RI11008 Revised (IND1111-018) November 29, 2011
Computer Science

IBM Research Report

On Managing Quality of Experience of Multiple Video Streams in Wireless Networks

Partha Dutta[1], Anand Seetharam[2], Vijay Arya[1], Malolan Chetlur[1], Shivkumar Kalyanaraman[2], Jim Kurose[2]

[1]IBM Research Division
Outer Ring Road
Embassy Manyatha B
Rachenahalli & Nagawara Villages
Bangalore KA 560045, India

[2]Department of Computer Science
University of Massachusetts
Amherst, MA USA

On Managing Quality of Experience of Multiple Video Streams in Wireless Networks

Partha Dutta[1], Anand Seetharam[2], Vijay Arya[1], Malolan Chetlur[1], Shivkumar Kalyanaraman[1], Jim Kurose[2]
[1]IBM Research, Bangalore, India
[2]Department of Computer Science, University of Massachusetts, Amherst, USA
{parthdut, vijay.arya, mchetlur, shivkumar-k}@in.ibm.com, {anand, kurose}@cs.umass.edu

***Abstract*—Managing the Quality-of-Experience (QoE) of video streaming for wireless clients is becoming increasingly important due to the rapid growth of video traffic on wireless networks. The inherent variability of the wireless channel as well as the variable bit rate (VBR) of the compressed video streams make managing the QoE a challenging problem. Prior work has studied this problem in the context of transmitting a single video stream. In this paper, we investigate multiplexing schemes to transmit multiple video streams from a base station to mobile clients that use number of playout stalls as a performance metric.**

In this context, we present an epoch-by-epoch framework to fairly allocate wireless transmission slots to streaming videos. In each epoch our scheme essentially reduces the vulnerability to stalling by allocating slots to videos in a way that maximizes the minimum 'playout lead' across all videos. Next, we show that the problem of allocating slots fairly is NP-complete even for a constant number of videos. We then present a fast lead-aware greedy algorithm for the problem. Our choice of greedy algorithm is motivated by the fact that this algorithm is optimal when the channel quality of a user remains unchanged within an epoch (but different users may experience different channel quality). Moreover our experimental results based on public MPEG-4 video traces and wireless channel traces collected from a WiMAX test-bed show that the greedy approach performs a fair distribution of stalls across the clients when compared to other algorithms, while still maintaining similar or lower average number of stalls per client.

I. Introduction

With the deployment of broadband wireless networks, the popularity of multimedia content on mobile devices is expected to increase significantly. A large portion of multimedia traffic is forecasted to be recorded videos such as movies, YouTube videos, and TV shows [1]. The inherent variability of both the wireless channel and the bit rate of compressed videos makes streaming videos on wireless networks a challenging task. This work investigates how multiple variable bit rate (VBR) videos can be multiplexed over a time-varying wireless channel while still maintaining a good QoE at the mobile clients.

A wireless streaming system consists of a video server connected to a base station over a high bandwidth wired backbone link and clients at mobile stations (MS) that communicate with the base station (BS) using a wireless channel (Fig. 1). The server stores pre-encoded videos, and upon receiving requests, streams out videos to the requesting clients. A video stream is composed of a sequence of frames that the client buffers and plays according to their playout times. If a frame is not received by its playout time, the client degrades the quality of the displayed video or it may *stall* the video to wait for more frames to arrive, or both. This work considers systems that stall in response to delayed frames.

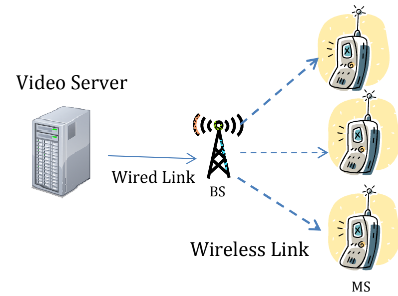


Fig. 1. A video streaming system

When streaming multiple videos over a wireless channel, in the case where the rate of each video as well as the rate available to each wireless client varies with time, the server can distribute stalls among video streams by appropriately multiplexing or scheduling their transmissions. This paper considers this multiplexing problem with the goal of minimizing stalls across all mobile clients.

The frame transmission scheduling/multiplexing algorithm we investigate in this paper makes three contributions. First, we present an epoch-by-epoch framework based on two ideas: (a) We divide the transmission time into *epochs* and use a Markov model to estimate the set of rates available to each wireless client during the next epoch. (b) We define the *playout lead* of a video as the duration of time the video can be played using the data already buffered by its client. Since the playout lead plays an important role in determining whether a video stalls in an epoch, we present a fair multiplexing scheme that takes into account the channel rates and maximizes the minimum lead among all videos in an epoch. Second, we show that the optimization problem of maximizing the minimum lead is NP-complete even for two videos. We present a fast lead-aware greedy algorithm that is sub-optimal for wireless channels, but we show that the algorithm is optimal for the special case where the channel quality of a user does not vary within an epoch, but different users may have different channel quality. Finally, we conduct trace-driven simulations with publicly available MPEG-4 video traces, and wireless channel quality traces that we collected from a WiMAX test-bed. Our simulations demonstrate that the greedy algorithm ensures a fair distribution of stalls across clients while maintaining a low average number of stalls per client. In particular, when the wireless network is average-provisioned or slightly over-provisioned as compared to the total bit-rate of the considered videos (cases that are interesting in practice), the greedy algorithm reduces the number of stalls by a factor of 3 to 4 when compared to other algorithms in our simulations. Our results also show that the greedy scheme is robust against changes in client's *stall-recovery buffering scheme* (which determines how long a client stalls the playout when a frame is not received in time) and changes in epoch duration.

