BAS-360°: Exploring Spatial and Temporal Adaptability in 360-degree Videos over HTTP/2

Mengbai Xiao[*], Chao Zhou[†], Viswanathan Swaminathan[‡], Yao Liu[†] and Songqing Chen[*]
[*]Department of Computer Science, George Mason University, {mxiao3, sqchen}@gmu.edu
[†]Department of Computer Science, SUNY Binghamton, {czhou5, yaoliu}@binghamton.edu
[‡]Adobe Research, Adobe Systems Inc., vishy@adobe.com

***Abstract*—Today, 360-degree video streaming has become a popular Internet service with the rise of affordable virtual reality (VR) technologies. However, streaming 360-degree videos suffers from the prohibitive bandwidth demand. Existing bandwidth-efficient solutions mainly focus on exploiting the inherent spatial adaptability of 360-degree videos, delivering only video content (spatially-cut tiles) in the viewer's region of interest (ROI) with higher quality. Temporal adaptability, which has been widely leveraged in HTTP streaming, has not been well exploited to select proper quality for video segments according to the bandwidth variations. When these two dimensions of adaptability are jointly considered, bitrate selection for the tiles become more complicated and challenging. The importance of a tile with a spatial coordination played at a specific time should be quantified so that we can determine how to allocate bandwidth for improving the viewer's quality of experience. Furthermore, viewer's head orientation prediction is highly variable, which makes the determination of important tiles highly dynamic. In addition, network fluctuations are very common on the Internet.**

To overcome these challenges, we propose Bi-Adaptive Streaming for 360-degree videos (BAS-360°). In BAS-360°, both spatial and temporal adaptabilities are explored in the bitrate selection for different tiles. The objective is to minimize the bandwidth waste by allocating bandwidth to more important tiles (the tiles that are more likely to be watched). To tackle the high variability of visual region prediction and the unpredictable network fluctuations, we employ two features provided by HTTP/2: *stream termination* and *stream priority*, to efficiently organize tile delivery. Evaluation results show that BAS-360° outperforms naive tile-based 360-degree video streaming strategies when network fluctuations or errors in viewport predictions occur.

***Index Terms*—HTTP streaming; 360-degree video streaming; HTTP/2;**

I. Introduction

With the support of the rapidly developing virtual reality (VR) techniques, streaming 360-degree videos is becoming increasingly popular. When watching a 360-degree video, the viewers can freely adjust their field of view (FoV) over the scene rendered surrounding them, providing the viewers with immersive watching experience. Many content providers, such as Facebook [1] and YouTube [2], have started to provide 360-degree videos to the public.

However, a significant challenge of 360-degree video streaming is its prohibitive bandwidth demand. Compared to traditional videos, 360-degree videos usually desire higher resolution due to two reasons: 1) it encodes an omnidirectional scene while a traditional video only encodes scene at a specific direction; 2) compared to the traditional displays, the head-mounted displays (HMDs) that play the 360-degree video are worn much closer to the viewer's eyes. Therefore, the 360-degree videos are commonly encoded with a 4K or 6K resolution for a satisfactory watching experience. According to a Netflix recommendation [3], streaming a 4K video requires at least 25 Mbps, while the average broadband connection in the USA is only 15.3 Mbps, according to another report from Akamai [4]. This significantly constrains the viewers' quality of experience (QoE) when watching 360-degree videos online.

An intuitive solution is to reduce the bitrate of the 360-degree videos. However, lowering the bitrate means degraded visual quality. Instead, a potentially effective bandwidth saving solution without deteriorating the viewers' visual quality is to exploit spatial adaptability. While watching a 360-degree video, the viewer's FoV only covers a portion of the full 360-degree scene. As a result, encoding and delivering the invisible region of the rendered scene at a lower bitrate level can hardly impact the visual quality of the 360-degree video. In particular, a 360-degree video streaming system can project the spherical scenes onto a 2D planar so as to be compatible with the traditional codec, spatially cut the generated rectangular frames into tiles and then only encode and deliver the visible tiles at a high bitrate [5], [6], [7], [8], [9], [10], [11], [12], [13], [14].

Since tiling focuses on the spatial dimension, intuitively bitrate selection for the tiles can be done as follows: By default all tiles are encoded at the lowest bitrate for rendering a complete scene at the client side, while the bitrate of visible tiles are opportunistically promoted if there is extra bandwidth remaining. Furthermore, temporal adaptability can be exploited as segment-based streaming, such as dynamic adaptive streaming over HTTP (DASH) [15]. The quality of a segment can be degraded when there is insufficient bandwidth.

However, in practice, prediction of the visible region (based on viewer's head orientation) in a 360-degree video streaming session is only valid in the near future (< 1s) [14], [16]. At the same time, it is not always possible to predict bandwidth accurately. These make optimal bitrate allocation to tiles complicated and challenging. For instance, if the bandwidth budget allows promoting quality of only a limited number of tiles, how do we determine the right tiles to promote among candidates potentially belong to different temporal segments to maximize viewer's QoE. More importantly, the probabilities of a viewer watching at different directions keep changing along the playback. Furthermore, modern streaming systems are often HTTP based while a HTTP streaming service can suffer from unpredictable bandwidth fluctuations as the HTTP session cannot be easily cancelled in practice. Thus, to efficiently explore two-dimensional (temporal and spatial) adaptability in 360-degree video streaming, we need to answer the following questions: 1) how to determine the importance of a tile and 2) how to react to the high variability of visual region prediction and the unpredictable network fluctuations.

