MASH: A Rate Adaptation Algorithm for Multiview Video Streaming over HTTP

Khaled Diab
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

Mohamed Hefeeda
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

***Abstract*—Multiview videos offer unprecedented experience by allowing users to explore scenes from different angles and perspectives. Thus, such videos have been gaining substantial interest from major content providers such as Google and Facebook. Adaptive streaming of multiview videos is, however, challenging because of the Internet dynamics and the diversity of user interests and network conditions. To address this challenge, we propose a novel rate adaptation algorithm for multiview videos (called MASH). Streaming multiview videos is more user centric than single-view videos, because it heavily depends on how users interact with the different views. To efficiently support this interactivity, MASH constructs probabilistic view switching models that capture the switching behavior of the user in the current session, as well as the aggregate switching behavior across all previous sessions of the same video. MASH then utilizes these models to dynamically assign relative importance to different views. Furthermore, MASH uses a new buffer-based approach to request video segments of various views at different qualities, such that the quality of the streamed videos is maximized while the network bandwidth is not wasted. We have implemented a multiview video player and integrated MASH in it. We compare MASH versus the state-of-the-art algorithm used by YouTube for streaming multiview videos. Our experimental results show that MASH can produce much higher and smoother quality than the algorithm used by YouTube, while it is more efficient in using the network bandwidth. In addition, we conduct large-scale experiments with up to 100 concurrent multiview streaming sessions, and we show that MASH maintains fairness across competing sessions, and it does not overload the streaming server.**

I. Introduction

Recently, there has been significant interest in streaming immersive multimedia content such as multiview and virtual reality (VR) videos. These videos consist of multiple views of the same scene but captured by several cameras at different angles. For example, YouTube [1] released a small-scale experiment to stream multiview content in early 2015, in which a user can experience the scene (a concert in that experiment) from different perspectives by switching among various views. Two of the main challenges in streaming this complex multiview video content over the Internet are adaptation to dynamic network conditions and supporting user interactivity; which we address in this paper.

Current popular streaming services such as YouTube and Netflix employ adaptive streaming over HTTP, using standards such as DASH [2]. In such services, the video is encoded into multiple quality levels at different bitrates. Each quality level is divided into segments of equal length. During the streaming session, clients request segments from the HTTP server. To handle varying network conditions, a *rate adaptation algorithm* is used by each client to dynamically request segments with the most suitable quality for the current conditions. The rate adaptation algorithm needs to balance multiple, sometimes conflicting, performance metrics including quality level, quality variations, and smoothness of playback [3], [4].

Unlike single-view videos, multiview videos require much more complex rate adaptation algorithms. This is because these algorithms need to fetch segments for active views as well as other views to enable the user to smoothly navigate across views. In addition, the video may have many possible views, and naively fetching segments from all of them can lead to significant waste in network bandwidth. On the other hand, fetching segments from only the active view and nothing from other views will introduce long delays when a user switches to another view, which can damage the immersive experience and lead to user dissatisfaction. Furthermore, users interact with multiview videos in different ways, based on their own interests and perspectives. Thus, predicting the view that may be needed next is not straightforward. These complications are all added to the challenge of handling the network dynamics. Although the rate adaptation problem for single-view videos has been extensively studied in the literature [5], [6], [4], [7], it has received little attention for the more complex multiview videos, which are expected to be quite popular in the near future given the huge interest and investments of major companies such as Google, Facebook, Microsoft, and Samsung.

In this paper, we propose a novel **M**ultiview **A**daptive **S**treaming over **H**TTP (MASH) algorithm. MASH introduces a new perspective to the rate adaptation problem in multiview video streaming systems: it constructs probabilistic view switching models and it utilizes these models in dynamically selecting segments of various views at different qualities, such that the quality and immersiveness of the videos are maximized while the network bandwidth is not wasted. Specifically, for each multiview video, MASH constructs *global* and *local* switching models. The global model captures user activities across all streaming sessions seen by the server so far, while the local model understands user activity during the current streaming session only. MASH combines the two models to

dynamically weigh the importance of each view given the one being watched. Then MASH uses a new buffer-based approach to select segments of different views according to their relative importance. The contributions of this paper are summarized as follows:

- We present view switching models based on Markov chains to capture user activities. We combine these models dynamically to prioritize the views (Section III).
- We propose a new buffer-based rate adaptation algorithm for multiview video streaming systems that use the HTTP protocol. We show that the proposed algorithm and view switching models impose negligible overheads and we analyze their convergence properties (Section IV).
- We developed a multiview video player and implemented our rate adaptation algorithm and view switching models in it. We conducted extensive empirical study to compare our algorithm versus the one used by YouTube for multiview videos, and the results show substantial improvements across multiple performance metrics and in different network conditions. For example, MASH achieves up to 300% improvement in the average rendering quality and up to 200% improvement in the prefetching efficiency compared to the YouTube algorithm. In addition, our results show that MASH achieves fairness across concurrent sessions, and it does not overload the streaming server. Furthermore, we compare MASH versus other rate adaptation algorithms, which are derived from current rate adaptation algorithms for single-view videos, and we show that MASH outperforms all of them. (Section V).

II. Related Work

We categorize rate adaptation algorithms to rate-based [8], [4], [9], [10], buffer-based [5] and hybrid [6] approaches. In rate-based approaches, the algorithm estimates the network capacity and chooses suitable segments accordingly. For example, FESTIVE [8] uses the harmonic mean to smooth the measured network capacity. It also uses randomized segment scheduling to mitigate differences in users join time. The probe and adapt algorithm [4] smoothes the estimated bandwidth using an approach similar to the TCP congestion control, and schedules segment requests to avoid buffer underflows. *Accurate* network capacity estimation is, however, difficult [11]. Buffer-based approaches, on the other hand, do not estimate the network capacity. They rather observe the current buffer occupancy (i.e., its length in time unit), and request low bitrate segments if the buffer occupancy is low and vice versa. For example, Huang et al. [5] integrated a buffer-based adaptation algorithm in a Netflix player, and they showed that their algorithm reduces the rebuffering events while preserving player video quality and stability. A hybrid approach [6] was proposed in the literature to combine both buffer occupancy and capacity estimation to improve the overall quality. This approach models rate adaptation as an optimal control problem, and solves it by enumerating pre-calculated solutions at the client-side. All of the above works address traditional single-view videos, and thus, they cannot handle multiview videos in which users switch among views. Rate adaptation algorithms for single-view videos will either waste network bandwidth if they prefetch all views or lead to playback interruptions if they prefetch the active view only.

