SMART360: Simulating Motion prediction and Adaptive bitRate sTrategies for 360° video streaming

Quentin Guimard, Lucile Sassatelli

SMART360: Simulating Motion prediction and Adaptive bitRate sTrategies for 360° video streaming

Quentin Guimard
quentin.guimard@univ-cotedazur.fr
Université Côte d'Azur, CNRS, I3S
Sophia-Antipolis, France

Lucile Sassatelli
Université Côte d'Azur, CNRS, I3S
Institut Universitaire de France
Sophia-Antipolis, France

ABSTRACT

Adaptive bitrate (ABR) algorithms are used in streaming media to adjust video or audio quality based on the viewer's network conditions to provide a smooth playback experience. With the rise of virtual reality (VR) headsets, 360° video streaming is growing rapidly and requires efficient ABR strategies to also adapt the video quality to the user's head position. However, research in this field is often difficult to compare due to a lack of reproducible simulations. To address this problem, we provide SMART360, a 360° streaming simulation environment to compare motion prediction and adaptive bitrates strategies. We provide sample inputs and baseline algorithms along with the simulator, as well as examples of results and visualizations that can be obtained with SMART360. The code and data are made publicly available.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Simulation environments**; • **Human-centered computing** → **Virtual reality**; • **Networks** → *Network simulations*.

KEYWORDS

360° videos, bitrate adaptation, streaming simulations

ACM Reference Format:
Quentin Guimard and Lucile Sassatelli. 2023. SMART360: Simulating Motion prediction and Adaptive bitRate sTrategies for 360° video streaming. In *Proceedings of the 14th ACM Multimedia Systems Conference (MMSys '23), June 7–10, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587819.3592547

1 INTRODUCTION

Adaptive bitrate (ABR) algorithms are used in streaming media to adjust the quality of a video or audio stream in real time based on the viewer's network conditions. The goal of ABR streaming is to provide a smooth playback experience by adapting the bitrate of the stream to match the viewer's available bandwidth.

With the recent rise in popularity and increasing affordability of virtual reality (VR) headsets, 360° video streaming is growing rapidly, but requires even higher bitrates, which makes the use of efficient streaming algorithms even more critical. ABR algorithms for 360° video streaming can even go one step further than regular ABR algorithms, as the video quality can be dynamically adapted to send higher quality content inside the user's viewport. This involves predicting the head and gaze movements of the user several seconds in advance to build a buffer that can be resilient to bandwidth variations.

Adaptive streaming of 360° videos has been the subject of many research works in the past few years [2, 7, 8]. While these works present new approaches and methods to improve the quality of experience (QoE) or bandwidth usage of adaptive streaming systems, it is often difficult to compare them fairly, as the code for simulating them is not always provided.

We present SMART360, a 360° streaming simulation environment that can be used to compare head motion prediction and ABR algorithms. Our contributions are the following:

- we provide a new simulator[1], equipped with large datasets and baseline algorithms that builds upon the existing solutions, with explanations about the code structure and logic;
- we also make the preprocessing pipeline available[2] for transparency and to give the ability to easily create new input configurations for the simulator;
- we explain in detail how SMART360 can be used by researchers to implement and compare existing and new motion prediction and adaptive bitrate strategies and show examples of metrics and visualizations that can readily be used to evaluate their algorithms.

2 RELATED WORK

As a result of the lack of reproducible simulations for most of the 360° adaptive streaming research, several tools have been made available in recent years in an effort to improve reproducibility in this field.

Ribezzo et al. [9] released TAPAS-360° [3], an open-source emulator that enables designing and experimenting omnidirectional video streaming algorithms. Unfortunately, TAPAS-360° does not support tile-based streaming, but works with a set of pre-defined "views". This makes it impossible to use with tile-based bitrate adaptation algorithms, which are the most common type of bitrate adaptation algorithms for 360° video streaming.

Spiteri [12] released Sabre360[4], a simulation testbed for 360° videos as an extension of Sabre[5] [13], an open-source simulation

MMSys '23, June 7–10, 2023, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0148-1/23/06…$15.00
https://doi.org/10.1145/3587819.3592547

[1] https://gitlab.com/SMART360/SMART360-simulator
[2] https://gitlab.com/SMART360/SMART360-preprocessing
[3] https://github.com/c3lab/tapas360
[4] https://github.com/UMass-LIDS/sabre360
[5] https://github.com/UMass-LIDS/sabre

environment for ABR algorithms. While Sabre360 can be used to compare adaptive bitrate algorithms, it has some drawbacks: (i) it does not implement stalls, but plays "blank tiles" instead, (ii) it is built around a "view" system that only supports one kind of tiling layout (4x4 tiles), and (iii) the ABR optimization for quality allocation is done between each tile download, and makes an individual request for each tile of each segment, which is not realistic.

