Storage Optimization for 3D Streaming Systems

Khaled Diab Tarek Elgamal Kiana Calagari Mohamed Hefeeda

Qatar Computing Research Institute
Qatar Foundation
Doha, Qatar

ABSTRACT

Three dimensional (3D) content is becoming attractive in entertainment events such as soccer games and movies. Also, 3D displays are widespread at homes, offices, and theaters. Yet, 3D content may lack good 3D experience due to varying display technologies and sizes. In addition, 3D content providers may not be able to deliver their content to all potential subscribers, which leads to viewership reduction or dissatisfaction. In this work, we propose the design of a system for enhanced 3D content streaming. In order to support all 3D display technologies and sizes in the system, we design different 3D versions of the original videos that are optimized for various displays. Moreover, we propose a storage optimization algorithm that optimizes storage usage in our system depending on versions popularity as well as storage and processing requirements. The algorithm satisfies the limited processing resources, maximum delay, and request rate requirements. We implemented and deployed the proposed system on the cloud for live testing. We simulated the proposed algorithm to study its effect on the storage requirements in 3D streaming systems. The results of the simulations show that the algorithm can achieve storage gain up to 360x compared to storing all versions.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentations**]: Multimedia Information Systems—*Video*; H.2.4 [**Database Management**]: Systems—*Multimedia databases*

General Terms

Design, Performance

Keywords

3D video, multimedia streaming, storage optimization, storage management, 3D streaming, stereo video

1. INTRODUCTION

MMSys '14, March 19 - 21 2014, Singapore

The growth of 3D multimedia content, 3D display technologies and high bandwidth Internet encourage many online streaming services, e.g., YouTube [6] and Trivido [5], to support 3D streaming. Currently, 3D displays vary in terms of technologies and sizes. This makes it difficult for content providers to support different types of displays with good 3D quality, which may decrease the viewership and satisfaction.

A 3D display is a device that is capable of presenting a depth perception to the user. Specifically, a 3D display presents two images, one for the left eye and the other for the right eye. The brain fuses the two images to give the 3D perception. Different technologies are used to enable 3D displays to present the two images needed to create the depth perception, including stereoscopic and autostereoscopic. In stereoscopic technology, the viewer needs glasses to control what each eye can see and at which time. In autostereoscopic technology, the viewer does not need glasses. The display uses optical elements to reflect the light of a view to the corresponding eye, and block the light of the opposite view.

Each 3D technology has different display sizes and for each display technology and size, the 3D content should be converted to a specific 3D format. Therefore, supporting all 3D display technologies requires generating display-specific 3D versions. Hence, storage demands for 3D content streaming systems become much higher than traditional 2D content streaming systems. Storage demands, however, can be reduced by processing 3D videos in real time, at the expense of increased computation requirements and latency to start streaming. This creates a trade-off between storage requirements and processing resources in 3D streaming systems.

In this paper, we consider the problem of increasing the viewership while optimizing storage usage in the 3D streaming systems. We focus on the design and implementation of a 3D streaming system that supports all current 3D displays and technologies while minimizing the needed storage by the system. As discussed in Section 2, we are not aware of previous works that analyze and optimize the storage requirements of 3D streaming systems.

In this paper, we make the following contributions:

- We categorize the 3D displays based on technology and size.
- We present the design of a 3D streaming system which enhances the 3D perceptual experience for different display sizes and technologies.
- We propose an efficient model, called 3D Version Tree, to represent and manage the creation and processing of

different 3D versions needed to support all 3D display technologies and sizes.

- We propose a storage optimization algorithm to minimize the storage requirements in 3D streaming systems. The algorithm decides on storing or removing a 3D version depending on its popularity and storage and processing requirements.
- We implement a 3D streaming system and evaluate our storage optimization algorithm using real 3D videos. Our experiments show that the proposed algorithm can achieve storage gain up to 360x, while meeting the delay requirements and available processing capacity constraints.

The rest of this paper is organized as follows. We describe related work in Section 2. We present the architecture of the proposed 3D streaming system in Section 3. In Section 4, we present the proposed storage optimization algorithm. In Section 5, we describe the implementation of our proof-of-concept 3D streaming system, and we evaluate the performance of the storage optimization algorithm. We conclude the paper in Section 6.

2. RELATED WORK

Early efforts in stereoscopic video streaming over the Internet were discussed by Johanson [10] who proposed the Smile! teleconferencing system. Smile! is a system for capturing, coding, compression and transmission of stereo streams. The sender associates two independently encoded monoscopic video streams and identifies the left view and the right view. The receiver of monoscopic streams uses timestamps of the two video streams to synchronize the left and right video streams for playback. The system was tested using active shutter-glasses. The main focus of this work is the transport protocol extension which enables two video streams to be associated and identified as left and right views. Pehlivan et al. [15] designed an end-to-end stereoscopic video streaming system that selects transmission of mono or stereo video depending on the available bandwidth and display equipment. However, the system was tested on one type of display where end users can view the stereo video using two projectors and polarized glasses.

Multi-view client-server systems, where a scene can be displayed from different viewpoints, were discussed in [13], [11] and [12]. Lou et al. [13] proposed a system where the server transmits two views requested by the user according to the viewing angle. This results in a relatively large delay due to the network latency. Kimata et al. [11] proposed a similar system with a goal to achieve low delay interactive 3D video live streaming, in which each user can change the camera angle fast enough to provide realistic feeling of the event. To achieve fast camera switching, the server transmits the requested multiple views and the client prefetches multiple views in order to decrease the latency of changing the views. Kurutepe et al. [12] proposed a system for efficient transmission of multi-view video over IP networks based on multicast. Baicheng et al. [20] described a 3D video streaming system which was developed to distribute 3D content for the 16th Asian Games. The main focus of this work was 3D media encoder and decoder and there is no consideration for storage optimization

In addition to the academic works mentioned above, there has been significant interest from the industry, such as YouTube, 3DVisionLive, Trivido, and 3DeeCentral. YouTube [6] supports multiple 3D formats including anaglyph (red-cyan , blue-yellow or green-magenta), side by side, row and column interleaved. In addition, it supports HTML5 stereo view, which is the format for active-shutter displays that utilize NVIDIA 3D Vision. YouTube also supports 3D content on autostereoscopic mobile devices such as LG Optimus 3D cellphone. YouTube is a commercial site with proprietary closed design. And unlike our system, YouTube does not change or customize the depth of videos for different displays. In addition, the storage system of YouTube and its optimization are not publicly known.