To overcome these challenges, in this paper, we propose Bi-Adaptive Streaming for 360-degree videos (BAS-360°). BAS-360° is a tile-based streaming solution that jointly explores spatial and temporal adaptabilities. For spatial adaptation, BAS-360° follows the principle to arrange high quality for visible tiles and low quality for invisible tiles. Additionally, BAS-360° selectively separates the visible region into a focal area and a few peripheral areas to further exploit the spatial adaptability. For temporal adaptation, BAS-360° carefully selects the quality for tiles without exceeding the predicted bandwidth budget. In addition, we develop a metric called *bandwidth waste expectation*, which is determined by both the probability of being watched and the visual quality of a tile, to characterize the importance of a tile and to allocate the available bandwidth to the different tiles. All of these considerations are captured by an integer linear programming (ILP) problem, in which two-dimension adaptation is represented as the constraints and the bandwidth waste expectation is considered in the optimization objective. To deal with the variability of the visual region prediction and the unpredictable network fluctuations, *stream priority* and *stream termination* features of HTTP/2 are employed. If it is determined that a tile in unsuitable for current network or viewport conditions is being delivered, we can use the *stream termination* feature to cancel it in a timely manner. *Stream priority* can help pipeline tile transmissions, eliminating the overhead of multiple HTTP requests, and adjust the quality of the tile that has not been downloaded yet at anytime. Furthermore, by organizing the tile transmissions in the order of their *bandwidth waste expectation*, the user's QoE in terms of visual quality can be preserved even if the bandwidth encounters unpredictable drop. It is worth noting that in this paper, we aim at developing a general framework for 360-degree video streaming. We are not focusing on specific prediction methods for either the viewer's head orientation or the future bandwidth. Any prediction method can be integrated into our streaming framework.

The rest of the paper is organized as follows. Section II discusses background and related work. Section III presents the design of our work. Section IV describes the prototype implementation and the results of evaluation are reported in Section V. Section VI concludes the paper.

II. Background and Motivation

In this section, we discuss some background and related work that motivated our research.

A. Human Visual System (HVS) and 360-degree video

Usually a 360-degree video is presented as a sphere surrounding the viewer's head, and the Field of View (FoV) can be described as a rectangle warped on the surface of the sphere. This region is $\sim 180°$ horizontally and $\sim 135°$ vertically right before the human eyes. This region will expand by $\sim 30°$ on both axes due to the eye rotation. In the visible region, there are generally two types of areas, the central area and the peripheral area. The peripheral vision is considered weak in HVS at distinguishing the detail, color, and shape. Figure 1 shows the HVS horizontally.

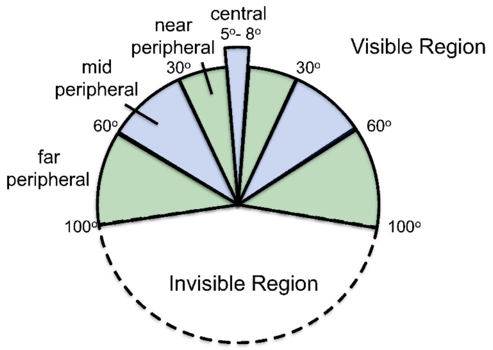

Fig. 1. Human Visual System

B. Spatially Adaptive Streaming

Naturally, the spatial adaptability can be exploited in the 360-degree video streaming to save bandwidth since ususally only portions of the rendered frames are viewed by the user.

One of two widely adopted means to save the bandwidth is to arrange more pixels for the visible region than the invisible region during projection before encoding. Corbillon at el. [17] proposed to generate the video representations characterized not only by their bit-rate but also by different Quality Emphasis Center (QEC), around which higher quality is provided. Similar methods are also used in Facebook's pyramid mapping [18] and offset cubic mapping [19], [20]. All of these projection-based methods need to generate separate versions of the video for a potential viewer's head orientation, and more versions are needed if they are to be integrated into the DASH-like streaming systems.

The alternative means is to spatially cut the video into tiles. Only the tiles overlapping with the user's FoV are delivered with a higher quality and the other ones are either transmitted with a lower quality or are simply discarded. DASH [15] and one of its amendment, SRD [21], are the major techniques used by many tile-based streaming schemes [5], [6], [7], [8], [9], [10], [11], [22]. In addition to DASH, Layered Video Coding (LVC) is another method to distinguish the quality between tiles [13]. Sreedhar et al. [23] proposed to encode the views from a same video in individual layers, wherein the complete view is encoded with low quality and the view at the direction watched by the user is encoded with high quality. Zare et al. [12] suggested to further improve the bandwidth efficiency by taking advantage of motion-constrained tile sets (MCTS).

Yu et al. [24] formulated the choice of representation in the tile-based streaming as a multi-dimensional, multiple-choice knapsack problem, maximizing the coding efficiency under resource constraints. As tile-based methods are more storage efficient compared to the projection-based methods, BAS-360° is built upon tile-based streaming.

C. HTTP/2

HTTP/2 [25] inserts an additional interpreted layer below the original HTTP/1.1 semantics, which breaks the HTTP requests and responses into *frames*. One HTTP session is associated with one *stream*. There are a few frames used to implement the HTTP/1.1 features, like the *HEADERS* frames delivering the HTTP header information and the *DATA* frames being the vehicle of the request/response data. More frames are defined to attain some prominent features only in HTTP/2, like *stream termination*, *stream priority*, and etc.

Stream termination: Stream termination is achieved by sending a specific *RST_STREAM* frame, either from the client or the server. When an endpoint receives a *RST_STREAM* frame, the active stream delivering that frame is then closed.

Stream priority: An HTTP/2 stream holds two flags representing its priority weight and parent stream. The flags can be setup or updated by the *HEADERS* frames or the specific *PRIORITY* frames. An HTTP/2 stream starts its transmission only if all streams that it depends on have been closed. For the HTTP/2 streams depending on the same parent stream, the bandwidth is allocated according to their priority weights, wherein more bandwidth is allocated to the stream with a higher priority weight. If an HTTP/2 stream sets the *exclusive* flag, it becomes the only child of its parent stream and the original children streams now depend on the exclusive stream.

III. BAS-360° Design

In this section, we will present the design details of BAS-360°. BAS-360° organizes the network delivery based on *macro-streaming units*, which convey meaningful visual parts in the 360-degree video. Bitrate selection for tiles in macro-streaming units is formalized as an integer linear programming problem for maximizing the expected visual quality. Spatial adaptability and temporal adaptability are exploited by imposing the constraints concerning continuous playback, quality arrangement, and smooth tile transition. A transmission sequence is generated as long as all involved tiles are arranged with a bitrate. In order to approach the optimal transmission sequence in linear time, an algorithm is then proposed. In the last part, we illustrate how to employ *stream termination* and *stream priority* of HTTP/2 to realize the variable optimal transmission sequence.