Jiang et al. [6] provide code for simulating 360° bitrate adaptation and motion prediction along with Plato[6], but the lack of documentation and obscure file structure makes it difficult to use, precluding other researchers from using it and test new algorithms.

Finally, Chopra, Chakraborty, et al. [1] provide the code for PARIMA[7], which allows to test and compare their model to some baselines with QoE metrics, but it is not a streaming simulation since it does not consider network aspects.

Our simulator takes a lot of inspiration from Sabre360, which we consider to be the closest solution to the problem we want to solve. Our work aims at rectifying any shortcomings the existing solutions may have for comparing motion prediction and adaptive bitrate strategies in the context of 360° streaming.

3 DATA PREPROCESSING

The objective of SMART360 is to provide a simulation environment that enables the comparison of ABR and viewport prediction algorithms when streaming 360° videos with network constraints. In this section, we describe the necessary inputs the simulator needs to perform this task, as well as the preprocessing pipeline the data undergoes before being used by the simulator.

3.1 Simulator inputs

All the input data for the SMART360 simulator is provided in the `config/` directory of the simulator repository. The input data uses the same JSON format as Sabre360 [12]. The data provided in the `SMART360-simulator/config/` directory is split in two types: *real* and *synthetic* data. The *real* data is extracted from multiple public datasets and is described in the following subsections. The *synthetic* data contains simple cases of network traces with constant bandwidth and manifests describing uniformly-sized 360° videos. The user head motion traces found in the *synthetic* directory are copied from *real* data.

3.1.1 Network traces. The network traces describe the available bandwidth and the latency over time in different situations. They allow for realistic simulations where the bandwidth is highly variable. The network traces provided in the *real* input data are the same as the ones used in Sabre360, and come from the 4G/LTE dataset published by van der Hooft et al. [14]. They are made of 40 traces of bandwidth measurements along several routes in the city of Ghent, Belgium.

For the comparisons between ABR algorithms to be relevant, we need to be in a situation where the algorithm has to adapt to the network constraints. On the one hand, if the bandwidth is very high relative to the video bitrate, there is no need for ABR streaming, as we can just download everything in the highest quality without any rebuffering (stall) event. On the other hand, if the bandwidth is very low relative to the video bitrate, ABR streaming is not so useful either, as we can only download everything in the lowest quality. To make for a relevant comparison between ABR algorithms, we provide a *Jupyter* notebook to scale the network traces relatively to the video bitrates, as illustrated in Fig. 1. This notebook is available in the `SMART360-simulator/notebooks/` directory.

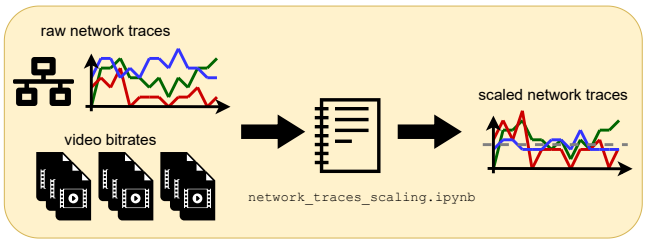

Figure 1: Network trace scaling principle.

3.1.2 User head motion traces. The user head motion traces describe the behavior of people watching 360° videos. They contain the coordinates of the head orientation over time. This allows calculating which tiles are visible to the user at any given time during the video. We provide 3518 head motion traces from users watching 94 different videos, extracted from three of the datasets used by Romero et al. [11] in their framework to evaluate head motion prediction methods in 360° videos[8]. The traces have a 5 Hz sampling rate and use a 3D Cartesian coordinate system, where the orientation of the head is represented as a point on the unit sphere. We provide a *Python* script, available in the `SMART360-preprocessing/` root directory to convert the traces from their original CSV format to a JSON format similar to the one used in Sabre360.