3DVisionLive [3] is a web channel for 3D video streaming and 3D photo sharing. 3DVisionLive uses the Microsoft Silverlight and IIS smooth streaming technologies. It is, however, limited to one display technology. In order to view 3D content on 3DVisionLive, NVIDIA active-shutter glasses are required.

Trivido [5] is a 3D Internet video platform. It supports anaglyph, side by side, row interleaved 3D formats in addition to 3D NVIDIA Vision format. Trivido supports only two display technologies: polarized and active-shutter, and it does not support autostereoscopic displays, nor does it address the problem of customizing the content for different displays.

3DeeCentral [2] supports 3D content on multiple 3D-enabled devices. Five different classes of displays are supported in the system: Internet-Connected 3D TVs, Windows 7 desktop devices, Windows 7 laptops, Android auto-stereoscopic devices such as LG Optimus, and HTC Evo 3D, iOS devices such as iPhone 4, and iPhone 4S. Despite the support of different types of devices, it is not clear whether 3DeeCentral provides adjusted depth for different sizes.

In summary, there has been significant interest from the academia and industry in streaming 3D videos and supporting various display technologies. However, the problem of optimizing storage requirements of such 3D streaming systems has received little attention. In addition, our work aims at providing a general categorization of 3D videos and display technologies that can support all current displays.

3. PROPOSED 3D STREAMING SYSTEM

In this section, we discuss the challenges of 3D streaming. Then, we describe the proposed architecture for 3D streaming systems.

3.1 Challenges of 3D Streaming

Storage and bandwidth demands of 3D content streaming are higher than the traditional 2D content streaming. Mainly, this is due to the larger number of displays and technologies that imply generation of multiple versions of the 3D video with different depth quality. In this section, we discuss 3D display technologies and their impact on the storage demands of 3D streaming systems.

3D display technologies can be classified into: (i) stereoscopic (require glasses), (ii) autostereoscopic (glasses-free), and (iii) autostereoscopic with eye tracking. Stereoscopic displays require wearing special glasses and they are categorized into *Active* and *Passive*. The Active category requires fast display that alternates between left and right views in synchronization with the glasses. Whereas the Passive category requires polarization filters on the display and polarization-based glasses for the viewer. The simplest

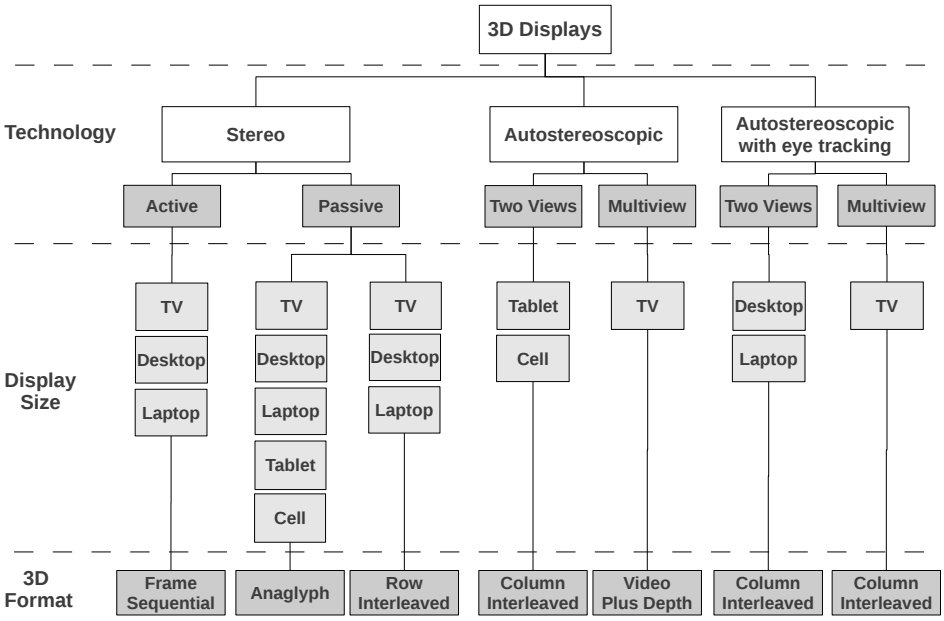

Figure 1: 3D Display Classification.

case of the Passive stereoscopic technology requires anaglyph glasses and an anaglyph image. An anaglyph image is composed of two differently-colored images, one for each eye.

On the other hand, autostereoscopic displays do not require any special glasses. Instead they use optical elements on the display that either block or reflect light to one of the two eyes. These optical elements are called parallax barriers and lenticular sheets. Two types of autostereoscopic displays exist: (1) Two views, and (2) Multiview. A multiview display allows multiple viewers to watch the display at the same time and each of them can see a different perspective according to the viewing angle. Because of their convenience, autostereoscopic displays have been used in 3D laptops, tablets and smartphones. The third category of 3D display technologies is the autostereoscopic with eye tracking in which the viewer's viewing angle is tracked in order to direct the light to his/her eyes for a better 3D experience.

Figure 1 shows our classification of 3D displays. As shown in the figure, 3D displays have a wide range of technologies. For each technology, displays with different sizes are available as depicted in the second level of the figure. The third level in the figure shows that for each display technology, the 3D content needs to be converted to a specific 3D format. Different types of 3D formats are summarized in Table 1. As a result of the above, the same 3D content should be stored many times with different formats in order to be visible on all 3D displays. In addition, 3D content should be adjusted to enhance 3D experience for different display sizes. Furthermore, in dynamic network conditions where the bandwidth varies with time, rate adaptation may be needed. Rate adaptation will require even creating more versions for each 3D video.

In summary, the challenges of designing efficient 3D streaming systems include:

- Supporting many 3D display technologies and sizes, as shown in Figure 1, in order to increase viewership and satisfaction. The wide range of 3D display technologies and sizes makes it hard for content providers to support all these variations.
- Large storage requirements. 3D streaming systems should store many versions of the same 3D content according to the technologies and sizes they support.
- Rate adaptation of views and depth due to varying network conditions. 3D streaming systems should keep the perceived 3D quality and smooth experience, while adapting to dynamic network conditions.

3.2 Proposed Architecture

To support most 3D display sizes and technologies described in Section 3.1, we propose a generalized structure to orchestrate the operations for generating various enhanced 3D versions, which we call 3D Version Tree. Each 3D version is a copy of the original 3D video, yet optimized for a specific class of displays.