A. Macro-streaming Units

BAS-360° spatially cuts a video composed of the projected rectangular frames into tiles. Every tile is then chunked into segments of fixed duration, and these segments are further encoded at multiple bitrates for HTTP streaming. As a result, a tile t can be identified as a three-tuple $\langle i, pos, b\rangle$, where the tile belongs to the ith segment, pos represents its position in the original frame, and b is its bitrate.

However, scheduling network transmissions in the unit of tile is problematic. Rendering only one or two tiles is hardly a satisfying watching experience for viewers. So we organize tiles into *macro-streaming units* during the network transmission according to HVS described in Section II-A. A macro-streaming unit consists of a set of tiles representing a meaningful visual part while rendering. A straightforward organization considers tiles overlapping the visible region as a macro-streaming unit, and so does the remaining ones inside the invisible region. The organization policy can be extended as needed. For example, spatial adaptability can be further exploited by organizing the invisible tiles as multiple macro-streaming units. The furthest invisible macro-streaming units to the visible region can be discarded while the other invisible macro-streaming units are delivered with low quality in case that the viewer turns his/her head. Focusing on the principles of our design, we will organize only two categories of macro-streaming units, representing the visible region and the invisible region.

The visible region is determined by a given head direction ω, derived from *yaw* and *pitch*. A macro-streaming unit can be formally denoted as

$$\mathbf{V}(\omega, i) = \{t(i, pos, b) \mid \mathbf{Proj}(pos) \text{ overlaps the visible region}\}$$

to represent a visible region, where $\mathbf{Proj}(\cdot)$ is the projection function that projects the tile from the 2D rectangular frame to the rendered sphere, or

$$\mathbf{I}(\omega, i) = \{t(i, pos, b) \mid \mathbf{Proj}(pos) \text{ resides in the invisible region}\}$$

to represent an invisible region. For generality, we will use $\mathbf{M}(\omega, i)$ to represent the macro-streaming unit if the type is not the concern. Figure 2 shows an example of all macro-streaming units for a video segment while there are only two probable head orientations in opposite directions.

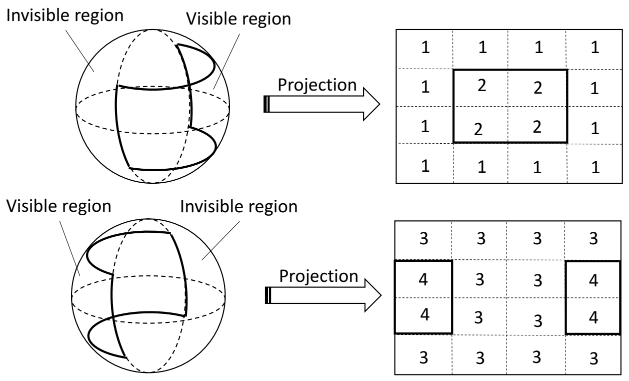

Fig. 2. All four macro-streaming units of a video segment if there are two possible head orientations in opposite directions. The macro-streaming units marked by 1 and 3 represent two possible invisible regions. The macro-streaming units marked by 2 and 4 represent the possible visible regions.

So we need to schedule the transmissions of the macro-streaming units from K temporal segments, which starts at the

i_0th segment. The transmission sequence can be represented as

$$Seq = \{M_j(\omega,i) \mid \omega \in \Omega, i = i_0,\ldots,i_0+K-1, j = 1,2,\ldots,N\}$$

where Ω contains all possible head orientations, j reflects the delivering order, and N is the number of macro-streaming units involved. When generating the transmission sequence Seq, our optimization objective is to maximize the visual quality without consuming all the available bandwidth.

1) Projection in BAS-360°: The projection functions can be equirectangular [26], polyhedron-based [19], [18], etc. For example, the equirectangular projection can be presented as

$$\{(yaw, pitch) \mid yaw = (x/w - 0.5)\cdot 360, pitch = (0.5 - y/h)\cdot 180\}$$

where w/h is the width/height of the 2D plane, and (x,y) represent the coordinates of a projected pixel on the plane. It is worth noting that BAS-360° will not be affected by any specifically employed projection method because it aims at providing a general streaming framework weighting the video content by its relative position to the viewer before projection.

B. Problem Formalization

1) Objective Function: In order to allocate bandwidth resources to the tiles that are more likely to be watched in high priority, we want to define bandwidth waste from both unused bandwidth and bandwidth allocated to deliver invisible tiles. Minimizing the former component improves the overall video quality, and reducing the second component helps promote the quality of visible tiles. So in our system, the objective function is

$$minimize : (\mathbf{B} - \sum_{Seq} S) + \sum_{Seq} E[M] \tag{1}$$

where $\mathbf{B}$ is the available bandwidth resources, S represents the size of a macro-streaming unit and $E[\cdot]$ is the *bandwidth waste expectation* of a macro-streaming unit. $\mathbf{B}$ and S can be calculated as

$$\mathbf{B} = \int_{t_0}^{(i_0+K-1)D} B(t)dt \quad \text{and} \quad S(\omega,i) = \sum_{\mathbf{M}(\omega,i)} bD$$

where t_0 is the timestamp of now, D is the segment duration, $B(t)$ is the predicted bandwidth function of time and b represents the corresponding bitrates of segments belonging to the macro-streaming unit $\mathbf{M}(\omega,i)$. Tian et al. in a previous study [27] suggest using historical information for the best prediction. The bandwidth waste expectation of a macro-streaming unit is denoted as

$$E[\mathbf{M}(\omega,i)] = (1 - P(\omega,i))\cdot S(\omega,i) \tag{2}$$

where $P(\omega,i)$ is the probability of the viewer watching the ith segment at the direction ω. In practice, this probability can be carried out by the Bayes-based method like $\frac{|N_i(\omega)\cap N_{i_0}(\omega_0)|}{|N_{i_0}(\omega_0)|}$, where $N_i(\omega)$ is the population whose watching direction is ω at the ith segment, ω_0 is the watching direction now.

2) Constraints in Spatial Dimension: To exploit the spatial adaptability in 360-degree video, we need to formalize how to select bitrate for tiles inside the visible region and the invisible region. We follow distinct rules for different types of macro-streaming units as the visible region is more important to the user's QoE in terms of visual quality.