3.1.3 Video manifests. The video manifests describe the video files to be streamed over the Internet. In the case of 360° tiled videos, the manifests describe the tiling layout and the different quality levels of encoding. The SMART360 simulator uses the video manifest to get the size of each downloaded tile. We provide simplified JSON video manifests for the 94 videos mentioned in Sec. 3.1.2 in the same format as the one used in Sabre360. We detail the preprocessing steps to obtain the video manifests from MP4 video files in the next subsection.

3.2 Preprocessing pipeline

The preprocessing pipeline is based on TOUCAN-preprocessing[9], a *Java* command line application to convert a regular 360° videos into DASH-SRD described videos, using *FFmpeg* and *MP4Box*, released by Dambra et al. [3]. We have made some changes to simplify the original pipeline, update the encoding parameters, and adapt the input and output formats to our problem. The preprocessing pipeline is described in Fig. 2 and detailed in the following subsections.

3.2.1 Video tiling and re-encoding. First, the MP4 videos are split into tiles using the *FFmpeg* crop filter. Since cropping the videos requires re-encoding them, we choose to re-encode the video

[6] https://github.com/federerjiang/Plato
[7] https://github.com/sarthak-chakraborty/PARIMA
[8] https://gitlab.com/miguelfromeror/head-motion-prediction
[9] https://github.com/UCA4SVR/TOUCAN-preprocessing

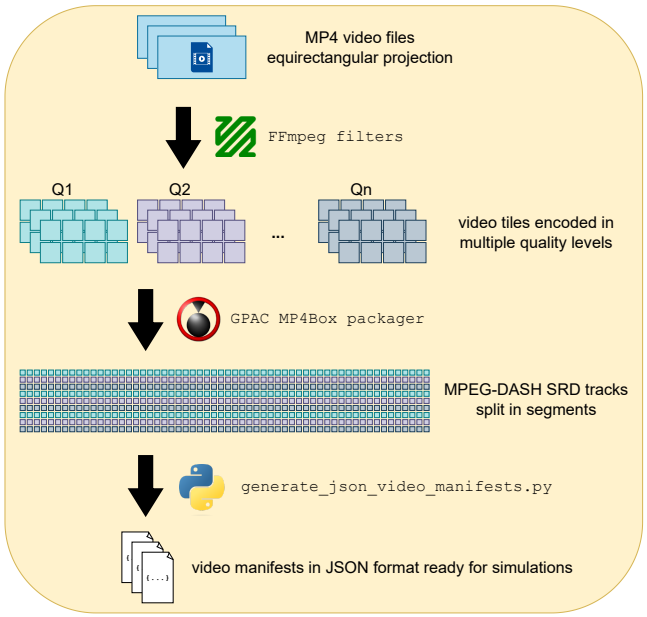

Figure 2: SMART360 video preprocessing pipeline.

tiles in different quality levels while tiling them. The tiling layouts and quality levels are configurable settings that can be specified in an XML file for each video.

The videos are re-encoded with *libx265*, using the HEVC compression standard. Different quality levels are achieved using different constant rate factors (CRFs). CRF is a method of video compression that is designed to maintain a constant level of perceived quality, as opposed to constant bitrate (CBR) encoding, but similar to using a constant quantization parameter (CQP). Unlike CQP, CRF adjusts the QP to compress different frames by varying amounts by taking motion into account. For high-motion frames, the QP is increased to compress the frame more, and for low-motion frames, the QP is lowered to reduce compression. This leads to a varying bitrate allocation over time, resulting in a more efficient use of the available bandwidth. While constant bitrate and constrained CRF may be better suited for streaming to avoid bitrate variations, CRF is better suited than CQP (the most popular encoding mode to compare adaptive bitrate strategies in 360° videos [15]), as it results in a more constant bitrate [10]. CRF was chosen as the best compromise between bitrate stability (better than CQP), efficiency (not wasting bits like constant bitrate), and encoding time (not needing multiple passes like constrained CRF).

3.2.2 DASH packaging. Once the videos are cropped into the desired tiling layouts and encoded in the appropriate quality levels, we use the *MP4Box* multimedia packager to obtain a DASH-SRD compliant video split in segments. The segment duration is also a configurable parameter that can be specified in the same XML file as mentioned in Sec. 3.2.1. The output files generated by this preprocessing step are MP4 tracks and an XML manifest for each video, which correspond to the files that can be streamed over the Internet.