As shown in Figure 2, the tree root at Level 0 is an input 3D video with no operations at this level. The next level in the tree is the Size Retargeting Operations, which process the input 3D video depending on the target size. These operations may span scaling up/down, cropping, and depth enhancement. The output of this level is size-specific N versions. These versions represent the input for the next level of operations, which is the 3D Technology Retargeting Operations. These operations enable a 3D viewing experience on different 3D display technologies, such as anaglyph, row interleaving, and depth estimation operations. The output of these operations is technology-specific M versions.

To create a 3D version at Level 2, its parent version should be created first at Level 1. For example, to create a 3D version for a large stereoscopic display, the depth enhancement size retargeting operation is executed first at Level 1. Then, the output of this operation is the input for row interleaving technology retargeting operations at Level 2. The order of these operations is important to preserve good 3D quality for the users. Technology Retargeting Operations result in

Format	Description
Side By Side	The width of the frame is double that of either left or right image, and the left and right frames are stacked with each other horizontally such that the left half of the frame is the left view and the right half is the right view.
Top Bottom	The height of the frame is double that of either left or right, and the left and right frames are stacked with each other vertically such that the top half of the image is the left view and the bottom half is the right view.
Frame Sequential	The left and right views are sent separately one after the other, this format is used in active stereo displays where the displays alternate between left and right typically at frequency of 120 Hz.
Anaglyph	The left and right views are multiplexed over each other with different color for each view, typically red and blue. A special glasses with the same color filter the red color in the left view and the blue color in the right view resulting in one image for each eye.
Row-Interleaved	The left and right views are interleaved with each other horizontally row by row such that odd rows belong to the left view and the even rows belong to the right view.
Column-Interleaved	The left and right frames are stacked with each other vertically column by column such that odd columns belong to the left view and the even columns belong to the right view.
Video Plus Depth	The video is encoded in 2D and a separate depth map is created for the 2D video. This format is used in multiview displays because the depth map allows the creation of many virtual (synthesized) views, which adds flexibility and supports wider viewing angles for users.

Table 1: Summary of current 3D video formats.

changing (i) pixels ordering, or (ii) color channels. Size Retargeting Operations require the original frame as an input. As a result, we execute Size Retargeting Operations first, then execute Technology Retargeting Operations. Also, our tree structure is flexible and can support different aspects of streaming systems. For example, we can add more levels to support multiple-bitrate versions and multiple container formats.

There is a trade-off between supporting large range of displays in the market, and the storage requirements of different versions. Specifically, to support all of the current displays, our system needs to store at least N x M different versions. As discussed in Section 3.1, we have five 3D display sizes and seven different display technologies. As a result, we have to store up to 35 versions to support all 3D displays.

Figure 3 depicts the high level architecture of the proposed 3D streaming system. The proposed system adopts the client-server architecture. At client side, a user sets his display preferences. Mainly, these are the display technology and size. A client application can also be used to automatically collect such information. When a client requests a specific version, an HTTP request is sent to the server which generates the version based on the request. Meanwhile, a DASH-based manifest file URL is constructed and posted-back to the designated client. The client fetches and parses the manifest file and downloads the required video segments accordingly.

The server side consists of Execution Manager, HTTP Web Server, and Storage Manager. At system initialization, the server generates Level 1 versions and stores them into system's storage. The Execution Manager intercepts clients' requests and decides whether to generate new versions if they do not exist. It also tracks the versions history and popularity. The interception routine works as follow.

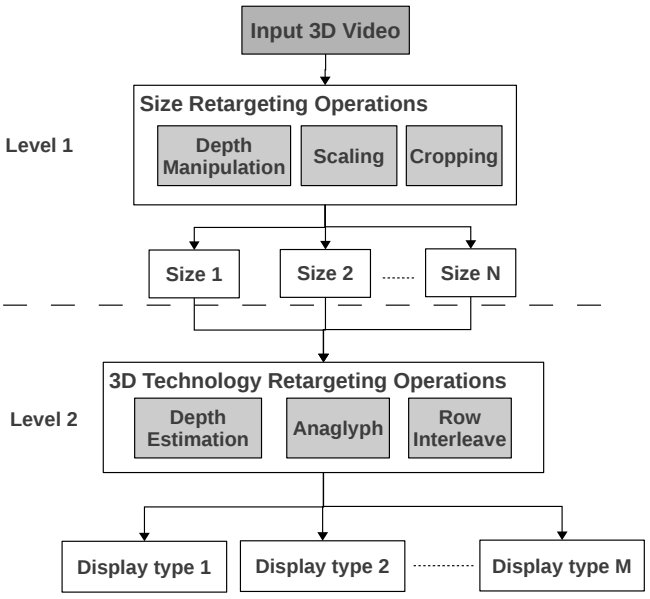

Figure 2: The proposed 3D Version Tree for managing the creation of different versions of 3D videos.

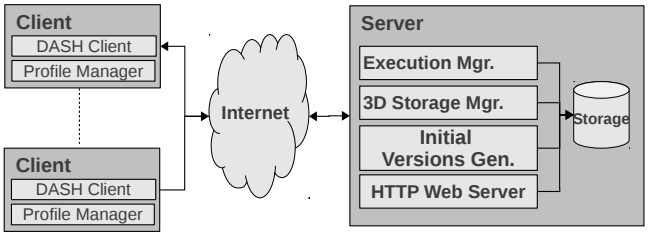

Figure 3: High level architecture of the proposed 3D streaming system.

Symbol	Description
β	Storage importance factor, integer in the range $[1-10]$
c_i (cycles)	CPU cycles to generate the initial segment of version v_i
CPU Resources	Number of CPU cores, clock speed and system memory
h_i	Popularity (number of requests) of version v_i
L (sec)	Maximum waiting time to create any 3D version in real time
N	Average number of requests per second
p_i (sec)	Processing time to generate the initial segment of version v_i
S (Bytes)	Available storage size
T (sec)	Time required before start streaming

Table 2: Symbols used in the formulation and solution of the storage optimization problem.

First, the routine tests whether the requested version exists. If the version exists, the routine posts-back the associated URL of the manifest file directly. Otherwise, it constructs the associated manifest file and starts generating video segments. To reduce the waiting time in the client-side, a new *processing* thread is spawned to generate the required video segments, while the URL of the manifest file is posted-back immediately. In the processing thread, the required retargeting operation is executed. Then, DASH compatible segments are generated and the associated manifest file is updated accordingly.