Visible region: According to HVS described in Section II-A, there are three rules for the visible region: 1) the visible region should either be fully occupied or consist of no tiles – a visible region can be composed of no tiles because more than one visible regions are considered in our streaming system, 2) the quality of tiles in an area, e.g., the central area, should be the same, and 3) tiles in the central area should be allocated equal or higher quality than the ones in the peripheral areas, and the quality difference of neighboring areas should be at most one bitrate level.

Specifically, if a tile is encoded into n different bitrate levels, the available bitrates are denoted as $\mathbf{b} = \{b_0, b_1, \ldots, b_n\}$, where b_0 is a special bitrate level that represent an undelivered tile. A visible region is denoted as $\mathbf{V} = \bigcup \mathbf{A}_i, i = 1,2,\ldots,$ where $\mathbf{A}_i$ represents the areas residing in the visible region, and the increasing subscripts means further peripheral areas. $\mathbf{A}_1$ always means the central area. Then the previously described rules can be formally represented as

$$\begin{aligned}\mathbf{V}(\omega,i) = \{t(i,pos,b) \mid \mathbf{Proj}(pos) &\text{ overlaps the visible region}\\ &\wedge\ \forall t : b = b_0 \vee \forall t : b != b_0\\ &\wedge\ \forall t \in \mathbf{A}_j : b = b', b' \in \mathbf{b}\\ &\wedge\ \forall b^j \in \mathbf{A}_j, \forall b^k \in \mathbf{A}_k, j = k-1,\\ &\quad b^j = b_m : b^k \in \{b_m, b_{m-1}\}\}\end{aligned}$$

Invisible region: Since the invisible region hardly affects the visual quality, the corresponding tiles should be delivered with the lowest quality or not delivered at all. The quality arrangement rule for the invisible region can be represented as

$$\begin{aligned}\mathbf{I}(\omega,i) = \{t(i,pos,b) \mid \mathbf{Proj}(pos) &\text{ resides in the invisible region}\\ &\wedge\ \forall t : b = b_0 \vee \forall t : b = b_1\}\end{aligned}$$

For a macro-streaming unit containing N tiles, there are in total $|\mathbf{b}|^N$ tile quality arrangement possibilities. By imposing the rules presented aforementioned, only a limited number of the quality arrangements are legal. Figure 3 shows an example where there are two available bitrate levels for a tile and a visible region contains one central area and one peripheral area. In this example, only limited (4) tile quality arrangements need to be considered. So for one type of the macro-streaming unit, we can generate all possible quality arrangements offline and store them in a *quality arrangement array*. While the stored quality arrangements are sorted according to the sum of the bitrates, an *upgrade* operation is defined as choosing the neighboring quality arrangement with higher bitrate sum, and a *degrade* operation is scaling down the quality arrangement.

3) Constraints in Temporal Dimension: From the temporal perspective, video playback should be continuous when selecting the quality of the tiles. Continuous playback means at least

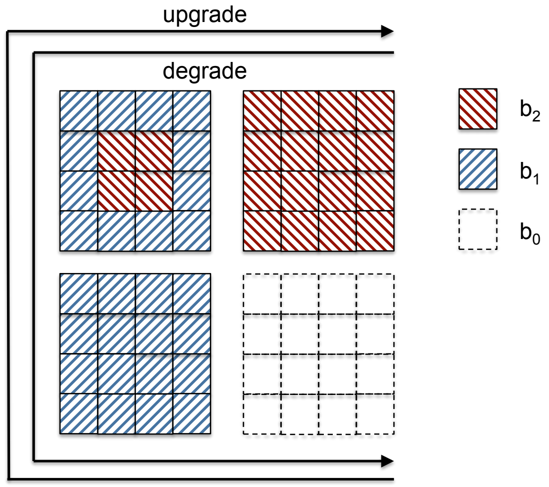

Fig. 3. An example quality arrangement array for a macro-streaming unit of 16 tiles

one visible region should be delivered for each segment. There might be two reasons that make the transmission illegal: 1) no visible region is scheduled for a segment, which means b_0 is selected for all corresponding tiles, or 2) the bandwidth budget does not allow delivering the regions before their playback, which can be validated by

$$S_j(\omega,i) \leq \int_{t_0}^{(i-1)D} B(t)dt - \sum_{n=1}^{j-1} S_n(\omega,i), \ j=1,2,\ldots,N$$

C. Transmission Sequence Determination

With the selected bitrates, generating the transmission sequence is nontrivial. We want to organize the transmissions in the order of increasing bandwidth waste expectation. This helps preserve the visual quality if a sudden bandwidth drop is encountered. It is also not necessary to deliver all involved tiles in the macro-streaming units due to region overlapping. The same tiles existing in multiple regions can be deduplicated for bandwidth saving.

In our scheme, we propose to firstly generate a *inner-segment transmission sequence* for each temporal segment where duplicate tiles appear, then merge them into a *inter-segment transmission sequence*.

Inner-segment transmission sequence: The inner-segment transmission sequence is selected from the permutation of all involved but not yet delivered macro-streaming units of one segment. In a particular transmission sequence, we sequentially apply tile deduplication to remove redundant tiles. Particularly, if we have three marco-streaming units M_1, M_2 and M_3. The macro-streaming units are sorted by their bandwidth waste expectation, leading to a preliminary sequence $\{M_1, M_2, M_3\}$. We first remove tiles existing in both M_1 and M_2 from M_2, then tiles existing in either M_1 and M_3 or M_2 and M_3 from M_3. Then, the trimmed inner-segment transmission sequences are ready to be merged to a unifying one.

Inter-segment transmission sequence: The inter-segment transmission sequence, or the resulting transmission sequence, is generated by merging all K inner-segment transmission sequences into one. Since we can not adjust the order among inner-segment units due to the deduplication operations, we need to carefully merge the K sequences. The merge algorithm is shown in Procedure 1. The basic idea of our algorithm is to sort the inter-segment macro-streaming units according to their bandwidth waste expectation while still retaining the transmission order of the inner-segment units.