3.2.3 JSON file generation. Finally, we provide a *Python* script, available in the `SMART360-preprocessing/` root directory to build the JSON manifests that can be used by the simulator. This script simply reads the files that were previously generated and keeps only the information that is relevant for the simulations to put them in the JSON video manifests described in 3.1.3.

4 SIMULATOR ARCHITECTURE

The SMART360 simulator architecture is based on the architecture of the Sabre360 simulator, with substantial differences. The changes mainly aim at rectifying the shortcomings formulated in Sec. 2, namely: (i) introducing actual stall events that pause the video playback instead of playing blank tiles, (ii) re-thinking the coordinate system and modifying the headset model to support any rectangular tiling layout, and (iii) re-designing the simulator and ABR logic, enabling the planning of quality allocation for multiple tiles and segments in advance.

4.1 File and object structure

We present a simplified class diagram in Fig. 3, where we choose to only keep the relevant attributes and methods of the SMART360 simulator. The classes are separated in multiple files located at `SMART360-simulator/simulator/`. The classes highlighted in red in the diagram, *BandwidthEstimator*, *TiledABR*, and *ViewportPredictor* are classes that can be easily extended to implement new algorithms. We detail the file structure and classes of the simulator in the following subsections.

4.1.1 Session. The `session.py` file contains the *Session* and *SessionInfo* classes. The *Session* class is the main class that contains all the objects necessary to the simulation. The *Session::run* method is the entry point of the simulator and is described in Algo. 1. The *SessionInfo* class is mainly used to access information and objects like the buffer, log file, or viewport predictor from other objects.

4.1.2 Buffer. The `buffer.py` file contains the *TiledBuffer* class. This class contains the buffer in the form of a two-dimensional *NumPy* array of size $B \times T$, where B is the buffer size (in number of segments) and T is the number of tiles in the video. This class also provides methods to update the buffer.

4.1.3 Headset. The `headset.py` file contains the *HeadsetModel* and *HeadsetConfig* classes. These classes contain information about the headset configuration (tile layout, FoV size) and provide methods to calculate which tiles are visible, given the user's head coordinates. Unlike most existing tools, the tile calculation considers the distortion produced by the equirectangular projection. The headset configuration is loaded from JSON file located in the `SMART360-simulator/config/` directory.

4.1.4 User. The `user.py` file contains the *UserModel* class. This class handles the user head motion trace and is used to get head motion coordinates updates.

4.1.5 Network. The `network.py` file contains the *NetworkModel* class. This class handles the network trace and provides methods to download groups of tiles in compliance with the bandwidth and latency information present in the network trace.

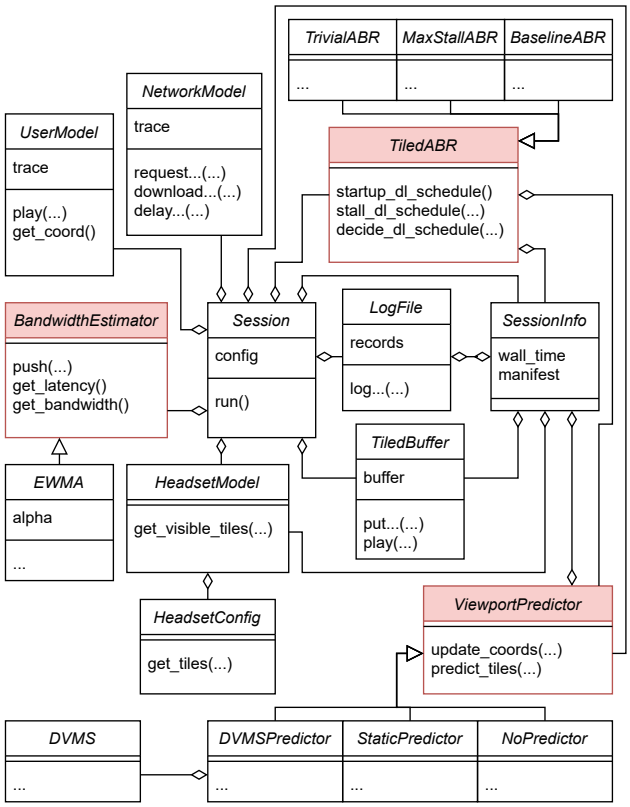

Figure 3: UML class diagram of the SMART360 simulator. All aggregation relationships are one-to-one.