We design the 3D Storage Manager to optimize the system storage using versions' popularity as well as storage and processing demands. The Storage Manager runs periodically to decide on storing or removing versions. We describe the 3D Storage Manager in Section 4.

4. PROPOSED STORAGE OPTIMIZATION ALGORITHM

In this section, we define the problem of storage management in 3D streaming systems. Then, we propose a storage optimization algorithm for 3D streaming systems.

4.1 Problem Statement and Formulation

The 3D streaming system, described in Section 3, needs to store many versions besides the original video to support different 3D displays. This results in a substantial increase in the storage usage.

We define the storage management problem as follows:

PROBLEM 1 (STORAGE OPTIMIZATION). *Consider a set of 3D display sizes and technologies and a set of 3D versions $V_{input} = \{v_1, v_2, \ldots, v_n\}$. Each version v_i has popularity, size and processing requirements to create it. Find the set of versions $V_{output} \subseteq V_{input}$ that minimizes the storage usage such that the waiting time of users requesting videos does not exceed a given upper bound and the system available processing resources are not exceeded.*

To formulate and solve the above problem, we start by describing the system model and define several parameters used in the formulation. For quick reference, Table 2 lists all symbols used in the formulation. From the perspective of the storage optimization algorithm, the time is divided into equal-length periods. The period length is in the order of hours or days, depending on the scale and dynamics of the 3D streaming system. Each period has an index, which is denoted by t, where $t = 0, 1, 2, \ldots$. During each period, viewers request different versions of the 3D videos in the system. The total number of requests for a specific 3D version v_i in all previous periods up to the current one is denoted by h_i. We use h_i to indicate the popularity of various versions of the 3D videos in the system. In addition, we denote the size needed to store version v_i by s_i.

Basically, the storage optimization algorithm uses information from previous periods to decide on the 3D versions that should be pre-created and stored and the ones that should be created on-demand in order to minimize the storage cost. The decision is constrained by the available processing capacity in the system as well as the storage cost and capacity. The total storage capacity in the system is denoted by S. The relative cost of the storage system compared to other costs such as processing and network costs varies from one streaming system to another. To capture this relative cost, we define the parameter β as the storage importance factor, which can be used by system administrators to control the importance of optimizing the cost usage relative to other resources (mainly CPU resources). β is an integer value in the range $[1-10]$, where 1 means that the storage is less important to optimize, and 10 indicates that the storage is fairly costly either because of the limited space or because of the abundance of CPU resources that are sitting idle.

Since some 3D versions will be created on demand in real time, the available computing resources need to be considered in our model. We define an upper bound on the time taken to generate any 3D version in real time, which is denoted by L and it is in the order of few seconds. The maximum time latency L ensures the responsiveness of the 3D streaming system. The processing cost to generate the 3D version v_i is denoted by c_i, which is expressed in terms of CPU cycles. Since we use DASH-based streaming, each video version is divided into segments. And these segments are served to the user sequentially, not all at once. Thus, c_i is considered to be the cost of generating the *initial* segment of version v_i, not the cost of generating all segments. Once the initial segment is generated, it will be served to the user, and the following segments will be successively created while the user is viewing earlier segments. We note that the costs of generating different versions are quite different. For example, the cost of generating a video plus depth 3D version from an input stereo video is substantially larger than the

cost of generating row-interleaved or anaglyph version from the same stereo video input.

In addition, in our model we consider the number of CPU cores in the system as well as the available memory. Considering the memory is important because the video conversion operations are memory intensive. Thus, the system may not be able to use all available CPU cores at the same time, because or memory limitations. We denote the CPU processing time to generate the initial segment of the 3D version v_i by p_i and it is calculated as follows:.

$$p_i = \frac{c_i \times T}{\zeta/(\rho \times N)}, \tag{1}$$

where N is the average number of user requests per second that arrive to the 3D streaming system and it is a parameter set by the system administrator; T is the time required before start streaming in seconds; ρ is the average percentage of new 3D versions to be generated; and ζ is the effective number of CPU cycles per second. Assuming that the current period in the system is $t+1$, ρ is estimated from the previous period t as follows:

$$\rho = \frac{\sum_j H^t(j) Y_j^t + \sum_i H^t(i) X_i^t}{\sum_j h_j^t + \sum_i h_i^t}, \tag{2}$$

where Y^t and X^t are binary vectors representing stored versions from period t for Level 1 and Level 2, respectively. $H^t(i)$ is a binary function that we define as:

$$H^t(i) = \begin{cases} 0 & h_i = 0 \\ 1 & h_i > 0 \end{cases} \tag{3}$$

The effective number of CPU cycles per second ζ is given by:

$$\zeta = \min(num\_cores \times clock\_speed, \frac{available\_RAM}{\max required\_RAM} \times clock\_speed). \tag{4}$$

The parameter ζ limits the number of utilized CPU cores when the available system memory is low, where $\max required\_RAM$ is the maximum system memory required by a retargeting operation to execute.

We model our version management strategy as an optimization problem, with the goal to minimize the required storage subject to available processing resources and initial delay time. Specifically, our model balances the storage cost of version v_i when it is stored and processing cost to generate version v_i.

With the above model description and definitions, we mathematically state the storage optimization problem as follows:

$$\text{minimize} \sum_j \sum_i \frac{\beta}{S} \times s_{ij} \times (1 - x_{ij}) + \sum_j \sum_i \frac{\log(h_{ij}+1)}{\sum_i L} \times (p_i + p_j) \times x_{ij} \tag{5}$$

subject to

$$\sum_j \sum_i s_{ij} \times (1 - x_{ij}) \leq S \tag{6}$$

$$(p_i + p_j) \times x_{ij} \leq L \tag{7}$$

$$p_j = x_{1j} \times delay_j \tag{8}$$

where

$$x_{ij} = \begin{cases} 0 & \text{version } ij \text{ is stored} \\ 1 & \text{otherwise} \end{cases} \tag{9}$$

where x_{ij} is the optimization problem decision variable and j and i represent Level 1 and Level 2 versions in the 3D Version Tree, respectively. Equation (5) is the objective function, which includes the storage cost for saving version v_{ij} and processing cost for executing operations to generate a 3D version v_{ij}. In particular, the storage cost represents the version size in bytes s_i normalized by the available storage S. We chose to normalize by S to preserve the balance between the used and available system storage. The processing cost considers the processing delay and popularity, where it is normalized by summation of the maximum delay over all versions. We design our optimization model to consider the processing cost for a Level 2 version when its parent version (at Level 1) does not exist. Equations (6) to (8) represent storage, maximum waiting time, and parent processing constraints, respectively. Particularly, the total storage requirements of stored versions must not exceed the available storage S. Also, the total processing time to generate a new version is bounded by the maximum waiting time L. By substituting Equation (8) in Equation (5), the optimization problem is quadratic in x.