In the remainder of this paper, the video streaming system is described in Section II. Section III introduces multiplexing based on playout leads and develops the corresponding problem formulation. Hardness results are given in Section IV followed by the greedy algorithm in Section V. The evaluation framework and results for the experiments are given in Section VI and Section VII, respectively. Comparison with related work is presented in section VIII. We conclude in Section IX with directions for future work.

II. Streaming system and Channel Model

We consider a video streaming system similar to [2] and shown in Fig. 1. We assume that the server simultaneously and separately streams n videos $v_1, \ldots, v_n$ to n clients $1, \ldots, n$ via the base station. A video object is composed of a sequence of frames that are displayed at a constant rate by the client. However, since the size of each frame varies significantly, the required transmission rate also varies with time. For a video v_i, its *playback curve* $p_i(t)$ specifies the cumulative data needed by time t relative to the start of its playout, in order to play the video without interruptions. The playback curve is a characteristic of a video and is independent of the underlying channel. We assume that clients have sufficient buffer space and they buffer frames that have been received but not yet displayed. If the next frame to be displayed is not received by its playout time, the client stalls playout for a certain duration during which it continues to buffer data received from the server. It resumes playout based on its *stall-recovery buffering scheme*. Common buffering schemes include: (i) waiting for a fixed amount of time, (ii) waiting for a fixed amount of future playout data, and (iii) waiting for a fixed number of future playout frames. For a client i, its *receiver curve* $G_i(t)$ specifies the cumulative amount of data it has received by time t. The cumulative amount of data played out by time t is given by its *playout curve* $O_i(t)$. Figure 2(a) shows an example playback, receiver, and playout curve for a client. The notation used in this paper is summarized in Table I.

We assume a broadband wireless system (such as WiMAX) wherein the transmission time is divided into *intervals* (Fig. 2(b)). The duration of an interval is small enough so that the channel state remains unchanged within it. Intervals are divided into a fixed number of (transmission) *slots* that are allocated to clients. The base station can transmit to at most one client in a slot. Depending on the channel conditions, each client receives a certain bit rate in the allocated slots. The bit rate for a client remains the same in all slots within an interval but can change between intervals. Following [2], we assume that the wireless channel is error-free due to an ideal error control mechanism such as ARQ.

Notation	Definition
n	number of clients
R, A	channel rate vector, transition matrix (resp.)
$N_{ep}^{in}, N_{in}^{sl}, N_{ep}^{sl}$	#intervals/epoch,#slots/interval,#slots/epoch (resp.)
I_i	initial probability distribution of channel state
F	frames played out per second
Y_i, V_i	#bits,#complete frames (resp.) transmitted in epoch
L_i	lead at the end of the epoch
Φ_i	inverse playback curve
r_{ij}	#bits that can be transmitted to client i in slot j

TABLE I
Important notations (note: subscript i refers to client i and # denotes 'number of')

III. Epoch-by-Epoch Multiplexing based on Playout Leads

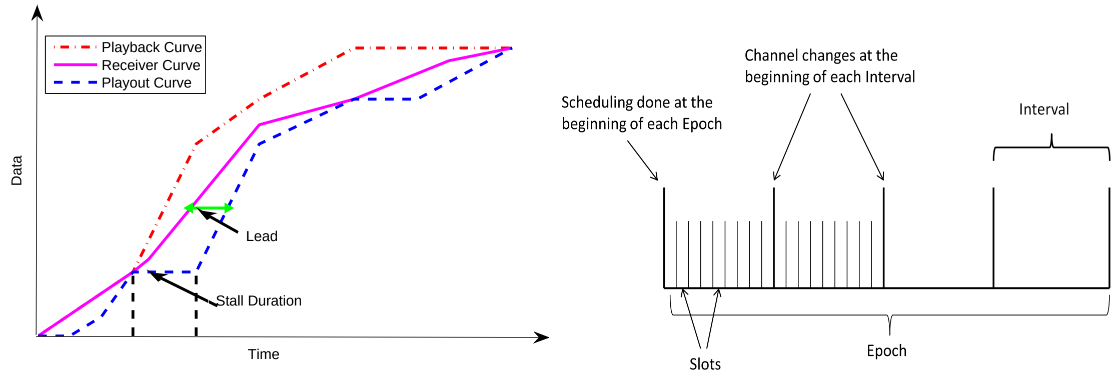
We define an *epoch* to contain a fixed number of intervals (Fig. 2(b)). The variation of rates across intervals, as seen at a client, is modeled using a generic discrete-time Markov model given by (R, A) where the possible channel states are identified by the transmission rates $R = (r_1, r_2, \ldots, r_K)$ and A is the transition matrix. (R is also called the rate vector.) Here r_i denotes the number of bits that can be transmitted in a time slot when the channel is in state i [2]. Each client's channel is modeled as an independent Markov chain, and each client estimates the transition matrix corresponding to its channel as discussed below. At the beginning of the epoch, clients send their transition matrix as well as the initial state of the channel to the server so that the server can compute the expected rates of all slots available to all clients during the epoch.