4.1.6 Bitrate adaptation. The `br_adaptation.py` file contains the *TiledABR* abstract class and its subclasses. This class is responsible for deciding which tiles of which segments will be downloaded in which quality, and in which order. We provide three simple ABR strategies with no buffer replacements. *TrivialABR* tries to download all tiles in the lowest quality and fills the buffer as quickly as possible. *MaxStallABR* is provided for experimental purposes to calculate the maximum possible stall ratio for a user, in the case where we only download tiles once they are missing in the viewport and causing a stall event. We also provide *BaselineABR*, a simple ABR strategy with some rate-based and buffer-based elements.

4.1.7 Viewport prediction. The `vp_prediction.py` file contains the *ViewportPredictor* abstract class and its subclasses. This class is used to make predictions about user head movements and the resulting viewports. These predictions can in turn be used by the ABR algorithms. We provide two baseline viewport predictors as well as an implementation of a deep learning predictor. *NoPredictor* gives equal probabilities for all tiles. *StaticPredictor* assumes the user will not move and gives higher probabilities to tiles that were inside the viewport. *DVMSPredictor* uses the DVMS deep learning model from Guimard et al. [4] to make predictions.

4.1.8 Bandwidth estimation. The `bw_estimation.py` file contains the *BandwidthEstimator* abstract class and its *EWMA* subclass. This class can be used to make estimates of the future bandwidth and latency of the network, useful for ABR planning. The *EWMA* subclass makes latency and bandwidth estimates following an exponentially weighted moving average model, as done in the *dash.js* reference player[10], but in a simplified manner.

4.1.9 Logging. The `_logging.py` file contains the *LogFile* class. This class provides methods to add simulation information and measurements to a list of records, that is then written to a JSON log file at the end of the simulation. New methods can easily be implemented to include more information and measurements.

4.1.10 Log parsing. The `parse_session_logs.py` file, located in the `log_parsing/` directory, consists of a post-processing pipeline that reads the JSON log file and builds data frames stored in *Feather* files. This file format produces very lightweight files that are quick to read and write compared to the raw log files.

4.1.11 Notebooks. There are two notebooks in the `notebooks/` directory. The `network_traces_scaling.ipynb` notebook is described in Sec. 3.1.1, and the `output_metrics.ipynb` notebook gives examples of possible visualizations of the SMART360 output metrics, as shown in Sec. 5.3.

4.2 Simulator logic

In this section, we describe the algorithmic flow of the SMART360 simulator. As mentioned in Sec. 4.1.1, the *Session::run* method is the entry point to the simulator. We describe the logic behind this method in Algo. 1. For the sake of readability, the algorithms described in Algo. 1 and Algo. 2 are simplified versions of the methods, where only the most relevant steps are shown.

The three ABR functions that appear on lines 3 and 9 of Algo. 1, and line 11 of Algo. 2 refer to the three methods of the *TiledABR* abstract class that have to be implemented by subclasses, as explained in Sec. 5.1. These functions return download schedules, noted skd. A download schedule is an ordered list of elements that each contain s, the segment number, t, the tile number, and q, the quality level. In the case of startup and stall (lines 3 of Algo. 1 and line 11 of Algo. 2), the full schedule must be downloaded before the video playback can be resumed. In the case of the regular ABR decision function (line 9 of Algo. 1), elements are downloaded in the same order as given by the schedule during Δ_{DL} seconds.

On line 5 of Algo. 1 and lines 13 and 17 of Algo. 2, "download" implies using the *NetworkModel* with the appropriate latency and bandwidth, as well as putting the downloaded tiles in the buffer.

In Algo. 2, we give some detail behind the logic of one the most complex methods of the simulator, *Session::play_and_download*. This method enables the simulation of video playback and tile download at the same time, while also making sure that the *NetworkModel* and the *UserModel* stay synchronized. This method brings two improvements over Sabre360:

- the ABR algorithm has to plan and make individual requests for downloading groups of tiles every Δ_{DL} seconds, which is

[10] https://github.com/Dash-Industry-Forum/dash.js

Algorithm 1 Simplified `run` method

```
l ← video length
p ← 0                                          ▷ video play head
skd_startup ← ABR_STARTUP                      ▷ startup schedule
for all s, t, q in skd_startup do
    download tile t from segment s in quality q
end for
while p < l do
    bw_est ← network bandwidth estimation
    skd ← ABR_DECIDE(bw_est, Δ_DL)             ▷ download schedule
    PLAY_AND_DOWNLOAD(skd, Δ_DL)               ▷ see Algo. 2
end while
```

more realistic than the very frequent ABR optimizations and requests in Sabre360;

- stall events can happen and stall periods can be measured, which we also consider more realistic than the video not pausing and showing blank tiles in Sabre360. We have chosen for stall events to happen in SMART360 only if a tile that should be visible to the user is not present in the buffer. This means that the video does not stop if tiles are missing from the buffer but are not in the user's field of view.