4.2 Proposed Solution

The quadratic optimization model presented in the previous section is computationally expensive to solve for large-scale systems, in which the number of videos is large. To address this complexity, we propose an algorithm that computes the solution much faster and produces near-optimal solutions. We relax our original optimization model into two consecutive linear programming models. The first one optimizes the storage in Level 1 of the versions tree as follows:

$$\text{minimize} \sum_j \frac{\beta}{S} \times s_j \times (1 - y_j) + \sum_j \frac{\log(h_j+1)}{\sum_j L} \times p_j \times y_j \tag{10}$$

subject to

$$\sum_j s_j \times (1 - y_j) \leq S \tag{11}$$

$$p_j \times y_j \leq L \tag{12}$$

where

$$y_j = \begin{cases} 0 & \text{version } j \text{ is stored} \\ 1 & \text{otherwise} \end{cases} \tag{13}$$

where y is the Level 1 optimization decision variable. The second optimization problem optimizes the storage in Level 2 given the output of the first linear model:

$$\text{minimize} \sum_j \sum_i \frac{\beta}{S} \times s_{ij} \times (1 - x_{ij}) + \sum_j \sum_i \frac{\log(h_i + 1)}{\sum_i L} \times (p_i + p_j y_j) \times x_{ij} \quad (14)$$

subject to

$$\sum_j \sum_i s_{ij} \times (1 - x_{ij}) \leq S - \sum_j s_j \times y_j \quad (15)$$

$$(p_i + p_j \times y_j) \times x_{ij} \leq L \quad (16)$$

where

$$x_{ij} = \begin{cases} 0 & \text{version } ij \text{ is stored} \\ 1 & \text{otherwise} \end{cases} \quad (17)$$

Equations (15) and (16) represent the storage and maximum waiting time constraints in the Level 2 optimization problem, respectively. In Equation (15), we subtract the used storage in Level 1 optimization problem $\sum_j s_j \times y_j$ from the available storage S. This guarantees that the solver satisfies storage requirements for Level 1 and Level 2. In Equation (16), p_j processing delay is included when the parent version j is not stored and needs to be generated.

We implemented a brute-force solver for the quadratic programming model. Our solver generates all solutions and searches for the solution with the minimum objective function value and satisfies the constraints. We compared the brute-force solver results with the relaxed model described above for small dataset. The difference between the outputs of the two approaches is small. However, the running time of the brute-force solver is large. For example, we solved the optimization problem in Equation (5) for 15 3D versions in two hours. On the other hand, our relaxed optimization problem in Equation (10) and Equation (14) finished in less than two seconds for the same dataset. For larger systems, the problem can not be solved on a practical time scale.

Algorithm 1 describes the proposed algorithm. The algorithm runs periodically to decide on storing 3D versions. We use vector notation to indicate vector operations. From previous period t information, we construct two binary vectors $\vec{Y}^t$ of size M_1 and $\vec{X}^t$ of size M_2 that represent whether Level 1 and Level 2 version is stored, respectively. Then, we calculate the percentage of expected new versions to be processed ρ, number of utilized CPU cores ζ and storage weighting parameter. We compute the processing time vector for Level 1 versions $\vec{p}_j$ and processing weighting parameter vector. Thereafter, we run the optimization solver for Level 1 versions whose output is the vector $\vec{Y}^{t+1}$. Besides $\vec{Y}^{t+1}$, the processing time vector $\vec{p}_i$ and processing weighting parameter vector are inputs for Level 2 optimization solver. Vectors $\vec{Y}^{t+1}$ and $\vec{X}^{t+1}$ are the decision variables of versions removal for period $t + 1$.

The time complexity of the proposed algorithm is calculated by summing the time complexities of constructing $\vec{Y}^t$ and $\vec{X}^t$ vectors, computing internal parameters and solving two linear programming optimization problems. This can be described by:

Algorithm 1 Proposed Storage Optimization Algorithm

Input: M_1 Total number of versions in Level 1
Input: M_2 Total number of versions in Level 2
Input: $\vec{V}_N$ A set of stored versions for period t of size N
1: $\vec{Y}^t = \text{get\_Y}(\vec{V}_N)$
2: $Y\_stored = \text{size}(\vec{Y}^t)$ where $\vec{Y}^t_j = 0$
3: $\vec{X}^t = \text{get\_X}(\vec{V}_N)$
4: $X\_stored = \text{size}(\vec{X}^t)$ where $\vec{X}^t_i = 0$
5: $total\_history = \text{sum}(\vec{h}_j) + \text{sum}(\vec{h}_i)$
6: $\rho = (Y\_stored + X\_stored)/total\_history$
7: $\zeta = \min(cpu\_cores, available\_ram/max\_ram)$ x $clock\_speed$
8: $\vec{p}_j = (\vec{c}_j \times T)$ / $(\zeta$ / $(\rho \times N))$
9: $\vec{\alpha}^j = log(\vec{h}_j + 1)$ / $(M_1$ x $L)$
10: $\vec{Y}^{t+1} = \text{run\_optimizer\_level\_one}(\rho, \zeta, \vec{p}_j, \beta$ / $S, \vec{\alpha}^j)$
11: $\vec{p}_i = (\vec{c}_i \times T)$ / $(\zeta$ / $(\rho \times N))$
12: $\vec{\alpha}^i = log(\vec{h}_i + 1)$ / $(M_2$ x $L)$
13: $\vec{X}^{t+1} = \text{run\_optimizer\_level\_two}(\vec{Y}^{t+1}, \rho, \zeta, \vec{p}_i, \beta$ / $S, \vec{\alpha}^i)$
14: remove Level 1 versions where $\vec{Y}^{t+1}_j = 1$
15: remove Level 2 versions where $\vec{X}^{t+1}_i = 1$

$$\text{Time Complexity} = O(\text{construct Y}) + O(\text{construct X}) + O(\text{parameters}) + O(\text{Level 1 solver}) + O(\text{Level 2 solver})$$

The time complexity for vectors construction and parameters' calculation is $O(M_1)+O(M_2)$. Thus, the above formula can be reduced to:

$$\text{Time Complexity} = O(M_1) + O(M_2) + O(\text{Level 1 solver}) + O(\text{Level 2 solver})$$

The time complexity of optimization solver depends on the used algorithm. In our system, we use the Simplex algorithm [14], which is fast for most practical applications. In addition, the work by Spielman and Teng [18] shows that the Simplex algorithm has polynomial time complexity using smoothed analysis.