Few recent works considered more complex videos, but none addressed our problem of rate adaptation for multiview videos, to the best of our knowledge. For example, Hamza and Hefeeda [12] proposed an interactive free-viewpoint video streaming system using DASH. The client fetches original views and generates virtual views at the client side when required. Their work, however, focuses on optimizing the quality of the synthesized virtual views using rate-distortion models. Su et al. [13] proposed a rate adaptation algorithm for 3D video streaming using the High Efficiency Video Coding (HEVC) standard. Unlike our work, they only address non-interactive 3D videos where users cannot choose the viewing angles. Another set of works address 3D tele-immersive systems (3DTI). 3DTI systems have different interactivity semantics and streaming models than our work. Such systems are multi-user, real-time, and object-oriented, while our work is concerned with video on demand streaming with view-level interaction by individual users. Thus, 3DTI systems address different problems such as object-level [14], frame-level [15] and perception-based [16] adaptation.

The closest approach to our work is the multiview player of YouTube, which is integrated in the Google Chrome web browser. It allows users to navigate to different views and it dynamically fetches segments of different views during the streaming session. We compare MASH against the algorithm used by YouTube. Furthermore, we modify recent buffer-based rate adaptation algorithms [5] to support multiview videos, and we compare against them.

III. Overview and Switching Models

A. Overview of MASH

We consider an adaptive streaming system based on HTTP using standards such as DASH [2], where a server streams *multiview videos* to diverse users over dynamic networks. A multiview video consists of N views, which enables users to experience the video from different angles. A user is allowed to interact with the video to choose the desired view at anytime. Supporting multiview videos is, however, challenging. First, the streaming system needs to support efficient view switching, without imposing long waiting delays (when switching happens, which could damage the viewing experience) or consuming excessive network resources (by naively sending all views to every user, even if most of them will not likely be viewed). Second, the streaming system needs to adapt to the dynamic network conditions of different clients and serve the best possible quality in such conditions. The proposed MASH algorithm addresses these challenges, by constructing probabilistic view switching models and using these models in a novel buffer-based rate adaptation method.

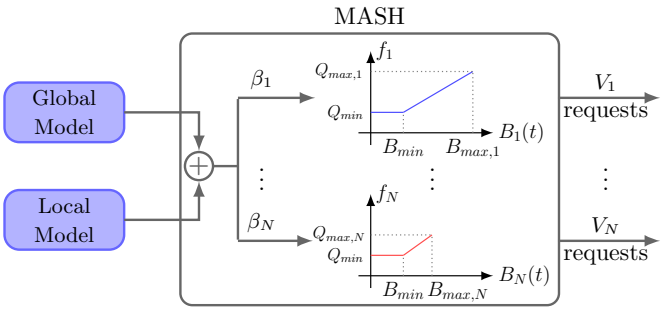


Fig. 1: High-level overview of MASH. MASH runs at the client side, uses global and local view switching models, and constructs buffer-rate functions f_i's to dynamically request segments from different views at different quality levels.

Following DASH principles, each view V_i is encoded into multiple quality levels $Q_i \in \{Q_{min}, \ldots, Q_{max}\}$ (measured in Mbps). Each quality level is divided into equal-length segments and stored at the server. At the client side, MASH decides which segments of which view(s) and at what quality levels should be requested from the server. When requested segments of view V_i arrive at the client, they are decoded and stored in the frame buffer corresponding to that view. We use $B_i(t)$ (in seconds) to denote the *buffer occupancy* of the frame buffer of view V_i at time t. We refer to the view being watched as the *active* view, while others are called *inactive* views. When a user switches to view V_j and it happened that the corresponding buffer is empty, the client will not be able to render V_j and playback interruption (or re-buffering) occurs. To minimize these re-buffering events, we design a composite model for predicting view switching, which captures the switching pattern of: (i) all users who watched this multiview video before (global model), and (ii) the user of the current streaming session (local model).

Figure 1 shows a high-level overview of MASH, which runs at the client side. MASH combines the outputs of the global and local view switching models to produce a relative *importance factor* β_i for each view V_i. MASH also constructs a *buffer-rate function* f_i for each view V_i, which maps the current buffer occupancy to the segment quality to be requested. The buffer-rate functions are dynamically updated during the session; whenever a view switch happens. As shown later, MASH strives to produce smooth and high quality playback for all views, while not wasting bandwidth by carefully prefetching views that will likely be watched.

B. View Switching Models

MASH combines the outputs of two stochastic models (local and global) to estimate the likelihood of different views being watched. We define each view switching model as a discrete-time Markov chain (DTMC) with N (number of views) states. An illustrative example is shown in Figure 2a for a video with four views. View switching is allowed at discrete time steps of length Δ. The time step Δ is the physical constraint on how fast the user can interact with the video. For example, we do not expect the user to switch views more than once in less than 100 ms, because of the limitations on the input interface (e.g., touch screen). In the following we describe the construction of each model. Then, we describe how we combine the two models together. We note that the construction of the switching models is carefully done to ensure convergence as well as minimize the imposed overheads on the system, as detailed in Section IV-B.

Local Model: It captures the user activities during the current streaming session, and it evolves with time. That is, the model is dynamic and is updated with every view switching event that happens in the session. The model maintains a count matrix $M(t)$ of size $N \times N$, where $M_{ij}(t)$ is proportional to the number of times the user switched from view V_i to V_j, from the beginning of the session up to time t. The count matrix $M(t)$ is initialized to all ones. Whenever a view switching occurs, the corresponding element in $M(t)$ is incremented. Increasing that element by 1, however, may result in wide fluctuations in the switching model, especially early in the session where all elements still have small values (typically 1). We use an exponential weighted moving average (EWMA) to smooth out these fluctuations. Thus, after switching from V_i to V_j, we set $M_{ij}(t) = \gamma M_{ij}(t-\Delta) + (1-\gamma)(M_{ij}(t-\Delta)+1)$, where γ is a smoothing factor. In our experiments, we set $\gamma = 0.2$. The count matrix $M(t)$ is used to compute the probability transition matrix of the local model $L(t)$, where $L_{ij}(t) = M_{ij}(t) / \sum_k^N M_{ik}(t)$. A row of $L(t)$ is denoted by $L_i(t)$, and it is a vector of size N that represents the conditional probability distribution $p(V_j|V_i)$ for every j (e.g., the probabilities in Figure 2a). At the end of the streaming session, the final transition matrix $L(t)$ is sent to the server to update the global model.