Algorithm 2 Simplified `play_and_download` method

```
procedure PLAY_AND_DOWNLOAD(skd, Δ_DL)
    Δ_left ← Δ_DL
    while Δ_left > 0 do
        τ_coord ← time until next user coord. update
        τ_segment ← time until next video segment
        τ ← min(Δ_left, τ_coord, τ_segment)
        T_buf ← set of tiles in buffer for current segment
        T_visible ← set of visible tiles calculated from coord.
        T_missing ← T_visible − T_buf ∩ T_visible
        if T_missing ≠ ∅ then                  ▷ stall event
            skd_stall ← ABR_STALL(T_missing)   ▷ stall schedule
            for all s, t, q in skd_startup do
                download tile t from segment s in quality q
            end for
        end if
        if |skd| > 0 then
            download (s, t, q) schedule elements for τ seconds
            remove downloaded elements from skd
        end if
        p ← p + τ
    end while
end procedure
```

5 USING SMART360 TO COMPARE MOTION PREDICTORS AND ADAPTIVE BITRATE ALGORITHMS

In this section, we explain how researchers can use the SMART360 simulation environment to implement new ABR strategies and motion prediction algorithms for 360° video streaming and compare them.

5.1 Implementing an ABR strategy within SMART360

To implement a new ABR strategy, one only needs to create a new subclass of *TiledABR* (see Sec. 4.1.6) that implements three methods. Each one of these methods returns a download schedule containing elements composed of s, the segment number, t, the tile number, and q, the quality level. In addition to the method parameters, the ABR class can access other information such as the buffer content, the video manifest, or a viewport predictor.

- *startup_dl_schedule()* is called at the beginning of the simulation. It must return a schedule of what to download before the video playback starts;
- *decide_dl_schedule(bw_{est}, Δ_{DL})* is the main ABR decision method. It is called every Δ_{DL} seconds and must return a schedule of what to download in the next Δ_{DL} seconds, given the estimated bandwidth;
- *stall_dl_schedule($\mathcal{T}_{missing}$)* is called whenever a stall event happens. When the video playback is paused during this event, the list of missing tiles in the user's field of view is passed as a parameter and the method must return a schedule of what to download. The video playback can only resume if all the missing tiles and everything in the stall schedule has been downloaded.

5.2 Implementing a motion predictor within SMART360

SMART360 also allows the implementation of head motion prediction algorithms, in the form of a viewport predictor that can in turn be used by the ABR algorithm. To implement a new motion predictor, one only needs to create a new subclass of *ViewportPredictor* (see Sec. 4.1.7) that can implement two methods:

- *predict_tiles(s)* has to be implemented by the subclass. The parameter s corresponds to the segment number for which we want to make predictions. This method returns a list of length T, where each element corresponds to the score given to each tile. A higher score means a higher probability of being present in the user's viewport during segment s;
- *update_coord(coord)* can be implemented, but is not mandatory. This method allows updating the motion predictor with new head coordinates that can be used to make predictions.

As of right now, the only information that can be used for predictions is the past head coordinates of the user. However, SMART360 could easily be extended to include video information such as saliency maps for head motion prediction.

5.3 SMART360 output metrics

SMART360 brings many QoE-related metrics and visualizations, as well as some network-related metrics. The logs that we provide already enable numerous types of insightful visualizations, as shown in Fig. 4, and can easily be extended to include more information and measurements. In this section, we show examples of figures that can be produced with SMART360 to compare ABR and head motion prediction algorithms. The figures presented in this section are extracted from the `output_metrics.ipynb` notebook, and