5. EVALUATION

In the following, we present a proof of concept prototype of the proposed 3D streaming system. Then, we evaluate the proposed storage optimization algorithm. First, we describe the used dataset, storage optimization parameters and performance metrics. Then, we present the experimental results that show the effectiveness of the algorithm. We study the effect of maximum delay variation on storage usage. Also, we vary the request rate and examine its effect on the storage. We show the importance of our algorithm by comparing it with storing Level 1 versions. Finally, we conduct experiments to show the impact of processing resources and storage cost on the storage usage and gain.

5.1 Proof of Concept Prototype

We implemented the streaming system as described in Section 3. The server-side is implemented as follows. We developed user-space operations framework using C++ and OpenCV [4]. Our framework provides stable interfaces, input and output management, state management and inputs

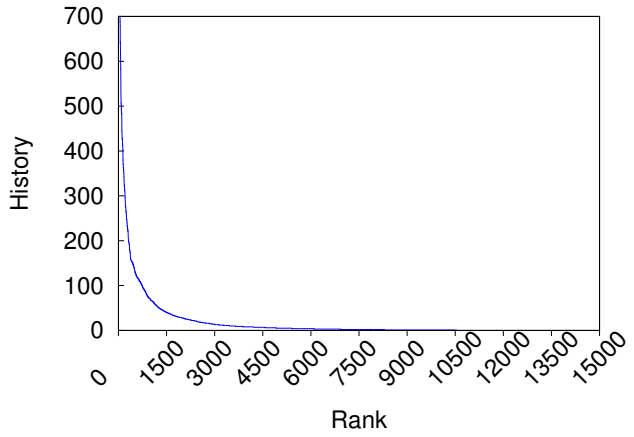

Figure 4: Version popularity distribution.

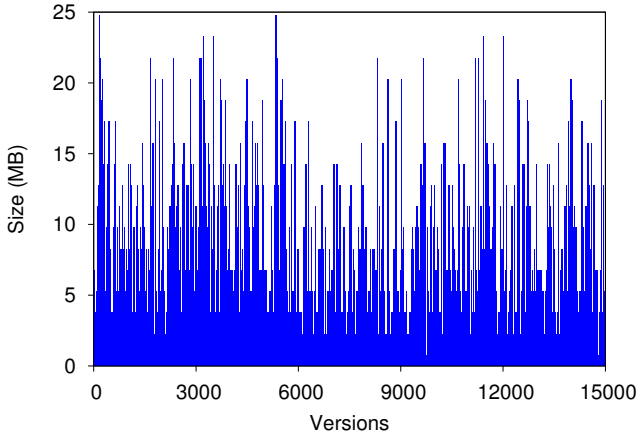

Figure 5: Version size histogram.

and outputs propagation to and from operations. Then, we implemented the operations of depth manipulation, anaglyph, row interleaving and depth estimation.

We used Apache Server [8] as an HTTP web server. We activated the `mod_headers` module in the Apache Server to control specific HTTP headers such as Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers and Accept-Ranges. Moreover, we implemented our storage optimization algorithm described in Section 4 using Python. We deployed our system on the cloud using Amazon Web Services [7].

For streaming purposes we adopt the Dynamic and Adaptive Streaming over HTTP (DASH) protocol [9, 19, 17]. DASH is an adaptive streaming technology, which provides a standard approach to enable audio and video streaming over HTTP. Specifically, DASH comprises two components; encoded audio/video streams called Media Presentations and a manifest file called Media Presentation Description (MPD). MPD is an XML file which describes the Media Presentations and the alternative streams according to bandwidth. DASH has several advantages over previous streaming protocols and technologies. First, streaming using HTTP servers is cost effective due to the existence of off-the-shelf web servers. Second, most firewalls allow HTTP connections unlike other streaming protocols. Third, video segments are considered as regular files, which can be cached and deployed in Content Delivery Networks (CDNs). Fourth, rate adaptation helps the client to switch between available bitrates depending on the varying network conditions, hence, delivering the best viewing experience. Fifth, the existence of a standard makes it easy for content providers to implement streaming infrastructures.

At the client-side, we developed two applications. The first one is a web client that uses HTML5 and DASH-JS [16]. DASH-JS is a JavaScript DASH library for the Google Chrome web browser. The second client is an Android client for HTC and LG phones, where we implemented a DASH client from scratch using Java and Android SDK [1]. We implemented MPD parser, HTTP client, and segment-based renderer. We tested our system using different 3D display sizes and technologies. Several 3D videos were processed and tested on all displays using the clients described before. The 3D quality was visually verified.

Property	Value
Mean	4.47
SD	4.5
Median	2.89
Mode	1.31
Maximum	24.5
Minimum	0.06

Table 3: Statistical properties in MB of size distribution.

5.2 Evaluation of the Storage Optimization Algorithm

We implemented a simulator to stress our storage optimization algorithm (SOA). We assume a single server setup with commodity processing resources. The server runs our algorithm daily to decide on storing or removing 3D versions. To evaluate our storage optimization algorithm using reasonable dataset, we downloaded $10,000$ 3D videos from YouTube. For each video, we gathered data about its duration, byte size, viewing history, frame rate and category. We used this information to synthesize our dataset. We generated a random dataset of $15,000$ 3D versions using the same histograms of the downloaded videos. Figures 4 and 5 and Table 3 show the statistical distributions and properties of our dataset. Figure 4 depicts the version popularity distribution, which follows the Zipf distribution. Figure 5 presents the version size distribution in our dataset. The mean size is 4.47 MB and the standard deviation is 4.5 MB. Table 3 lists more statistical properties of the distribution.