At the beginning of each epoch, our multiplexing scheme allocates slots to clients within that epoch. To motivate the allocation strategy, note that a client's current buffer size (in bits) indicates its vulnerability to stalling: the smaller the buffer, the more likely is the occurrence of a stall. However, for VBR videos, buffer size is a poor indicator of this vulnerability since it does not consider the amount of data needed to play the next few frames. On the other hand, the *playout lead* of the video, i.e., the duration of additional time a client can play the video using only its buffered data, takes into account the VBR nature of the video. Therefore in our scheme, within each epoch the server attempts to prevent stalls by maximizing the playout leads. To ensure that the stalls are evenly distributed across all videos, slots are allocated such that the minimum lead among all videos is maximized. In our system model, we assume that clients communicate their playout leads to the server at the beginning of each epoch.

A. Modeling the Multiplexing Problem

As previously noted, to avoid stalls, at the beginning of each epoch, slots are allocated to clients such that the minimum lead among all videos is maximized at the end of that epoch. We now present our modeling of this multiplexing problem.

Preliminaries: Let N_{ep}^{in} and N_{in}^{sl} denote the number of intervals in an epoch, and the number of slots in an interval, respectively. Thus the total number slots in an epoch $N_{ep}^{sl} = N_{ep}^{in}.N_{in}^{sl}$. Each video is played at the constant rate of F frames per second.



(a) (b)

Fig. 2. (a) Playback, receiver and playout curves of a video stream (b) Epochs, Intervals, Slots

Consider the i^{th} client in a particular epoch. Let I_i be the state vector denoting the probability distribution of channel states at the i^{th} client at the beginning of the epoch. Then, given the Markov channel model, the state probability distribution of the channel state at the client at the beginning of the k^{th} interval in the epoch is I_iA^k.

Let X_{ik} be the random variable denoting the number of bits that can be transmitted to client i in any slot of the k^{th} interval. Then, its expectation $E[X_{ik}]$ is the dot product of I_iA^k and the channel transmission rate vector R. Suppose that the server assign s_{ik} slots to client i in the k^{th} interval. Then the random variable Y_i for the number of bits transmitted to client i in this epoch can be expressed as $\sum_{k=1}^{N_{ep}^{in}} s_{ik}X_{ik}$. From linearity of expectation, $E[Y_i] = \sum_{k=1}^{N_{ep}^{in}} s_{ik}E[X_{ik}] = \sum_{k=1}^{N_{ep}^{in}} s_{ik}E[I_iA^k.R]$.

Playout Lead: The playout lead of a video at a given time is the additional duration of time that the video can be played out using only data currently in the client buffer. Therefore, the playout lead is equal to the number of complete frames in the client buffer divided by the frame rate F. At the beginning of the epoch, let o_i and g_i denote the amount of time for which the video has been played out at the client i, and the amount of time for which the data required for the playout has been received at the client, respectively. (The values of o_i and g_i can be computed from the calculation in the previous epoch, and the video playout and receiver curves.) Thus, the playout lead of the video i at the beginning of this epoch is $g_i - o_i$, and this value is known at the beginning of the epoch. Let L_i be the random variable denoting the playout lead of the video at the end of this epoch, and V_i be the random variable denoting the number of additional frames that can be *completely* received by the end of this epoch. Then, $L_i = g_i - o_i + (V_i/F)$.

Inverse Playback Curve: For an epoch, we now define a deterministic function that maps the number of bits received to the number of *complete* frames received. The *inverse (frame) playback curve* Φ_i for each video i is defined as follows: if b bits are transmitted to video i in this epoch, then the number of complete frames that are received increases by $\Phi_i(b)$ at the end of the epoch. Thus, $V_i = \Phi_i(Y_i)$. (Note that partially transmitting a frame does not increase the lead of the video.) The inverse playback curve can be easily computed from the video frame sizes.