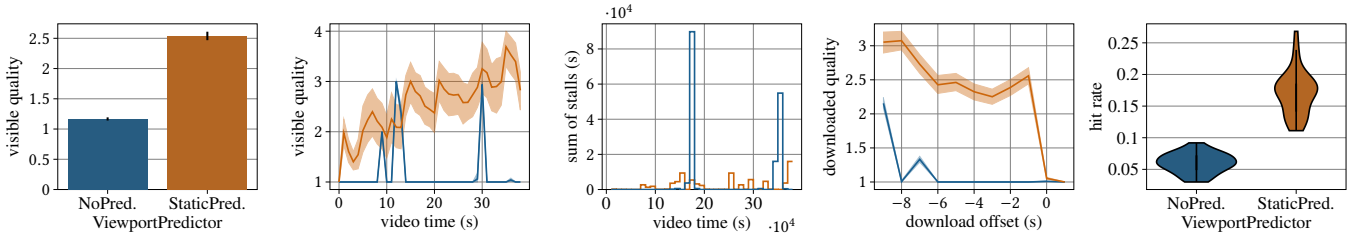

Figure 4: Some examples of visualizations from SMART360 simulation output metrics. From left to right: (a) average visible quality, (b) average visible quality against video timestamp, (c) sum of all user stalls against video timestamp, (d) average downloaded quality against download offset, (e) bandwidth efficiency. Colors have the same meaning across all subfigures.

new visualizations can readily be generated from the same data frames without needing to extend the logs.

• Fig. 4a compares the **average visible quality** when using two different viewport predictors over one video for all users who have watched this video. In this example, there are five quality levels ranging from 1 to 5, and more details about the simulation settings can be found in the notebook. Here, we can see that the *StaticPredictor* gives higher average visible quality than *NoPredictor*.

• Fig. 4b compares the **average visible quality** when using two different viewport predictors **against** the **video timestamp** for all users who have watched this video. The quality levels and simulation settings are the same as in Fig. 4a. Here, we can see with more detail when, in the video, *StaticPredictor* gives higher average visible quality than *NoPredictor*.

• Fig. 4c compares the sum of **stall periods** when using two different viewport predictors **against** the **video timestamp** for all users who have watched this video. The simulation settings are the same as in Fig. 4a. Here, we can see with precision exactly when, in the video, the stalls are happening, and that *StaticPredictor* gives fewer stall periods than *NoPredictor*.

• Fig. 4d compares the **average quality** of downloaded tiles that end up in the user's viewport when using two different viewport predictors **against** the "**download offset**" for all users who have watched this video. The download offset is inversely proportional to the buffer level: a download offset of -6 means that the tile was downloaded 6 seconds before it was played. The quality levels and simulation settings are the same as in Fig. 4a. Here, we can understand better how the ABR strategy works and how the prediction impacts its behavior regarding the buffer level, and that *StaticPredictor* gives higher average visible quality than *NoPredictor*, regardless of the buffer level.

• Fig. 4e compares the **distribution** of the "**hit rate**" when using two different viewport predictors over one video for all users who have watched this video. The hit rate is calculated by dividing the number of bits that appeared in the user's viewport by the total number of bits that were downloaded, it can be seen as a form of bandwidth efficiency. The simulation settings are the same as in Fig. 4a. Here, we can see that *StaticPredictor* is more efficient than *NoPredictor* and wastes less bandwidth.

The notebook also includes metrics on the spatial and temporal quality variance of 360° videos, as well fairness metrics based on the QoE fairness index described by Hoßfeld et al. [5].

6 DISCUSSION

We believe that SMART360 successfully addresses the shortcomings of the existing solutions to realistically simulate 360° streaming systems and efficiently compare ABR and head motion prediction algorithms. With SMART360, we yearn to encourage reproducible research by providing transparent code that can be adapted and improved.

Possible improvements include but are not limited to: considering the percentage of each tile actually in the viewport for more accurate measurements of the visible quality, instead of counting them as inside the viewport regardless of the proportion of the tile actually in the viewport; giving the ability to the viewport predictor to use more information than the past head coordinates, such as video saliency maps; using multiple threads and communication between threads when events occur during the simulation instead of the monolithic structure of the *Session::play_and_download* method.

7 CONCLUSION

In this article, we have presented SMART360, a new simulation environment for 360° video streaming that allows comparing different motion prediction and adaptive bitrate strategies with numerous metrics and graphical visualizations.

SMART360 overcomes the drawbacks of the few existing alternative tools by providing highly-configurable code, with many inputs and settings, as well as offering a realistic streaming behavior, with stall events and ABR planning.

We have described the inputs and outputs of the simulator, as well as its internal structure. We have explained how new motion predictors and adaptive bitrate algorithms can be implemented inside the simulation environment to be evaluated and compared.