Global Model: This model aggregates users activities across all streaming sessions that have been served by the server so far. At beginning of the streaming session, the client downloads the global model parameters from the server. We use G to denote the transition matrix of the global model, where $G_{ij} = p(V_j|V_i)$ is the probability of switching to V_j given V_i. If this is the fist streaming session, G_{ij} is initialized to $1/N$ for every i and j.

Combined View Switching Model: The local and global model complement each other in predicting the (complex) switching behavior of users during watching multiview videos. For example, in some streaming sessions, the user activity may significantly deviate from the global model expectations, because the user is exploring the video from different viewing angles than most previous users have. Or the multiview video may be new, and the global model has not captured the expected view switching pattern yet. On the other hand, the local model may not be very helpful when the user has not had enough view switches yet, e.g., at the beginning of a streaming session. We combine the local and global models to compute an importance factor β_i for each view V_i as follows. We normalize the elements of the local and global transition matrices by subtracting the diagonal elements from them; for

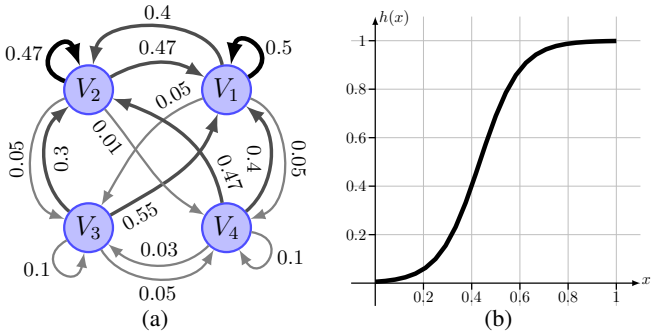


Fig. 2: (a) Simple example of the stochastic switching model used by MASH. (b) The sigmoid function used in assigning importance factors to different views.

$L(t)$ (and similarly for G):

$$\bar{L}_{ii}(t) = 0, \tag{1}$$

$$\bar{L}_{ij}(t) = L_{ij}(t)/(1 - L_{ii}(t)), \forall j \neq i, 1 \leq i, j \leq N. \tag{2}$$

These distributions are similar to the original ones with initializing the diagonal elements by zeros, and updating the counting matrix only when there is a view switch. Given the active view is V_i, we calculate its importance factor β_i as well as the importance factors β_j of all other views $V_j \, \forall j \neq i$ at time t as follows:

$$\beta_i = 1, \tag{3}$$

$$\beta_j = \alpha_i \times \bar{L}_{ij}(t) + (1 - \alpha_i) \times \bar{G}_{ij}, \forall j \neq i, \tag{4}$$

$$\alpha_i = h\left(E\left(\bar{L}_i\left(t\right), \bar{G}_i\right)\right), \text{and} \tag{5}$$

$$E(\bar{L}_i(t), \bar{G}_i)) = \sqrt{\frac{1}{N-1} \sum_{j=1, j\neq i}^{N} \left(\bar{L}_{ij}(t) - \bar{G}_{ij}\right)^2} \Big/ \sqrt{2} \tag{6}$$

$$h(x) = \left(1 + \exp^{-(ax-b)}\right)^{-1} \tag{7}$$

Equation (3) states that the active view is the most important view. We calculate the inactive view importance factor β_j as a linear combination of $\bar{G}_i$ and $\bar{L}_i(t)$ in Equation (4), where the α_i parameter is carefully computed (by Equations (5)–(7)) to dynamically adjust the relative weights of the global and local models. Specifically, α_i is calculated as a normalized root mean squared error $E(\cdot)$ of $\bar{G}_i$ and $\bar{L}_i(t)$ in Equation (6). The denominator $\sqrt{2}$ is the maximum value, since both $\bar{G}_i$ and $\bar{L}_i(t)$ have values less than 1. The intuition behind $E(\cdot)$ is to prioritize only the most important inactive views. At the beginning of a streaming session, $E(\cdot)$ is expected to be high, thus, all inactive views will have low β values unless they are very important in the global model. During the streaming session, if the user activity approaches the global model, then $E(\cdot)$ decreases and the global model output would have a higher weight. If the user activity deviates from the global model, $E(\cdot)$ is high and β will follow the local model more. After computing $E(\cdot)$, we apply a shifted sigmoid function $h(\cdot)$ as in Figure 2b. We design $h(\cdot)$ such that its domain and range are in the period $[0, 1]$, and it grows faster than the identity function when $E(\cdot)$ increases. The design of $h(E(\cdot))$ ensures that the local model weighs more, unless an inactive view is very important in the global model. Thus, MASH fetches only important segments to save bandwidth while ensuring smooth playback.

IV. Proposed MASH Algorithm

A. Details of MASH

The proposed combined view switching model in the previous section results in an importance factor β_i for each view V_i, which is computed dynamically at each view change. Using the importance factors for all views, MASH determines the quality for the segments to be requested.

MASH is a buffer-based rate adaptation algorithm for multiview videos, which means it determines the requested segment quality based on the buffer occupancy level, and it does not need to estimate the network capacity. Previous buffer-based algorithms [5], [6] are designed for traditional single-view videos. Thus, they employ a simple buffer-rate mapping function. This function basically defines two parameters: the minimum B_{min} and maximum B_{max} buffer occupancies in seconds. If the buffer level is less than B_{min}, the requested quality for the segment is set to the minimum quality Q_{min}. And if the buffer level is greater than Q_{max}, the requested quality is set to the maximum quality Q_{max}. For buffer levels between $Q_{min}$and Q_{max}, the requested quality is linearly proportional to the slope of the function: $(Q_{max} - Q_{min}) \,/\, (B_{max} - B_{min})$.