Estimating $E[V_i]$ from $E[Y_i]$: As g_i and o_i are known constants, $E[L_i] = g_i - o_i + E[V_i]/F$. Unfortunately, since the video frame sizes can vary widely, the mapping Φ_i from Y_i to V_i is non-linear, and hence, we cannot easily obtain $E[V_i]$ from $E[Y_i]$. Therefore, we estimate $E[V_i]$ by $\Phi_i(E[Y_i])$. Thus, $E[L_i] \approx g_i - o_i + \Phi_i(E[Y_i]) = g_i - o_i + (1/F)\Phi_i(\sum_{k=1}^{N_{ep}^{in}} s_{ik}E[I_iA^k \ . \ R])$.

The Multiplexing Problem: Our goal, at the beginning of an epoch, is to assign slots with the goal of maximizing the minimum expected lead at the end of the epoch. This problem can be expressed as follows:

Objective: $Max \ Min\{E[L_1], \ldots, E[L_n]\}$
subject to the constraints:
1. $\sum_{i=1}^{n} s_{ik} = N_{in}^{sl}, \forall k \leq N_{ep}^{in}$
2. $s_{ik} \geq 0 \ , \forall i \leq n, \forall k \leq N_{ep}^{in}$

IV. Hardness Result

We now investigate the optimization problem described in the previous section. We first reformulate the problem as a combinatorial problem. (We assume that slots in an epoch are numbered sequentially from 1 to N_{ep}^{sl}.)

Inputs and Constraints. At the beginning of an epoch, the video of the i^{th} client has an initial lead of $l_i = g_i - o_i$ seconds; i.e., it has received the data corresponding to the $F * l_i$ frames after the last played frame.

Let r_{ij} be the number of bits of video that can be transmitted to client i in slot j. Thus, $r_{ij} = E[I_iA^k.R]$, when slot j belongs to interval k. For ease of presentation, we also call r_{ij} the rate of video i in slot j. Given the values of the rates, a slot allocation for an epoch specifies the client to which each slot is allocated.

The Problem. In the Lead-based Multiple Video Transmission (LMVT) problem, given the above input, we need to find a slot allocation that maximizes the minimum lead among all videos

1: **function** initialization
2: $AvailableSlots \leftarrow \{1, \ldots, N_{ep}^{sl}\}$; $j \leftarrow 1$
3: $\forall$ client i: $lead_i \leftarrow$ initial lead of i; $I_i \leftarrow$ initial state distribution; $rcvbits_i \leftarrow 0$
4: $\forall$ client i: compute the inverse playback curve Φ_i for this epoch
5: **for** $1 \leq k \leq N_{ep}^{in}$ **do** {*for all intervals in epoch*}
6: **while** $j < kN_{in}^{sl}$ **do** {*for all slots in interval*}
7: $r_{ij} \leftarrow E[I_i A^k.R]$; $j \leftarrow j+1$
8: **function** greedy algorithm
9: select a client with the lowest id i s.t. ($\forall j \leq n, lead_i \leq lead_j$)
10: select a slot j s.t. ($j \in AvailableSlots$) and ($\forall x \in AvailableSlots, r_{ij} \geq r_{ix}$)
11: allocate slot j to client i; $rcvbits_i \leftarrow rcvbits_i + r_{ij}$
12: $lead_i \leftarrow$ initial lead of video i + $\frac{\Phi_i(rcvbits_i)}{F}$
13: remove j from $AvailableSlots$

Fig. 3. A greedy algorithm (executed at the beginning of each epoch)

at the end of the epoch. (Here, ‘lead’ refers to the expected playout lead described in the previous section.) We now show that the following decision version of LMVT is NP-complete: given a constant L, does there exist a slot allocation such that every user has a lead of at least L seconds at the end of the epoch? We show the NP-completeness by reduction from the subset-sum problem [3]. Due to lack of space the proof of NP-completeness is given in the appendix.

Lemma 1: The decision version of the LMVT problem is NP-complete.

For a constant number of videos, we have designed a pseudo-polynomial time algorithm to optimally solve LMVT using dynamic programming. However, this algorithm requires long running time when the number of videos is high. Due to lack of space the algorithm is presented in the appendix.

Lemma 2: For a constant number of videos, there is a pseudo-polynomial time algorithm to optimally solve LMVT.

V. A Lead-Aware Greedy Algorithm

We now present a fast lead-aware greedy algorithm for the LMVT problem. The algorithm is optimal for LMVT for the case when the channel conditions remain constant within an epoch, but different users may have different channel quality (as shown in Lemma 3 below). Later in our simulations, we numerically evaluate the algorithm for the general case when the channel conditions of users may vary.

Lead-Aware Greedy Algorithm. Starting with the initial playout leads of the videos and all the slots in the epoch, the greedy algorithm allocates slots one by one (Figure 3) as follows. In each iteration, the algorithm selects a video i with the minimum lead, such that video i has the lowest id among the videos with the minimum lead. Then the algorithm allocates client i a slot j in which client i has the highest rate r among all available slots. Before moving to the next iteration, slot j is marked unavailable for all videos, and the lead of client i is increased corresponding to the transmission of r bits to video i using the inverse playback curve Φ_i (line 12 of Figure 3). The algorithm iterates until there are no available slots in the epoch. (We would like to remind the reader that the lead in this algorithm refers to the *expected* value of the lead random variable.)