We believe that SMART360 can improve the reproducibility of research regarding 360° video motion prediction and adaptive streaming algorithms, and make future comparisons of new strategies easier for researchers.

ACKNOWLEDGMENTS

This work has been partly supported by the French government, through the UCA JEDI and EUR DS4H Investments in the Future projects ANR-15-IDEX-0001 and ANR-17-EURE-0004. This work was partly supported by EU Horizon 2020 project AI4Media, under contract no. 951911 (https://ai4media.eu/).

REFERENCES

[1] Lovish Chopra, Sarthak Chakraborty, Abhijit Mondal, and Sandip Chakraborty. 2021. PARIMA: Viewport Adaptive 360-Degree Video Streaming. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery, New York, NY, USA, 2379–2391. https://doi.org/10.1145/3442381.3450070

[2] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *2017 IEEE International Conference on Communications (ICC)*. 1–7. https://doi.org/10.1109/ICC.2017.7996611

[3] Savino Dambra, Giuseppe Samela, Lucile Sassatelli, Romaric Pighetti, Ramon Aparicio-Pardo, and Anne-Marie Pinna-Déry. 2018. Film Editing: New Levers to Improve VR Streaming. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 27–39. https://doi.org/10.1145/3204949.3204962

[4] Quentin Guimard, Lucile Sassatelli, Francesco Marchetti, Federico Becattini, Lorenzo Seidenari, and Alberto Del Bimbo. 2022. Deep Variational Learning for Multiple Trajectory Prediction of 360° Head Movements. In *Proceedings of the 13th ACM Multimedia Systems Conference (MMSys '22)*. Association for Computing Machinery, New York, NY, USA, 12–26. https://doi.org/10.1145/3524273.3528176

[5] Tobias Hoßfeld, Lea Skorin-Kapov, Poul E. Heegaard, and Martin Varela. 2017. Definition of QoE Fairness in Shared Systems. *IEEE Communications Letters* 21, 1 (2017), 184–187. https://doi.org/10.1109/LCOMM.2016.2616342

[6] Xiaolan Jiang, Yi-Han Chiang, Yang Zhao, and Yusheng Ji. 2018. Plato: Learning-based Adaptive Streaming of 360-Degree Videos. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. 393–400. https://doi.org/10.1109/LCN.2018.8638092

[7] Sohee Park, Minh Hoai, Arani Bhattacharya, and Samir R. Das. 2021. Adaptive Streaming of 360-Degree Videos With Reinforcement Learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 1839–1848.

[8] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. 2016. Optimizing 360 Video Delivery over Cellular Networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges (ATC '16)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/2980055.2980056

[9] Giuseppe Ribezzo, Luca De Cicco, Vittorio Palmisano, and Saverio Mascolo. 2020. TAPAS-360°: A Tool for the Design and Experimental Evaluation of 360° Video Streaming Systems. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20)*. Association for Computing Machinery, New York, NY, USA, 4477–4480. https://doi.org/10.1145/3394171.3414541

[10] Werner Robitza. 2019. CRF Guide (Constant Rate Factor in x264, x265 and libvpx). https://slhck.info/video/2017/02/24/crf-guide.html.

[11] Miguel Fabián Romero Rondón, Lucile Sassatelli, Ramón Aparicio-Pardo, and Frédéric Precioso. 2020. A Unified Evaluation Framework for Head Motion Prediction Methods in 360° Videos. In *Proceedings of the 11th ACM Multimedia Systems Conference (MMSys '20)*. Association for Computing Machinery, New York, NY, USA, 279–284. https://doi.org/10.1145/3339825.3394934

[12] Kevin Spiteri. 2021. *Video Adaptation for High-Quality Content Delivery*. Ph.D. Dissertation. https://doi.org/10.7275/20604181

[13] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 123–137. https://doi.org/10.1145/3204949.3204953

[14] Jeroen van der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. 2016. HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks. *IEEE Communications Letters* 20, 11 (2016), 2177–2180. https://doi.org/10.1109/LCOMM.2016.2601087

[15] Abid Yaqoob, Ting Bi, and Gabriel-Miro Muntean. 2020. A Survey on Adaptive 360° Video Streaming: Solutions, Challenges and Opportunities. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2801–2838. https://doi.org/10.1109/COMST.2020.3006999