We set the parameters of our algorithm as listed in Table 4. We set the available storage S and maximum waiting time L as 100 GB and 5 seconds, respectively. The request rate N is 100 requests per second and the storage cost β is 5. We assume having a single server with 8 CPU cores and 12 GB of memory. These values result in versions processing timings equal to 2.1, 1.1, 0.2 and 0.1 seconds for depth estimation, depth manipulation, row interleave and anaglyph, respectively. These processing times were obtained empirically by measuring the time to convert a stereoscopic input video to the requested video version.

We use the storage in GB as a performance metric to show the effectiveness and importance of our storage optimization

Parameter	Value
L	100 GB
S	5 seconds
N	100 requests per second
β	5
CPU cores	8
System Memory	12 GB

Table 4: Used algorithm parameters throughout our experiments.

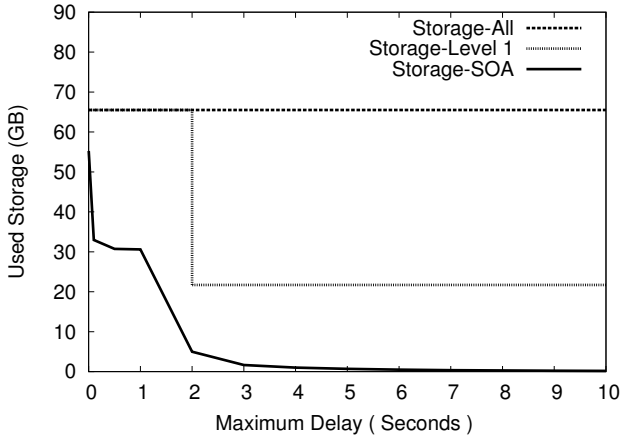

Figure 6: Effect of Maximum Waiting Time on Storage.

algorithm. In each experiment, we measure the total storage used by our algorithm. We use the same dataset described before for all experiments and same algorithm parameters unless stated otherwise. Running the same experiment multiple times does not affect the results, since we are using a deterministic mathematical model.

5.3 Effect of Maximum Waiting Time (L)

In this experiment, we study the impact of the maximum delay on the available storage. Figure 6 shows the results of applying our algorithm while varying the maximum delay. When L is less than 0.1 seconds, our algorithm decides to store up to 85% of the versions since the lowest processing time in our experiments is 0.1 seconds. If L is increased to be 0.5 seconds, only 47% of the versions are stored because two versions, row interleave and anaglyph, can be processed in less than 0.5 seconds. Generally, setting L to a value x allows our algorithm not to store a version whose processing time is less than x.

This experiment presented the effectiveness of our algorithm to optimize storage usage with respect to client waiting time. Decreasing waiting time gives larger value to the objective function, hence, our algorithm decides to store more versions to minimize the overall cost. By storing more versions, our system is able to satisfy tight time requirements. On average, our algorithm achieves up to 105.4x storage gain compared to storing all versions, while satisfying available processing resources and maximum waiting time.

5.4 Effect of Request Rate (N)

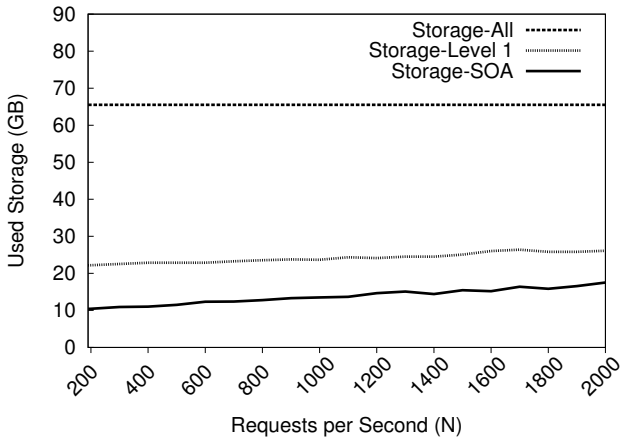

Figure 7: Request Rate Effect on Storage.

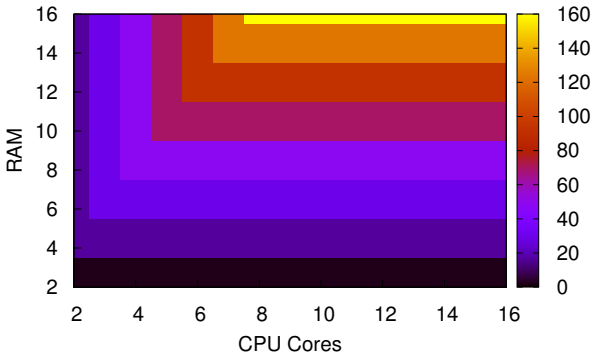

Figure 8: Achieved storage gain when changing CPU cores and system memory.

We simulate the case of increasing the request rate N. We measure the extra storage in GB required to handle the request rate increase compared to an initial storage. We generate clients' requests for the next day depending on versions popularity. For generation, we sample requests from a Zipf distribution whose parameter s is equal to 2.3. We estimate the parameter s from our dataset described before.

We compare the storage requirements of our storage optimization algorithm with storing all versions and storing Level 1 versions, while increasing request rate. The distinction between the three approaches is that our algorithm uses version's information to decide on storing it, while the other approaches stores versions *blindly* without considering version's popularity, processing and storage requirements.

Figure 7 depicts the comparison of our algorithm and the other approaches. Compared to storing all versions when the request rate increases, our algorithm stores the most popular versions that are likely to be watched again. Our algorithm outperforms storing all versions by 400%. In addition, our algorithm outperforms storing Level 1 versions by 2.3x on average. For the same dataset and clients' requests, our algorithm requires 17.5 GB of storage, whereas storing Level 1 versions requires 26.1 GB.

5.5 Trade-off between Storage and Processing Resources

The availability of CPU and memory resources affects the storage usage. In our algorithm, we incorporate available processing resources such that they are shared among versions to be processed. In this experiment, we vary the number of CPU cores and system memory (RAM) and measure the storage usage. We do not increase the number of CPU cores more than 16 cores for two reasons. First, we assume one single server setup with commodity hardware. Second, increasing the number of CPU cores affects the storage usage until reaching a limit depending on our versions' processing requirements. Figure 8 shows a heat map of the storage gain while increasing the CPU cores and system memory. Both storage usage and gain are 3D functions with respect to CPU cores and memory. We project their values as heat maps for convenience. In a heat map, large values are *hot* thus are colored with red and yellow; they appear as lighter areas in the figure. On the other hand, small values are considered *cold* and colored with blue, where they are represented by the darker areas. On average, our algorithm achieves 36.2x storage gain when varying processing resources.