Rate adaptation for multiview videos is far more complex, as it needs to handle many views of different importance, while not wasting network bandwidth or resulting in many stalls during playback for re-buffering. To handle this complexity, we propose employing a *family* of buffer-rate functions, which considers the relative importance of the active and inactive views and how this relative view importance dynamically changes during the streaming session. Specifically, we define a function $f_i(B_i(t))$ for each view V_i, which maps the buffer level $B_i(t)$ of that view to a target quality $Q_i(t)$ based on its importance factor β_i at time t. We use β_i to limit the maximum buffer occupancy level for view V_i as: $B_{max,i} = \beta_i \times B_{max}$. Since we set $\beta_i = 1$ for the active view, the algorithm can request segments up to the maximum quality $Q_{max,i}$. Notice that this equation implies that $\beta_i \leq \beta_j \iff B_{max,i} \leq B_{max,j} \iff Q_{max,i} \leq Q_{max,j}$. Thus, for inactive views, MASH can request segments for up to a fraction of their maximum qualities. Figure 3 illustrates the buffer-rate functions for two views V_i and V_j. V_i is the active view, so $B_{max,i} = B_{max}$. The figure shows when the requests stop for both V_i and V_j, and the maximum bitrate difference to reflect the importance of each view.

We show the pseudo code of MASH in Algorithm 1. We first note that there is an initialization phase of MASH (not shown in Algorithm 1), which occurs in the beginning of the streaming session. In this phase, MASH downloads the global transition matrix G from the server, and calculates the local

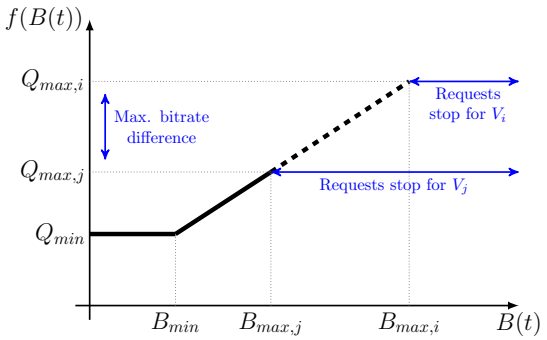


Fig. 3: Proposed buffer-rate functions of views V_i (active) and V_j (inactive).

model transition matrix $L(0)$ and its normalized form $\bar{L}(0)$ using Equations (1) and (2). Then, given an initial active view V_i, the view importance factors β_k (Equations (3) and (4)) and $B_{max,k}$ are calculated for all $k \neq i$. Finally, the buffer-rate functions f_k are calculated given B_{min}, $B_{max,k}$ and the available quality levels. After the initialization phase, MASH runs periodically every Δ seconds and it updates the local counting and transition matrices given $M(t-\Delta)$. It then checks if a view switch happens since the last invocation (Line 3), and updates its state accordingly. First, MASH updates normalized local vector $\bar{L}_i(t)$ for the current active view. Second, it updates parameter α_i for the current active view V_i (Line 5). Third, MASH iterates over all views and updates their importance factors β_k and maximum buffer occupancy $B_{max,k}$ values. Then, it updates the buffer-rate functions according to these new values (Line 13). Finally, at Line 17, MASH uses corresponding buffer-rate function f_k to compute segments rates.

B. Analysis and Overheads of MASH

Overheads: MASH requires two stochastic matrices: G and L, each of size N^2, where N is the number of views. For simplicity, let us assume that each element in all matrices takes 2 bytes, although more optimization is possible. Since G is maintained at the server, the client would need to download extra $2N^2$ bytes at the beginning of the streaming session. At the end of the session, the client uploads the final local model L to the server to update G, which also takes $2N^2$ bytes. The client stores G and L in the local memory, along with the counting matrix M used to simplify calculation of L. Thus, the added memory requirement is $6N^2$ bytes. On the server side, G is updated only once after the completion of each streaming session with total computational complexity of $O(N^2)$. On the client side, the initialization of L and M matrices takes $O(N^2)$ steps, but it is done once. During the session, only $O(N)$ steps are required to update M and L with every view switch. To put these bounds in perspective, consider a rich multiview video with 100 views. The client will download/upload up to 20 KB and need up to 60 KB of memory. These values are insignificant compared to the size of the multiview video (100s of MB). To create and maintain

Algorithm 1 MASH rate adaptation algorithm.

Input: V_i, V_j: current and previous active views.
Output: *requests*: segments to be requested from server.

1: $M_{ji}(t) = \gamma M_{ji}(t-\Delta) + (1-\gamma)(M_{ji}(t-\Delta)+1)$
2: $L_j(t) = M_{jl}(t) / \sum_k^N M_{jk}(t), \forall 1 \leq l \leq N$
3: **if** $(V_i \neq V_j)$ **then**
4: $\quad \bar{L}_{ii}(t) = 0$; $\bar{L}_{ik}(t) = L_{ik}(t)/(1-L_{ii}(t)), \forall k \neq i$
5: $\quad \alpha_i = h(E(\bar{L}_i(t), \bar{G}_i))$
6: $\quad$ **for** $(V_k \in V)$ **do**
7: $\quad\quad$ **if** $(V_i == V_k)$ **then**
8: $\quad\quad\quad \beta_k = 1$; $B_{max,k} = B_{max}$
9: $\quad\quad$ **else**
10: $\quad\quad\quad \beta_k = \alpha_i \times \bar{L}_{ik}(t) + (1-\alpha_i) \times \bar{G}_{ik}$
11: $\quad\quad\quad B_{max,k} = \beta_k \times B_{max}$
12: $\quad\quad$ **end if**
13: $\quad\quad f_k = f(B_{min}, B_{max,k}, Q_{min,k}, Q_{max,k})$
14: $\quad$ **end for**
15: **end if**
16: **for** $(V_k \in V)$ **do**
17: $\quad$ requests.add(getRequest(f_k, $B_k(t)$))
18: **end for**
19: **return** *requests*

the matrices, only $N^2 = 10$K simple operations are performed in few msec per each session (which lasts for minutes).

Analysis of the View Switching Models: We focus on two properties in the context of multiview video streaming: (1) convergence and (2) rate of convergence. These properties imply whether and when such models represent user activities.