Procedure 1 *Merge K inner-segment transmission sequences*

```
Input: $Seqs[K]$, $E[\cdot]$
▷ $Seqs[K]$ stores the $K$ inner-segment transmission sequences

while There are elements in $Seqs$ do
    $min \leftarrow \infty$
    for $i$ in $1 \ldots K$ do
        if $E[Seqs[i][0]] < min$ then
            $min \leftarrow Seqs[i][0]$
            $min\_idx \leftarrow i$
    Push $min$ to the tail of $Seq_{out}$
    Remove the head of $Seqs[min\_idx]$
    if The size of $Seqs[min\_idx]$ is 0 then
        Remove $Seqs[min\_idx]$ from $Seqs$
return $Seq_{out}$
```

D. Approaching the Optimal Sequence

Generating the optimal transmission sequence from all possible ones are time consuming. The computation complexity for searching the optimal sequence is $O((|\Omega|!)^K |arr|^{|\Omega| \cdot K})$, where *arr* is the quality arrangement array described in Section III-B2. The time to search for the solution increases exponentially with the increasing number of temporal segments, K, and the number of possible user head orientations. This makes our method impractical in a streaming session where the determination should be made in a few seconds.

We notice that when degrading a macro-streaming unit in a transmission sequence, the value of the objective function always increases because the bandwidth resources freed are greater than the reduced bandwidth waste expectation by degradation ($E(M) = (1-P) \cdot S < S$). So if we degrade the macro-streaming unit in a transmission sequence that increases the objective function the least, we can successfully approach the optimal solution. Our approaching algorithm is presented in Procedure 2. The idea of this algorithm is to first setup all involved tiles at the highest quality. Then we repeatedly degrade the quality of one selected macro-streaming unit if the transmission sequence demands the bandwidth resources exceeding the budget. The criteria of selecting the victim macro-streaming unit is the minimum bandwidth waste increment resulting from the corresponding degrade operation. In the algorithm, *dedup()* means the tile deduplication described in Section III-C and *merge()*, which is presented in Algorithm 1, is used to merge inner-segment transmission sequences. *size()* calculates the size of corresponding macro-streaming unit. *isLegal()* checks if the resulting sequence consumes bandwidth resources under the budget, which has been discussed in Section III-B3, and *degrade()* has been defined in Section III-B2. To the end, the computation complexity of the approaching algorithm is $O(|arr| \cdot |\Omega|^2 K^2)$.

Procedure 2 *Optimal Sequence Approaching Algorithm*

```
Input: $Seqs[K]$, $E[\cdot]$, size($\cdot$), dedup($\cdot$), merge($\cdot$), isLegal($\cdot$), Degrade($\cdot$)

Arrange all tiles in $Seqs$ with the highest quality
while $Seqs$ is degradable do
    for $i$ in $1 \ldots K$ do
        for $j$ in $1 \ldots Seqs[i].size$ do
            for $k$ in $1 \ldots j-1$ do
                $Seqs_{dedup}[i][j] \leftarrow$ dedup($Seqs[i][j]$, $Seqs[i][k]$)

    $Seq_{out} \leftarrow$ merge($Seqs_{dedup}$)
    if isLegal($Seq_{out}$) then
        break

    $MinDiff \leftarrow 0$
    for $i$ in $1 \ldots K$ do
        for $\mathbf{M}$ in $Seqs[i]$ do
            $\mathbf{M}' \leftarrow$ Degrade($\mathbf{M}$)
            $WasteDiff \leftarrow \text{size}(M) - \text{size}(M') - E[M] + E[M']$
            if $WasteDiff < MinDiff$ then
                $MinDiff \leftarrow WasteDiff$
                $\mathbf{M}_d \leftarrow \mathbf{M}$

    Degrade($\mathbf{M}_d$)
return $Seq_{out}$
```

E. Transmission Reprioritization and Termination

Recent studies [16], [14] have shown that long-term prediction of the user head movement is error-prone and short-term prediction is more accurate and is highly correlated to the current user head orientation, implying the volatile probabilities applied in Equation 2. On the other hand, although historical bandwidth information can be used to improve the bandwidth prediction [27], several abrupt and unexpected network fluctuations can appear during a streaming session. The variability of visual region prediction and the unpredictable network fluctuations lead to frequently varied optimal transmission sequence. Traditional HTTP/1.1-based streaming system is incapable of agilely tailoring the transmission sequence due to the lack of mechanisms to terminate the ongoing HTTP session and prioritize the network transmissions. To efficiently realize the variable optimal transmission sequence, we switch the streaming vehicle from HTTP/1.1 to HTTP/2, which features *stream priority* and *stream termination*.

In particular, BAS-360° monitors the viewer's head orientation after one video segment has been watched and generates the optimal transmission sequence from the newly calculated probabilities. Existing transmission sequence is updated by adding, removing, and changing order of the tile transmissions according to the new sequence. The manipulation is implemented at the HTTP/2 layer, and more details can be found in Section IV. Furthermore, if the measured bandwidth has significantly deviated from the predicted value, we will terminate all active tile transmissions and prepare for rescheduling.

IV. BAS-360° IMPLEMENTATION

We implement an HTTP/2-based streaming system which includes a 360-degree video player equipped with BAS-360° in C++, an HTTP/2 client on which the player operates, and an HTTP/2-based server handling the requests for stored video tiles. Both the client and the server are built upon *nghttp2* [28], which is a widely deployed HTTP/2 library in C. The architecture of our implemented streaming system is depicted in Figure 4. As shown in the figure, there are five major modules we implement to support BAS-360°: The *bandwidth predictor* predicts the future bandwidth from the historical information recorded by the *bandwidth monitor*, and the specific method used is presented in DASH2M [29]. The basic idea of the bandwidth prediction method is to average the historical bandwidth measurements, which are in advanced weighed by their temporal distance to now. The *head orientation monitor* collects current user head orientation. The *ILP solver* (Integer linear programming solver) generates the optimal transmission sequence as long as a macro-streaming unit is downloaded. The predicted bandwidth, current user head orientation combined with the user head orientation distribution, which in practice can be collected by the server and sent to the client, are fed as input to the *ILP solver*. Both the *ILP solver* and the *bandwidth monitor* are able to direct the HTTP/2 client on what to request from the server. The *ILP solver* manipulates the order of tile delivery to comply with its output transmission sequence, and the *bandwidth monitor* tears down all active transmissions as long as the measured bandwidth has significantly deviated from the predicted one. Retrieved video tiles are filled into the playback buffer. The playback of a 360-degree video in this player continues only if all tiles within the user's viewport exist, and it stops as long as any visible tiles are missing. When playing, the tile with the highest available quality at a given position is rendered.