To motivate our choice of the above greedy algorithm, we now show that the algorithm is optimal for LMVT when each client's channel condition does not change within an epoch (but different clients may have different rates).

Lemma 3: If the rate of each client does not change within an epoch, the greedy algorithm yields an optimal solution for LMVT.

Proof: As the rate of a client i does not change within an epoch, each slot that is allocated to the client i provides a constant number of bits, say r_i. In this setting, the greedy algorithm simply chooses the client i that has the lowest id among the clients with the minimum lead, and selects the next available slot and allocates it to i. The proof of optimality is by induction on the number of allocated slots.

For the induction, we first introduce some notation and observations. At any point in the execution of the LMVT algorithm, the lead of a client can only change on receiving sufficient slots for the client's next video frame, and therefore, the client's lead can change only by a multiple of $1/F$. For any LMVT solution (slot allocation to clients) X, let l_i^X denote the lead of client i in solution X, and let $l_{min}^X = min_i\{l_i^X\}$ be the minimum lead in X. Let $sl(X, j)$ denote the number of slots allocated to client j in solution X. Note that for a solution Y and client k, if $l_j^X > l_k^Y$ then $sl(X, j) > sl(Y, k)$, on the other hand, if $sl(X, j) \geq sl(Y, k)$ then $l_j^X \geq l_k^Y$.

Base Case: If only 1 slot is available, the greedy algorithm allocates it to a client with the minimum lead and therefore the minimum lead is maximized.

Induction Step: Let us assume that the greedy algorithm yields an optimal solution G for every $d \leq c$ slots. Let $G(c+1)$ be the solution given by the greedy algorithm for $c+1$ slots. We must prove that $G(c+1)$ is optimal. To show by contradiction, let us assume that there exists an alternate solution $S(c+1) \neq G(c+1)$ that is optimal for $c+1$ slots, and $S(c+1)$ has a higher minimum lead than $G(c+1)$. Thus, $l_{min}^{S(c+1)} > l_{min}^{G(c+1)}$ (i.e., $l_{min}^{S(c+1)} \geq l_{min}^{G(c+1)} + 1/F$) [Observation **A0**]. Let client i have the lowest id among the clients with the minimum lead in $G(c)$. After the $(c+1)$th slot is allocated to i by the greedy algorithm, we have one of the following two cases:

Case 1: Minimum lead changes, i.e., $l_{min}^{G(c+1)} > l_{min}^{G(c)}$.

Let j be a client with the minimum lead in $G(c+1)$, i.e., $l_{min}^{G(c+1)} = l_j^{G(c+1)}$ (j need not be different from i). Then $l_j^{S(c+1)} \geq l_{min}^{S(c+1)} > l_{min}^{G(c+1)} = l_j^{G(c+1)}$ [Observation **A**]. Thus, j is allocated at least one more slot in $S(c+1)$ than in $G(c+1)$. Let us remove a slot from j in $S(c+1)$ to obtain a solution $S(c)$ for c slots. Since we have only removed one slot from j in $S(c+1)$ to obtain $S(c)$, $l_j^{S(c)} \geq l_j^{S(c+1)} - 1/F \geq l_j^{G(c+1)}$ [Observation **B**], and $l_{min}^{S(c)} = min\{l_j^{S(c)}, l_{min}^{S(c+1)}\} \geq l_j^{G(c+1)}$ (where the last inequality follows from inequalities **A** and **B**). Thus, we have $l_{min}^{S(c)} \geq l_j^{G(c+1)} = l_{min}^{G(c+1)} > l_{min}^{G(c)}$ which is a contradiction since $G(c)$ is optimal for c slots.

Case 2: Minimum lead remains unchanged at some value z, i.e., $l_{min}^{G(c+1)} = l_{min}^{G(c)} = z$.

Observe that this can happen either when (a) i has not received data constituting an entire frame and therefore its lead has not advanced (b) i received data constituting one or more frames and its lead advanced but there is another client j such that $l_j^{G(c)} = l_i^{G(c)} = z$.

We first consider the case when $z = 0$. As $l_{min}^{S(c+1)} \geq z + 1/F > 0$ (from **A0**), in $S(c+1)$ every client is allocated enough slots for at least its first frame, Thus, for each client j, the minimum number of slots needed for the first frame, say sl'_j, is less or equal to than $sl(S(c+1), j)$, and therefore, $\sum_j sl'_j \leq c+1$. Now consider the execution of the greedy algorithm until the minimum lead (over all videos) becomes greater than 0. The algorithm selects a client j, in the increasing order of their client id, and allocates client j enough slots for its first frame, i.e., sl'_j, and then moves to the next frame. Therefore, given $c+1 \geq \sum_j sl'_j$ slots, the greedy algorithm will allocate sufficient slots to each client for its first frame, and hence, the allocation will have a minimum lead of at least $1/F$. Thus, $l_{min}^{G(c+1)} \geq 1/F$, a contradiction.