Lemma 1 (Model Convergence): The global and local view switching models converge to the stationary distributions $\pi^{(g)}$ and $\pi^{(l)}$, respectively.

Proof: We constructed our models with the following properties, which reflect real multiview video applications. First, each model has a *finite state space* with state count equals N, which represents the finite views of the video. Second, the user can switch from any view to any view, that is, any state i can reach any other state j. Thus, our models are *irreducible*. Third, each state i can return to the same state i in irregular number of view switches. Hence, our models are *aperiodic*. It was shown in [17] (Theorem 1.8.3) that a finite state, irreducible, and aperiodic Markov chain converges. That is, for our global G (and similarly local L), there exists a unique probability distribution $\pi^{(g)}$ such that $\pi^{(g)} = \pi^{(g)}G$. That is, the global and local models converge to $\pi^{(g)}$ and $\pi^{(l)}$, respectively. ■

We note that by definition, $\pi^{(g)}$ is the left eigenvector $\vec{e}$ of G where its corresponding eigenvalue $\lambda_1 = 1$, and it equals to $\vec{e}/\sum_i^N e_i$. Such eigenvalue $\lambda_1 = 1$ exists since G is a stochastic matrix. Moreover, there are N eigenvalues such that $\lambda_1 = 1 > |\lambda_2| \geq \cdots \geq |\lambda_N|$. Given the eigenvalues λ's in the above lemma, the global model G can be shown to converge to $\pi^{(g)}$ in $O(|\lambda_2|^k)$ steps. Specifically, it was shown in [18] (Theorem 5.1, p. 378) that the convergence rate of such model is exponential in the order of the second dominating

eigenvalue λ_2. Since the global model is updated at the end of a streaming session, k is the streaming sessions count. We note that as λ_2 approaches 0, the model convergences very quickly (i.e., requires a small number of streaming sessions). λ_2 is small when most users agreed on watching specific views. As λ_2 increases, the global model needs more streaming sessions to reach its stationary distribution. This fact shows the advantage of the local model to capture user activity during single streaming session, where such user may deviate from people consensus, or there are no enough streaming sessions yet to represent such agreement.

V. Evaluation

We assess the performance of MASH through actual implementation and comparisons against the state-of-the-art algorithm in the industry. We also analyze the fairness of our algorithm across concurrent multiview video sessions and compare its performance against two variations of current rate-adaptation algorithms that could be used for multiview videos.

A. Experimental Setup

We have implemented a complete multiview video player in about 4,000 lines of Java code. It consists of HTTP client, decoders, renderer and rate adapter. Each view has its own decoder and frame buffer. The rate adapter decides on the segments to be requested and their quality levels. Then, segments are fetched from the HTTP server. Once a segment is fetched, it is decoded to frames and stored in the corresponding frame buffer. The renderer has references to all frame buffers, and it renders the corresponding active view.

Figure 4 shows our testbed, which consists of multiple virtual machines (VMs) running on the Amazon cloud. We chose high-end VMs with 1 Gbps links, so that the shared cloud environment does not interfere much with our network setup. The HTTP server in the figure is YouTube, when we actually run the YouTube multiview client. In other experiments, we install and use the `nginx` [1] as our HTTP server. Users run our multiview player with different rate adaptation algorithms. When we compare against YouTube, we use the player embedded in the Google Chrome web browser. The bandwidth and latency of the network links connecting VMs with the server are controlled using the Linux Traffic Control `tc` utility. We experiment with multiple network conditions to stress our algorithm.

We use a multiview video released by YouTube [1]. The video is for a concert and it has four different views shot by four cameras. The views cover the singer, band, stage, and fans. The user is allowed to switch among the four views at any time. The video is about 350 sec long, and it has four quality levels $Q = \{0.5, 1, 1.6, 2.8\}$ Mbps. As of the writing of this paper, YouTube did not release other multiview videos, and we can not import multiview videos to YouTube because of the proprietary nature of its multiview player.

We consider a general user activity model, which is captured by the Markov chain in Figure 2a. The Markov model has four states corresponding to the four views of the video. We control the transition probabilities to analyze the performance under different user activity patterns. Specifically, we create three activity patterns from this Markov chain by adjusting the transition probabilities: (1) FQ, frequent activity, where the user frequently switches views with an average view watching time of 30 seconds, (2) IFQ, infrequent activity, where the number of view switches is much less and the average view watching time is 60 seconds, and (3) GLB, where pattern follows the probabilities in Figure 2a. In FQ and IFQ patterns, we assign higher probabilities to switching to next view such that $p(V_{i+1}|V_i) > p(V_j|V_i), i \neq j$ to make sure that the user will watch the four views during the session. We note that our algorithm is totally unaware of these user patterns initially, but it does try to infer and use them as the session proceeds.

[1] https://www.nginx.com/

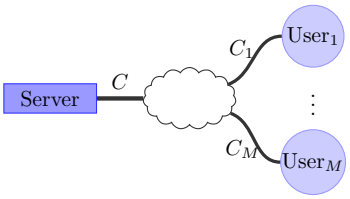


Fig. 4: Setup of our testbed. Link characteristics (e.g., bandwidth and latency) are controlled using the Linux Traffic Control (`tc`) utility.