The results of this experiment show two aspects of our algorithm. First, our algorithm adaptively stores more versions in case of shortage in processing resources. Second, the algorithm uses the smallest number of CPU cores that satisfies memory requirements. If there are more available CPU cores than memory, our algorithm does not use them. This is important to avoid unnecessary memory swapping. For example, when the system memory is set to 12 in Figure 8, we achieve larger storage gain as increasing the number of CPU cores increases until this number reaches 6 CPU cores. If there is more available memory than CPU cores, our algorithm uses the required memory only to avoid cores' over-subscription which leads to extra context switching.

5.6 Impact of Storage Cost (β)

We study the effect of the storage cost β on the performance of our algorithm. β is a user-defined parameter that determines the storage importance of a version compared to its processing requirements. In our algorithm, it is an integer parameter whose value ranges from one to ten. Figure 9 shows the result of this experiment. Small β values mean that the storage is inexpensive, which guides our algorithm to store more versions. On the contrary, large β values guide the algorithm to remove more versions since the storage is expensive. Costly storage does not necessarily imply price per storage unit; it may mean storage availability.

This experiment verifies our algorithm behavior, when modifying the storage cost parameter. Specifically, our algorithm achieves storage gain up to 360x and 141x on average. The storage cost parameter can be viewed as an external parameter that guides our algorithm to decide on storing versions depending on the storage physical state, storage price, or storage availability.

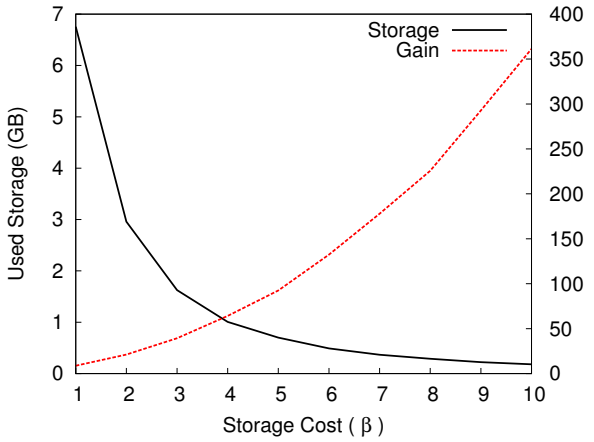

Figure 9: Impact of storage cost parameter on storage usage.

6. CONCLUSIONS AND FUTURE WORK

We presented the design of a general architecture for a 3D streaming system. The goal of this system is to increase the viewership of 3D content, by serving various 3D versions optimized to different display technologies and sizes. We focused on the problem of storage management in this 3D streaming system. We proposed a storage optimization algorithm. The algorithm decides on storing a 3D version depending on its popularity as well as storage and processing requirements. The algorithm considers system-aspect constraints such as limited storage capacity, limited processing resources and tight Quality of Service (QoS) conditions. We developed server-side components that process the original 3D videos and execute the storage algorithm. We implemented the operations of depth manipulation, anaglyph, row interleaving and depth estimation. We deployed these server-side components on the cloud using Amazon Web Services. Several videos have been processed using our system and tested on different displays.

We evaluated the performance of the storage optimization algorithm using simulation. The results of the simulation show that the algorithm: (1) can achieve storage gain up to 360x, (2) meets tight Quality of Service (QoS) requirements, and (3) handles the case of high request rate efficiently.

In the future, we plan to extend our algorithm to manage storage of multi-bitrate 3D videos, and videos with different container formats.

7. REFERENCES

[1] Android SDK . `http://developer.android.com/sdk/index.html`.

[2] 3DeeCentral. `http://www.3deecentral.com/`.

[3] 3DVisionLive. `https://www.3dvisionlive.com/`.

[4] OpenCV. `http://opencv.org/`.

[5] Trivido. `http://www.trivido.com/`.

[6] Youtube. `http://www.youtube.com/`.

[7] Amazon.com . Amazon Web Services . `http://aws.amazon.com/`.

[8] The Apache Software Foundation . Apache, HTTP Server Project . `http://httpd.apache.org/`.

[9] ISO/IEC 23009-1:2012. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats.

[10] M. Johanson. Stereoscopic video transmission over the internet. In *Proc. of IEEE Workshop on Internet Applications (WIAPP'01)*, pages 12–19, Washington,

DC, July 2001.

[11] H. Kimata, K. Fukazawa, A. Kameda, Y. Yamaguchi, and N. Matsuura. Interactive 3d multi-angle live streaming system. In *Proc. of IEEE International Symposium on Consumer Electronics (ISCE'01)*, pages 576–579, Singapore, June 2011.

[12] E. Kurutepe, M. R. Civanlar, and A. M. Tekalp. Interactive transport of multi-view videos for 3dtv applications. *Journal of Zhejiang University SCIENCE A*, 7(5):830–836, May 2006.

[13] J.-G. Lou, H. Cai, and J. Li. A real-time interactive multi-view video system. In *Proc. of ACM International Conference on Multimedia*, pages 161–170, Singapore, November 2005.

[14] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[15] S. Pehlivan, A. Aksay, C. Bilen, G. B. Akar, and M. R. Civanlar. End-to-end stereoscopic video streaming system. In *Proc. of IEEE International Conference on Multimedia and Expo*, pages 2169–2172, Toronto, Canada, July 2006.

[16] B. Rainer, S. Lederer, C. Mueller, and C. Timmerer. A seamless web integration of adaptive http streaming. In *Proc. of European Signal Processing Conference (EUSIPCO'12)*, pages 1519–1523, Bucharest, Romania, August 2012.

[17] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE Multimedia Magazine*, 18(4):62–67, April 2011.

[18] D. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proc. of ACM Symposium on Theory of Computing (STOC'01)*, pages 296–305, Hersonissos, Greece, July 2001.

[19] T. Stockhammer. Dynamic adaptive streaming over http: standards and design principles. In *Proc. of ACM Conference on Multimedia Systems (MMSys'11)*, pages 133–144, San Jose, CA, February 2011.

[20] B. Xin, R. Wang, Z. Wang, W. Wang, C. Gu, Q. Zheng, and W. Gao. Avs 3d video streaming system over internet. In *Proc. of IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC'12)*, pages 286–289, Hong Kong, August 2012.