To manipulate tile delivery, we need the *adding*, *deleting* and *reprioritizing* operations. All the operations are implemented in the HTTP/2 layer. For the *adding* operation, a tile transmission can be easily added by establishing a new HTTP/2 stream and then sending the corresponding HTTP request. To *delete* an active tile transmission, we send the *RESET* frame over its HTTP/2 stream. The *reprioritizing* operation is implemented by the *stream priority* feature in HTTP/2. A tile transmission can be reordered at any time by informing the server with a *PRIORITY* frame, where the *stream dependency* field represents another specific HTTP/2 stream (another tile transmission) that should be completed before its delivery, and the *exclusive* flag is always set to disable the transmission multiplexing among HTTP/2 streams.

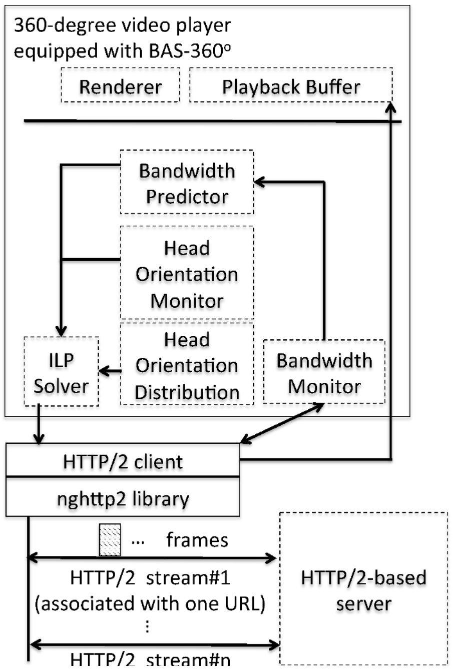

Fig. 4. Architecture of BAS-360° Prototype

V. BAS-360° Evaluation

We evaluate BAS-360° by comparing its performance with naive tile-based streaming methods built upon HTTP/1.1. In the evaluation, we actively manipulate the network conditions to observe how different schemes react to the network fluctuations. We also simulate both failed and successful user vision predictions to assess the streaming performance of different schemes.

A. Experiment Setup

In addition to BAS-360°, we also implement two naive tile-based streaming methods for 360-degree video streaming. That is, the *all-download* scheme that downloads all tiles in all directions and the *on-demand* scheme that expects to only download the tiles in the user's vision according to the prediction result. Both naive schemes will choose the same bitrate for all involved tiles without exceeding the predicted bandwidth budget. We choose to compare BAS-360° against these two schemes that represent the most conservative and the most aggressive tile-based streaming methods.

All of our experiments are conducted on a Linux machine with a 64-bit Intel Pentium CPU 2.8 GHz dual core, 6 GB memory, 2×32 KB L1 caches, 2×256 KB L2 caches and shared 3 MB L3 cache. The installed operating system is Ubuntu 12.04 with Linux kernel 3.13.0-66-generic. The video file is first split spatially into tiles by ffmpeg [30], and the tiles are further segmented temporally by the MP4Box tool [31]. We use *tc* as the network shaping tool impose shaping rules on IP packets that have 8080 as either *dst port* or *src port*. Such a setting is to eliminate the impact of other uncertainties so that we can get a more accurate assessment of our scheduling method.

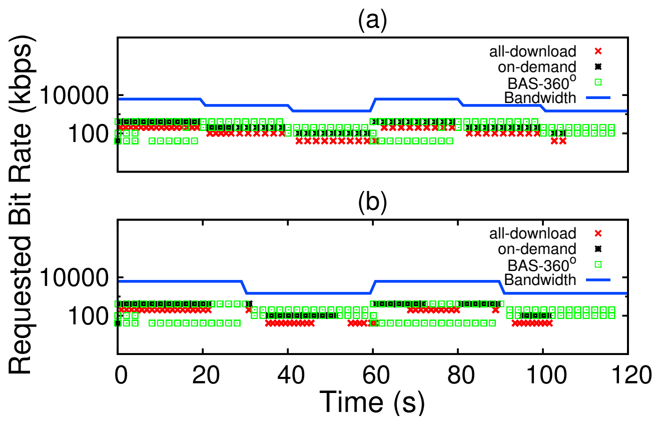

Fig. 5. Requested segment bitrate

B. Streaming Performance vs. Network Fluctuations

At first, we want to evaluate if BAS-360° can promptly react to the network fluctuations as *stream termination* is used. We compare the streaming performance of BAS-360° with the naive streaming schemes. The 360-degree video we used in this experiment is two-minute long, encoded with H.264/AVC after the equirectangular projection. We split the video into 16 (4×4) tiles, and each tile is further chunked into segments of 2 seconds duration. To make the experimental result more clear, each tile is encoded into four constant bitrate levels, which are 400 kbps, 200 kbps, 100 kbps and 40 kbps. We set 4 possible user head directions as yaw is one of $0°, 90°, 180°, 270°$ and pitch is always $0°$, covering 8 tiles as the visible region for each.

To eliminate the impact from failed head movement predictions, *on-demand* and our proposed scheme always know the user's head directions during the playback. And we set the probability for the user abruptly turning back as 0.05 for every segment. In the *on-demand* and the *all-download* schemes, we select the highest bitrate for the tiles of the next segment without exceeding the predicted bandwidth. For our scheme, we invoke the scheduling algorithm and arrange the corresponding requests if a macro-streaming unit is downloaded. The naive scheduling schemes send requests as long as the buffer length drops below the critical value, which is set as 12 seconds, and stop requesting new segments if the buffer length reaches 20 seconds. In all schemes, we download all tiles at the lowest bitrates for the first 3 segments to start the video playback and request for the visible tiles of the next segment with the lowest quality as long as the playback buffer is empty. We first check the streaming performance if the bandwidth can be accurately predicted. We manually introduce bandwidth variation in the following sequence — 6150, 2850, 1450, 6150, 2850, 1450 kbps. The bandwidth varies every 20 seconds. The RTT is set to be 20 ms all the time. In this experiment we preset the bandwidth information in all schemes. After that, we also conduct experiment as the bandwidth can not be accurately predicted. In the second experiment, the bandwidth variation is set to follow the sequence — 2850, 1450, 2850, 1450 kbps.

