeness.
[4] We provide a default example trace in the repository named `hmd_trace.csv`.