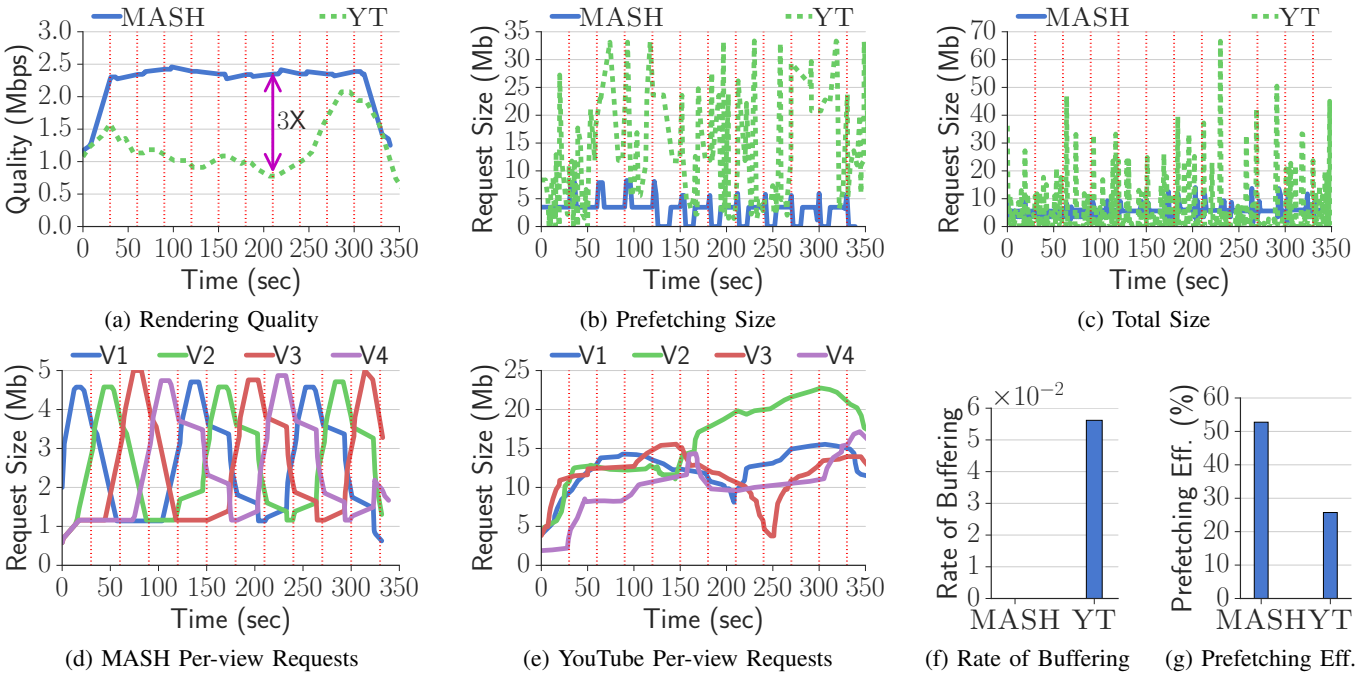
We consider multiple performance metrics; most of them were used in similar previous works, e.g., in [3], [4]. In particular, we measure the *average* of each of the following metrics across all streaming sessions in each experiment:

- *Rendering Quality:* bitrate of the watched views.
- *Rate of Buffering Events:* number of buffering events divided by the session duration.
- *Prefetching Efficiency:* ratio of the used (i.e., rendered) bytes to the total amount of fetched bytes from all views.
- *Fairness (Jain's) Index:* for M concurrent sessions, we calculate Fairness Index as $JI = (\sum_{i=1}^{M} Q_i)^2/(M \sum_{i=1}^{M} Q_i^2)$, where Q_i is the average quality in session i.
- *Server Load:* total bits sent by the server per second to serve all streaming sessions.

User studies [19], [3] show that the rendering quality and rate of buffering events should be optimized to gain user engagement. Thus, the first two metrics indicate user satisfaction. The third metric shows the bandwidth utilization. The Fairness Index indicates whether competing users get their fair share of bandwidth. The server load represents the bandwidth requirements on the server side to support multiview streaming.

B. MASH vs. YouTube Multiview Rate Adaptation

In this experiment, we compare individual multiview streaming sessions managed by the rate adaptation algorithm of YouTube (denoted by YT), which is proprietary and implemented in the Google Chrome web browser, against the



(a) Rendering Quality (b) Prefetching Size (c) Total Size

(d) MASH Per-view Requests (e) YouTube Per-view Requests (f) Rate of Buffering (g) Prefetching Eff.

Fig. 5: Comparison of MASH versus YouTube, for FQ (frequent view switching) scenario and bandwidth = 8 Mbps.

MASH algorithm implemented in our own multiview player. We stream the considered multiview video many times in different network conditions using the two rate adaptation algorithms: MASH and YT.

Specifically, we use the `tc` utility to create five scenarios for the average bandwidth on the link between the client and the Internet, assuming the bottleneck link is the last hop as YouTube is well provisioned: (1) average bandwidth of 8 Mbps, (2) average bandwidth of 16 Mbps, (3) bandwidth changes from 16 to 40 Mbps at $t = 60$ seconds, (4) bandwidth changes from 40 to 16 Mbps at $t = 60$ seconds, and (5) bandwidth changes from 16 to 40 Mbps at $t = 60$ seconds, then from 40 to 16 Mbps at $t = 90$ seconds. In the first two scenarios, the average bandwidth is kept constant, while in the last three the bandwidth changes. Thus, in our experiments, we are capturing the view switching by users as well as the dynamic changes of bandwidth. For each bandwidth scenario and for each user activity pattern, we start the streaming session using the Chrome browser and request the considered multiview video from YouTube. Once the session starts, we open the Developer Tools panel in Chrome to collect HTTP request and response records. During the streaming session, we switch among the views following the user activity patterns. After the session ends, we collect the corresponding HAR file. HAR files store information about the page, HTTP requests and responses, and timing information. We are interested in one object called `entries`. YouTube client interacts with the server using request query string. Specifically, YouTube client includes 36 query strings in each request for each view. We extract four query strings: (1) `id`: distinguishes different views, (2) `mime`: indicates whether this is a video or audio segment, (3) `itag`: represents the quality profile, and (4) `bytesrange`: the requested byte range. We focus on video segments only. We also do not pause the video at any time. We repeat the whole experiment using the MASH algorithm. Due to space limitations, we only present a sample of our results; other results are similar. In all following figures, vertical dotted red lines correspond to view switch time.