We now consider the case when $z > 0$. Let us look back in time to the point in the greedy algorithm's execution when the minimum lead in G has last changed. Let us assume that this occurred δ slots back, i.e., $l_{min}^{G(c-\delta)} = z - 1/F$ and $l_{min}^{G(c-\delta+1)} = \ldots = l_{min}^{G(c+1)} = z$ [Observation **C**]. Thus, in the solution $G(c+1-\delta)$, there must have been a set of clients P each with lead z.

Consider the period of execution of the greedy algorithm while going from $G(c+1-\delta)$ to $G(c+1)$. In this period, the algorithm must have assigned slots only to clients in P. Also, no client in P would have received slots more than what is required for its next one frame (because on receiving slots required for one frame, the client's lead increases, and it does not remain a client with the minimum lead) [Observation **C1**]. Let $P1$ be the set of clients in P that have received sufficient slots for their next frame in this period, and $P2$ be the remaining set of clients in P (that have not received enough slots for their next frame in this period). We note that $P2$ cannot be an empty set, otherwise, the lead of $G(c+1)$ would be higher than $G(c+1-\delta)$.

Let q be any client in $P2$. Then $l_q^{G(c+1)} = z$. Since, from our initial assumptions, $l_{min}^{S(c+1)} > l_{min}^{G(c+1)} = z$, $l_q^{S(c+1)} \geq l_{min}^{S(c+1)} > z = l_q^{G(c+1)}$ [Observation **D**]. Also, for any client j in $P1$, $l_j^{G(c+1)} = z + 1/F$ (since it has received slots for the next frame) [Observation **D1**]. As, $l_j^{S(c+1)} \geq l_{min}^{S(c+1)} > l_{min}^{G(c+1)} = z$, we have, $l_j^{S(c+1)} \geq z + 1/F = l_j^{G(c+1)}$ [Observation **E**].

To show a contradiction, let us modify the solution $S(c+1)$ by removing $\delta + 1$ slots to obtain a solution $S(c-\delta)$ for $c - \delta$ slots as follows. For every client j in P, we remove any $sl(G(c+1), j) - sl(G(c+1-\delta), j)$ slots from its slot allocation, and in addition, we remove one more slot from one (arbitrarily chosen) client, say w, in $P2$. (The removed slots add up to $\delta + 1$ because δ slots were allocated by the greedy algorithm to obtain $G(c+1)$ from $G(c+1-\delta)$.) We now show that the minimum lead in $S(c-\delta)$ is higher than the minimum lead in $G(c-\delta)$, thus resulting in a contradiction (because $G(c-\delta)$ is optimal for $c-\delta$ slots). Let q be the client with the minimum lead $S(c-\delta)$. We consider four possible cases.

(1) q is not in P. In this case, no slots were removed from q to obtain $S(c-\delta)$ from $S(c+1)$, and so q had the minimum lead in $S(c+1)$ as well. Therefore, $l_q^{S(c-\delta)} = l_{min}^{S(c+1)} > l_{min}^{G(c+1)} = z > l_{min}^{G(c-\delta)}$ (from **A0** and **C**).

(2) q belongs to $P1$. Note that, since a process in $P1$ receives the minimum number of slots that is required for its lead to be $z + 1/F$ in $G(c+1)$ (from **C1** and **D1**), and $l_q^{S(c+1)} \geq l_{min}^{S(c+1)} \geq l_{min}^{G(c+1)} + 1/F = z + 1/F$ (from **A0**), q receives equal or more slots in $S(c+1)$ than in $G(c+1)$. Then, $sl(S(c-\delta), q) = sl(S(c+1), q) - (sl(G(c+1), q) - sl(G(c+1-\delta), q)) \geq sl(G(c+1), q) - (sl(G(c+1), q) - sl(G(c+1-\delta), q)) = sl(G(c+1-\delta), q)$. Therefore, $l_q^{S(c-\delta)} \geq l_q^{G(c+1-\delta)} = z > l_{min}^{G(c-\delta)} = z - 1/F$ (where the last inequality follows from **C**).

(3) q belongs to $P2$ but is distinct from w. Since $q \in P2$, $l_q^{S(c+1)} > l_q^{G(c+1)}$ (from **D**), and therefore $sl(S(c+1), q) > sl(G(c+1), q)$. Now, $sl(S(c-\delta), q) = sl(S(c+1), q) - (sl(G(c+1), q) - sl(G(c+1-\delta), q)) > sl(G(c+1), q)) - (sl(G(c+1), q) - sl(G(c+1-\delta), q)) = sl(G(c+1-\delta), q)$. Therefore, $l_q^{S(c-\delta)} \geq l_q^{G(c+1-\delta)} = z > l_{min}^{G(c-\delta)} = z - 1/F$ (where the last inequality follows from **C**).