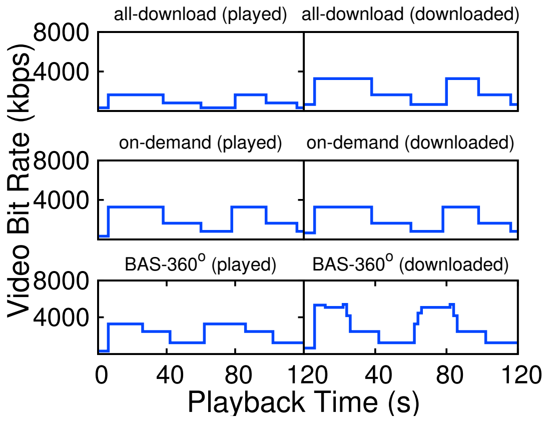

Fig. 6. Video quality variations with preset bandwidth information

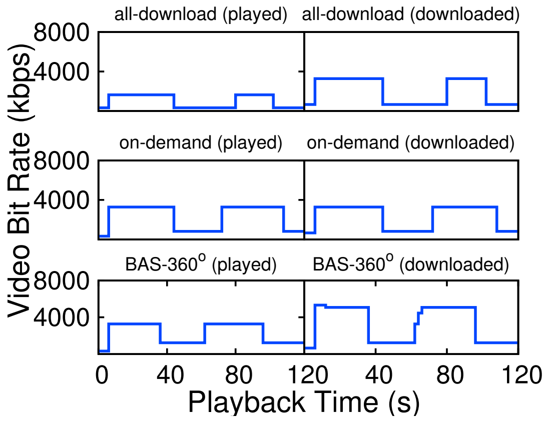

Fig. 7. Video quality variations without preset bandwidth information

Figure 5 shows the timing of requests during the playback for both experiments. The x-axis is the time when the requests are sent, and the y-axis is in log-scale, representing the bitrate of the tiles requested in kbps. The bandwidth variation is plotted as a solid line in the figures. From Figure 5(a), all schemes can promptly react to bandwidth variations because of accurate bandwidth estimation. The *all-download* scheme always chooses a lower quality compared to *on-demand* and BAS-360° because the number of tiles required to be downloaded doubles. Different from the naive schemes, our scheme also downloads tiles in the invisible region with the lowest quality if the bandwidth resources are abundant (750 kbps duration). In Figure 5(b), in spite of abrupt bandwidth drops at the 30th second and the 90th second, all the schemes can detect network fluctuations and scale down the requested tile quality. Our scheme has the same level network adaptability as HTTP/1.1 based streaming schemes equipped with short segment duration because of the adopted stream termination feature. In addition, our scheduling method can combine different tiles of different bitrates in a frame under the same bandwidth budget, providing better users' QoE compared to the naive streaming schemes.

Figure 6 and Figure 7 show the played video quality and the downloaded video quality in both experiments. The x-axis is the playback time in seconds and the y-axis is quality summation of the played tiles (left) and the downloaded tiles (right), indicating the proportion of bandwidth allocated to deliver the actually viewed content. We can observe that even though the *all-download* scheme downloads tiles with the same quality summation as the *on-demand* scheme, the played quality summation is much lower because half of the tiles are downloaded but not watched. With accurate knowledge of the future bandwidth, our scheme can fully exploit the bandwidth resources to improve visual quality, whose average downloaded quality summation is 2865 kbps and the average played quality summation is 2245 kbps. Both of the metrics are higher than *all-download* (the average played quality summation is 1026 kbps and the average downloaded quality summation is 2052 kbps) and *on-demand* (the average played quality summation is 2111 kbps and the average downloaded quality summation is 2127 kbps). When the future bandwidth is unpredictable, the visual video quality of our scheme will not be impaired even the delivery sequence was determined by the misleading bandwidth prediction. The average played/downloaded video quality summation of our scheme in the experiment without preset bandwidth information is 2259/3193 kbps. Our scheme performs as good as *on-demand* (2286/2302 kbps), which promptly reacts to the bandwidth variation due to the short segment duration. The average played/downloaded video quality summation of *all-download* in the second experiment is 972/1944 kbps.

C. QoE vs. Head Orientation Prediction

To evaluate how viewport prediction accuracy affects the viewer's QoE across all schemes, we also conducted an experiment using a simulator implemented in C++. In the simulation, the video length is 1 minute long. For the naive scheduling methods, the critical buffer level is set as 6 seconds. To eliminate the impact of bandwidth variation, we set the bandwidth as a fixed value during the simulation, which is selected from $\{1800, 2850, 6150\}$ kbps. We randomly generate the user's head orientation for each run of the simulation as follows: 1) The probability distribution of head orientation are generated for each segment. The distribution is selected from $\{0.75, 0.1, 0.1, 0.05\}$, $\{0.4, 0.4, 0.1, 0.1\}$, $\{0.3, 0.3, 0.3, 0.1\}$ and $\{0.25, 0.25, 0.25, 0.25\}$, which represents that the user wants to simultaneously watch 1, 2, 3 or all 4 directions. 2) The user's actual watching directions are then generated according to the distributions. And 3) for the *on-demand* scheme only, the player predicts the user's vision for a segment by choosing the direction that holds the highest probability. All other experimental parameters are the same as those described in Section V-B. We run the simulation 5000 times for each combination of the scheduling scheme and the chosen bandwidth.