Figure 5 summarizes the results for the FQ user activity pattern, when the client average bandwidth is 8 Mbps. We note that 8 Mbps should be sufficient for YouTube since it naively downloads multiple views. If we decrease the bandwidth, YouTube will have lower rendering quality and higher rate of buffering. Figure 5a shows that MASH achieves much higher and more consistent rendering quality than YT. The figure indicates an improvement of up to 300% in the rendering quality can be achieved by our rate adaptation algorithm. The maximum improvement is shown in the figure around $t = 210$ seconds. MASH is conservative about fetching segments of different views at high bitrates. YT, however, is aggressive and requests highest bitrates most of the time, even if a view is inactive. This is shown in Figures 5b and 5c where YT prefetches a lot of bytes most of the time, especially at the beginning. MASH prefetches more bytes only when an activity pattern is captured. For example, Figures 5b and 5d show that at period $[240, 270]$, MASH prefetches V_2 at higher bitrate although the active view is V_1. That is why MASH's rendering quality does not suffer at view switches. On the other hand, Figure 5e shows that YT is unaware of such pattern. At the beginning, YT fetches all views at highest bitrates, then it focuses on the current and previous active views only. For example, if the user switches from V_1 to

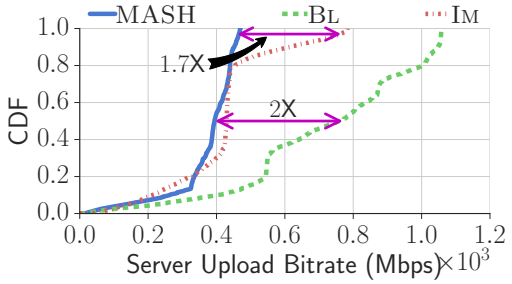


Fig. 6: Load on the server to handle 100 concurrent users.

V_2, YT keeps requesting V_1 at high bitrates for a while. This is shown in Figure 5e, where YT fetches bytes for inactive views V_1, V_3 and V_4 in period $[150, 180]$ almost as V_2. So, when it requests segments for active view V_2 in the same period, it takes time to reach a high bitrate. The decisions taken by MASH and YT do not only affect rendering quality, but the rate of buffering and prefetching efficiency as well. YT suffers from playback interruptions as shown in Figure 5f, with buffering rate of 0.056 (i.e., 20 re-buffering events in 350 sec), while MASH is perfectly smooth. Similarly, YT prefetches a lot of bytes without rendering them with prefetching efficiency of 24.3%, whereas for MASH it is 52.7% (Figure 5g). For other view switching and bandwidth scenarios, similar prefetching efficiency and rate of buffering results are obtained. YT can achieve the same rendering quality as MASH in high bandwidth scenarios only.

In summary, our experiments showed that MASH can produce much higher (up to 3X) and smoother quality than YT. They also show that unlike YT, MASH does not suffer from any playback interruptions even in presence of frequent user activities and dynamic bandwidth changes. Moreover, MASH is more efficient in using the network bandwidth, with a prefetching efficiency up to 2X higher than that of YT.

C. Fairness and Comparisons vs. others

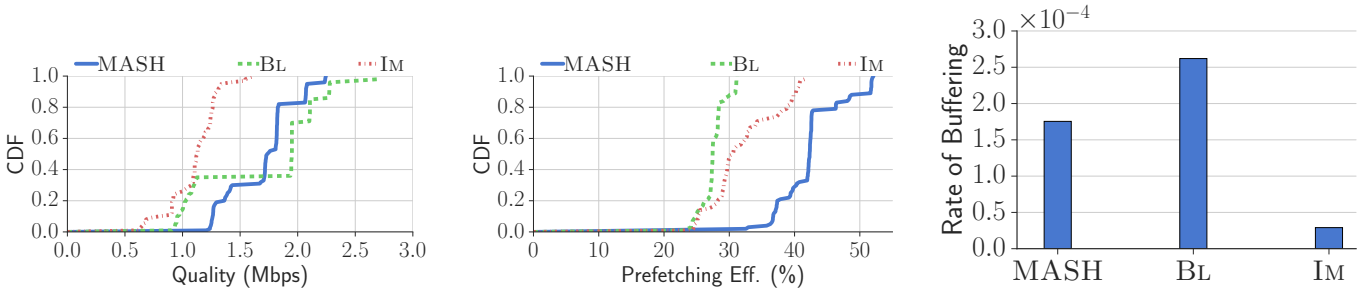
In this experiment, we assess the fairness and scalability of MASH, by analyzing the performance of concurrent multiview streaming sessions. We also compare our algorithm versus others. As shown in Figure 4, each of the M VMs concurrently runs our multiview player and requests the video from the server. The upload capacity of the server is C Mbps, and it is shared among all M users. Users have different download capacities. We set these capacities according to the Akamai state of the Internet report.[2] Specifically, the probability of $C_i = x$ Mbps follows this distribution $\{0.31, 0.37, 0.13, 0.12, 0.07\}$, where $x \in \{4, 10, 15, 25, 35\}$ Mbps. Similarly, we set the latency using the AT&T global IP network latency averages[3] with values $\{20, 35, 55, 100\}$ msec following a uniform distribution. For user activities, 60%, 25% and 15% of players follow the GLB, FQ and IFQ patterns, respectively. We use same quality levels Q as the previous experiment. We evaluated two configurations: (1) $C = 100$ Mbps, $M = 10$ users, and (2) $C = 1$ Gbps, $M = 100$ users. We only report the results of second, larger, configuration due to space limitations.

As elaborated in Section II, we are not aware of any rate adaptation algorithms for multiview video streaming over HTTP in the literature. To partially mitigate this issue, we compare MASH against two variations of current rate adaptation algorithms for single-view videos that can be used, after our modifications, with multiview videos. One might think of these variations as representative samples of possible extensions of current algorithms [5]. Specifically, we consider two buffer-based rate adaptation algorithms: (1) BL, baseline, that prefetches every view independently whether it is active or not, and (2) IM, inactive minimum, that prefetches the active view based on a buffer-rate function, and all inactive views with the minimum quality Q_{min}.