(4) $q = w$. Since $q \in P2$, $l_q^{S(c+1)} > l_q^{G(c+1)}$ (from **D**), and therefore $sl(S(c+1), q) > sl(G(c+1), q)$. Now, $sl(S(c-\delta), q) = sl(S(c+1), q) - (sl(G(c+1), q) - sl(G(c+1-\delta), q)) - 1 > sl(G(c+1), q)) - (sl(G(c+1), q) - sl(G(c+1-\delta), q)) - 1 \geq sl(G(c+1-\delta), q)$. Therefore, $l_q^{S(c-\delta)} \geq l_q^{G(c+1-\delta)} = z > l_{min}^{G(c-\delta)} = z - 1/F$ (where the last inequality follows from **C**). ■

As a special case of the above lemma, when the transmission channel is of Constant Bit Rate (CBR), i.e., the rate of slots do not change within an epoch or across the users, e.g., in a wired link, the greedy algorithm is optimal.

Corollary 1: For a CBR channel, the greedy algorithm yields an optimal solution for LMVT.

VI. Experimental Setup

A. Trace-Driven Experiments

To demonstrate the efficacy of the greedy algorithm, we perform trace-based experiments and report the results in this section. Our evaluation uses two types of traces:
(i) VBR Video Traces describing the variation in the frame sizes of videos for emulating video playouts.
(ii) User-Level Wireless Channel Traces describing the rates received by various users over time to emulate real wireless channel conditions.

We use the publicly available MPEG-4 *VBR Video Traces* [4], [5] in our experiments. The videos play out at a constant frame rate of 30 frames per second. We perform experiments with video traces encoded in Common Intermediate Format (CIF) and Quarter CIF (QCIF). All evaluation is performed considering that a group of 8 different videos is being streamed simultaneously to 8 different users over a wireless channel. Unless mentioned otherwise, all results are reported for CIF

Name of Video	Mean bit rate (Mbps)	Mean frame size (Kb)	Standard deviation of frame size(Kb)
Star Wars IV	0.42	14	17.6
Lord of the Rings I	0.65	21.6	22.7
Tokyo Olypmics	1.06	35.4	39.4
Matrix I	0.41	13.4	17.1
Matrix II	0.61	20.2	25.5
Matrix III	0.52	17.1	20.5
NBC News	1.33	44	34
Silence of the Lambs	0.44	14.7	22.2

TABLE II
CIF VIDEO TRACE STATISTICS

Parameter	Value
PHY	OFDMA
Carrier Frequency	2.59 Ghz
Channel Bandwidth	10 MHz
Frame duration	5 ms
Transmission power	30 dbm
Antenna model	Sector
Fragmentation/Packing	ON
ARQ	OFF

TABLE III
WIMAX SYSTEM PARAMETERS FOR TRACE COLLECTION

videos. A brief description of the 8 CIF video traces used, is given in Table II. Detailed information about the CIF and QCIF traces is available in [5].

User-Level Wireless Channel Traces describe the rates achieved by different users in every interval of each epoch. To generate these traces we collected signal strength measurements over a (802.16e) WiMAX network deployed in WINLAB at Rutgers University. During our trace collection, the base station was made to continuously transmit data packets and signal strength (RSSI) was recorded; we performed the measurement at the receiver (a laptop) under vehicular and pedestrian mobility. A brief description of the parameters of the WiMAX network used in our trace collection is given in Table III. (Due to lack of space, we present further details in the appendix.)

B. Scheduling Algorithm: Parameters

To evaluate our epoch-by-epoch multiplexing strategy based on playout lead we need to specify the epoch duration, interval size and the number of slots in an interval. To count the number of stalls at the client, we assume the following buffering scheme: if the client is not allocated enough data in the current epoch to playout for the whole duration of the epoch, the client stalls for the whole epoch. (We evaluate other common buffering schemes in Section VII-C.) For ensuring a smooth viewing experience, it is undesirable to have small or large epochs as the former will result in frequent glitches while the latter will significantly delay playout. Hence in our experiments we consider epochs to be in the seconds' timescale. We perform our experiments considering an epoch duration of 10 seconds. However, we also evaluate our algorithm for varying epoch durations.

Recall that in our multiplexing strategy, epochs are divided into intervals, which are subdivided in slots (Figure 2(b)). Our algorithm takes scheduling decisions based on the assumption that the channel state changes significantly only from one interval to the other. We use the mapping between the modulation and coding schemes (MCS) and the SINR values for a WiMAX network provided in [6] to generate the Markov Chain for modeling wireless channel state transitions from one interval to the next. The transition matrix is then determined by empirically computing the probabilities of transitioning between these states from the traces collected. We choose an interval duration to be 1 second in our experiments because we want to capture channel variation due to path loss and shadowing effects. The fast fading behavior of the channel will average out for video frames (as their playout duration is typically large). Due to lack of space, we present further details in the appendix.