TABLE I
USERS' QOE STATISTICS

Bandwidth	*all-download*	*on-demand*	BAS-360°
Average Stalling Time (s)			
1800 kbps	0.00 ± 0.00	8.24 ± 0.06	7.83 ± 0.08
2850 kbps	0.00 ± 0.00	6.03 ± 0.09	3.88 ± 0.10
6150 kbps	0.00 ± 0.00	2.98 ± 0.04	0.02 ± 0.00
Average Played Quality (kbps)			
1800 kbps	770 ± 0	1066 ± 9	810 ± 9
2850 kbps	770 ± 0	1057 ± 14	1254 ± 14
6150 kbps	1507 ± 0	1991 ± 16	1977 ± 5

Table I lists the statistical results of the simulation with 95% confidence interval. We can observe that *all-download* has the stalling time of 0 second all the time since all tiles are downloaded without any head movement prediction. However, *all-download* has the lowest average played quality, which are 770 kbps and 1507 kbps, respectively. Our scheme always has less stalling time compared to *on-demand* because of the prioritization operations. The *all-download* scheme and the *on-demand* scheme cannot ramp up the video quality when the bandwidth increases from 1800 kbps to 2850 kbps because of the unifying tile quality arrangement strategy. Our scheme can effectively take advantage of the increased bandwidth to improve the video quality and reduce the average stalling time.

VI. CONCLUSION

In this paper, we present BAS-360°, a tile-based 360-degree video streaming solution, in which the bitrate selection for tiles is formulated as an integer linear programming problem. The ILP aims at minimizing the bandwidth waste while several constraints are developed to exploit the spatial and temporal adaptabilities simultaneously. BAS-360° also leverages the *stream termination* and *stream priority* features of HTTP/2. Evaluation results show that BAS-360° outperforms naive tile-based 360-degree streaming strategies when there are network fluctuations or error-prone predictions.

VII. ACKNOWLEDGMENT

We appreciate constructive comments from anonymous referees. This work is partially supported by NSF under grants CNS-1524462 and CNS-1618931 and gift funding from Adobe Research.

REFERENCES

[1] "Facebook," https://www.facebook.com/.

[2] "YouTube," https://www.youtube.com/.

[3] Netflix, "Internet Connection Speed Recommendations," https://help.netflix.com/en/node/306.

[4] "Akamai's [state of the internet] q1 2016 report," https://www.akamai.com/uk/en/multimedia/documents/state-of-the-internet/akamai-state-of-the-internet-report-q1-2016.pdf.

[5] M. Hosseini and V. Swaminathan, "Adaptive 360 VR Video Streaming: Divide and Conquer!" *arXiv preprint arXiv:1609.08729*, 2016.

[6] ——, "Adaptive 360 VR Video Streaming based on MPEG-DASH SRD," in *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 407–408.

[7] M. Hosseini, "View-Aware Tile-Based Adaptations in 360 Virtual Reality Video Streaming," in *Virtual Reality (VR), 2017 IEEE*. IEEE, 2017, pp. 423–424.

[8] D. Ochi, Y. Kunita, K. Fujii, A. Kojima, S. Iwaki, and J. Hirose, "HMD Viewing Spherical Video Streaming System," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 763–764.

[9] D. Ochi, Y. Kunita, A. Kameda, A. Kojima, and S. Iwaki, "Live Streaming System for Omnidirectional Video," in *Virtual Reality (VR), 2015 IEEE*. IEEE, 2015, pp. 349–350.

[10] J. Le Feuvre and C. Concolato, "Tiled-Based Adaptive Streaming Using MPEG-DASH," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 41.

[11] L. D'Acunto, J. van den Berg, E. Thomas, and O. Niamut, "Using MPEG DASH SRD for Zoomable and Navigable Video," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 34.

[12] A. Zare, A. Aminlou, M. M. Hannuksela, and M. Gabbouj, "HEVC-compliant Tile-based Streaming of Panoramic Video for Virtual Reality Applications," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 601–605.

[13] A. TaghaviNasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash, "Adaptive 360-Degree Video Streaming Using Layered Video Coding," in *Virtual Reality (VR), 2017 IEEE*. IEEE, 2017, pp. 347–348.

[14] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 Video Delivery over Cellular Networks," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. ACM, 2016, pp. 1–6.

[15] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 133–144.

[16] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu, "Shooting a Moving Target: Motion-Prediction-Based Transmission for 360-Degree Videos."

[17] X. Corbillon, A. Devlic, G. Simon, and J. Chakareski, "Viewport-Adaptive Navigable 360-Degree Video Delivery," *arXiv preprint arXiv:1609.08042*, 2016.

[18] Facebook, "Next-generation video encoding techniques for 360 video and VR," https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/.

[19] ——, "Optimizing 360 Video for Oculus," https://developers.facebook.com/videos/f8-2016/optimizing-360-video-for-oculus/.

[20] C. Zhou, Z. Li, and Y. Liu, "A Measurement Study of Oculus 360 Degree Video Streaming," in *Proceedings of the 8th International Conference on Multimedia Systems*. ACM, 2017.

[21] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim, "MPEG DASH SRD: Spatial Relationship Description," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, pp. 5:1–5:8.

[22] M. Graf, C. Timmerer, and C. Mueller, "Towards bandwidth efficient adaptive streaming of omnidirectional video over http: Design, implementation, and evaluation," in *Proceedings of the 8th ACM on Multimedia Systems Conference*. ACM, 2017, pp. 261–271.

[23] K. K. Sreedhar, A. Aminlou, M. M. Hannuksela, and M. Gabbouj, "Standard-Compliant Multiview Video Coding and Streaming for Virtual Reality Applications," in *Multimedia (ISM), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 295–300.

[24] M. Yu, H. Lakshman, and B. Girod, "Content Adaptive Representations of Omnidirectional Videos for Cinematic Virtual Reality," in *Proceedings of the 3rd International Workshop on Immersive Media Experiences*. ACM, 2015, pp. 1–6.

[25] IETF, "Hypertext Transfer Protocol Version 2 (HTTP/2)," https://tools.ietf.org/html/rfc7540.

[26] "Equirectangular Projection," http://mathworld.wolfram.com/EquirectangularProjection.html.

[27] G. Tian and Y. Liu, "Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming," in *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 2012, pp. 109–120.

[28] "nghttp2," https://github.com/nghttp2/nghttp2.

[29] M. Xiao, V. Swaminathan, S. Wei, and S. Chen, "DASH2M: Exploring HTTP/2 for Internet Streaming to Mobile Devices," in *Proceedings of the 2016 ACM on Multimedia Conference*. ACM, 2016, pp. 22–31.

[30] "ffmpeg," https://ffmpeg.org.

[31] "MP4Box," https://gpac.wp.mines-telecom.fr/mp4box/.