Figure 6 shows the cumulative distribution function (CDF) of the server upload bitrate to serve the 100 users by each algorithm. BL requests more data most of the time: more than 500 Mbps 80% of the time and up to 1050 Mbps. Whereas the maximum server upload bitrate is 780 and 470 Mbps for IM and MASH, respectively. The total amount of data during streaming sessions is 253 Gb, 146 Gb and 131 Gb for BL, IM and MASH respectively. Compared to BL and IM, MASH requests less data while improves rendering quality, prefetching efficiency, buffering rate and fairness. Figures 7a and 7b show that MASH outperforms IM in terms of rendering quality and prefetching efficiency for all streaming sessions. Specifically, MASH improves the average rendering quality and average prefetching ratio by up to 78.5% and 38.6%, respectively. This is because IM fills its inactive buffers with low bitrate segments, hence, the quality drops. Also, when a view switch occurs, most of these segments are not rendered, thus, the prefetching efficiency decreases. Since BL fetches all views all the time, it shows an improvement of the average rendering quality by up to 10%. This, however, comes at the cost of low prefetching efficiency. MASH improves the average prefetching efficiency compared to BL by up to 54.4%. Moreover, MASH improves BL average buffering rate by up to 50% as shown in Figure 7c. MASH incurs less buffering events in less number of streaming sessions. In particular, MASH and BL incur 1 and 1.125 buffering events for 6 and 8 streaming sessions, respectively. Since IM prefetches all views, it delivers smooth playback for 99 streaming sessions. However, it fills the buffers of inactive views with low quality segments, hence, it significantly reduces rendering quality compared to MASH (Figure 7a). Finally, MASH provides fair share of average quality across concurrent sessions with Fairness Index of 0.93. On the other hand, the index is 0.88 and 0.82 for BL and IM, respectively (figure not shown). This is because these algorithms request segments more than a user may need, hence, resulting in large quality variation across concurrent sessions. While MASH smartly considers user quality needs according to his/her activity pattern, resulting in fair share of

[2]https://www.stateoftheinternet.com/downloads/pdfs/2015-q4-state-of-the-internet-report.pdf

[3]https://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html



(a) Rendering Quality (b) Prefetching Efficiency (c) Rate of Buffering

Fig. 7: Comparison of MASH versus other rate adaptation algorithms.

high quality for all concurrent sessions.

In summary, our experiments showed that MASH achieves fairness across competing multiview streaming sessions, and it does not overload the streaming server. Moreover, MASH outperforms other rate adaptation algorithms, which are derived from current adaptation algorithms for single-view videos.

VI. Conclusions

Adaptively streaming multiview videos over HTTP is more challenging than streaming single-view videos, because the system needs to support user interactivities as well as handles the network dynamics. Despite the recent interest in the generation of multiview videos, very few works considered designing rate adaptation algorithms for such complex videos. This paper addressed this problem and presented a novel client-based multiview adaptive streaming over HTTP algorithm (called MASH). MASH achieves high rendering quality, smooth playback and efficient bandwidth utilization by modeling and supporting user interactivities. Specifically, MASH constructs and combines local and global view switching Markov chains to weigh the importance of views in the video. We showed that these models converge and impose low overheads on the client and server. We presented a new buffer-based approach to request segments based on the relative importance of different views and the current network conditions. We implemented the proposed algorithm and compared it against the rate adaptation algorithm used by YouTube to stream multiview videos. Our extensive empirical evaluation showed that MASH substantially outperforms the YouTube algorithm in terms of rendered quality, number of buffering events, and prefetching efficiency. In addition, our results showed that MASH: (i) is scalable as it does not overload the server, (ii) achieves fairness across concurrent streaming sessions, and (iii) renders smooth and high quality even in presence frequent view changes. This work can be extended in multiple directions. For example, we plan to apply MASH in virtual reality (VR) streaming and evaluate MASH with real users and more content.

Acknowledgment

This work is supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada, and by the Qatar National Research Fund (grant # [NPRP8-519-1-108]).

References

[1] "YouTube Experiment," http://tcrn.ch/1I1TcdK, accessed: January, 2017.
[2] T. Stockhammer, "Dynamic adaptive streaming over http: standards and design principles," in *Proc. of ACM MMSys*, San Jose, CA, Feb 2011, pp. 133–144.
[3] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement," in *Proc. of ACM SIGCOMM*, Toronto, Canada, Aug 2011, pp. 362–373.
[4] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran, "Probe and adapt: Rate adaptation for http video streaming at scale," *IEEE JSAC*, vol. 32, no. 4, pp. 719–733, Apr 2014.
[5] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. of ACM SIGCOMM*, Chicago, IL, Aug 2014, pp. 187–198.
[6] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proc. of ACM SIGCOMM*, London, UK, Aug 2015, pp. 325–338.
[7] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proc. of ACM MMSys*, San Jose, CA, Feb 2011, pp. 157–168.
[8] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," in *Proc. of ACM CoNEXT*, Nice, France, Dec 2012, pp. 97–108.
[9] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "Qdash: A qoe-aware dash system," in *Proc. of ACM MMSys*, Chapel Hill, NC, Feb 2012, pp. 11–22.
[10] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic http streaming," in *Proc. of ACM CoNEXT*, Nice, France, Dec 2012, pp. 109–120.
[11] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: Picking a video streaming rate is hard," in *Proc. of ACM IMC*, Boston, MA, Nov 2012, pp. 225–238.
[12] A. Hamza and M. Hefeeda, "A dash-based free-viewpoint video streaming system," in *Proc. of ACM NOSSDAV*, Singapore, Singapore, Mar 2014, pp. 55–60.
[13] T. Su, A. Javadtalab, A. Yassine, and S. Shirmohammadi, "A dash-based 3d multi-view video rate control system," in *Proc. of ICSPCS*, Gold Coast, Australia, Dec 2014, pp. 1–6.
[14] P. Xia and K. Nahrstedt, "Object-level bandwidth adaptation framework for 3d tele-immersive system," in *Proc. of IEEE ICME*, San Jose, CA, Jul 2013.
[15] Z. Yang, B. Yu, K. Nahrstedt, and R. Bajscy, "A multi-stream adaptation framework for bandwidth management in 3d tele-immersion," in *Proc. of ACM NOSSDAV*, Newport, RI, May 2006, pp. 14:1–14:6.
[16] W. Wu, A. Arefin, G. Kurillo, P. Agarwal, K. Nahrstedt, and R. Bajcsy, "Color-plus-depth level-of-detail in 3d tele-immersive video: A psychophysical approach," in *Proc. of ACM Multimedia*, Scottsdale, AZ, Nov 2011, pp. 13–22.
[17] J. R. Norris, *Markov chains*. Cambridge University Press, 1998.
[18] E. Cinlar, *Introduction to stochastic processes*. Prentice-Hall, 1975.
[19] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *Proc. of ACM IMC*, Boston, MA, Nov 2012, pp. 211–224.