The main objective of our experiments is to demonstrate that the proposed greedy algorithm is able to achieve its goal of minimizing the number of stalls irrespective of the epoch duration, interval size or number of slots per interval. Determining the optimal epoch duration, the interval size or the number of slots in an interval so as to maximize viewer satisfaction is beyond the scope of this work.

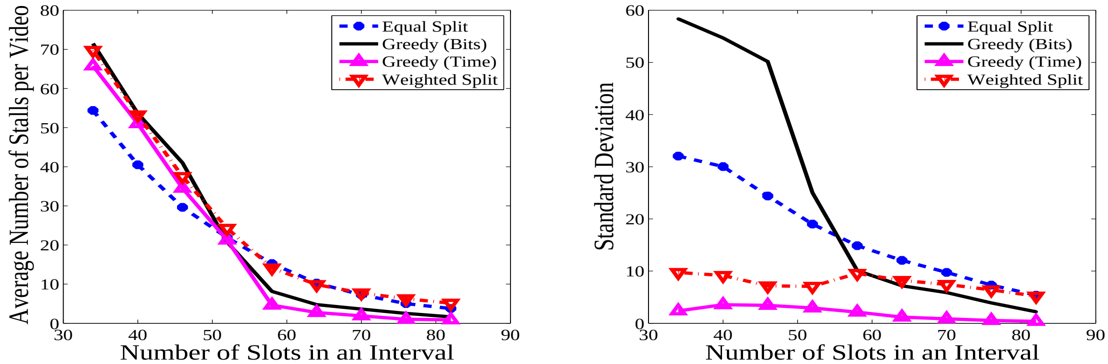
VII. RESULTS

In this section we present and discuss the results for the various experiments conducted. We compare the performance of the greedy algorithm against two baseline approaches: the equal-split and the weighted-split algorithms. In the equal-split approach, we divide the number of slots available in every interval equally among all the users. In the weighted-split the total number of slots in any interval is divided in proportion to the mean bit rate of the individual video streams. While allocating the slots, these two algorithms neither consider the playout lead nor the wireless channel variability, and hence, we expect them to be significantly unfair compared to our greedy strategy.

To emphasize the importance of making scheduling decisions based on playout lead, we also consider a variant of our greedy algorithm from Section V (we denote our algorithm from Section V by greedy-time). We consider a greedy-bit algorithm which is similar to our greedy-time algorithm except for one crucial aspect; it allocates the next slot to the video with the minimum lead in terms of playout bits instead of playout time. To avoid cluttering the plots with many lines, we show only a few results for the greedy-bit algorithm. The greedy-bit approach ignores the variability in the frame sizes (i.e., burstiness) of a video with the result that it allocates fewer resources to a video experiencing a burst, thereby unfairly making it stall for longer durations.

A. Distribution of Stalls

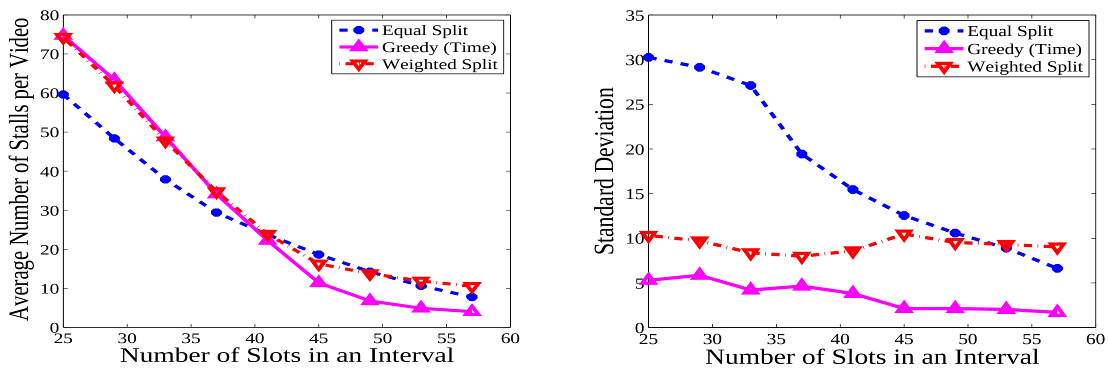
In this subsection we study the distribution of stalls as a function of the number of slots in an interval (keeping the interval duration constant). The epoch duration is taken to be 10 seconds. Using the steady state probabilities of the Markov model, one can compute the expected number of bits received per slot. By varying the number of slots in an interval we are essentially varying the total resource (in terms of bandwidth) that is available at the base station.



(a) Average number of stalls

(b) Fairness

Fig. 4. Vehicular: Distribution of stalls with variation of wireless channel resource (slots) for CIF videos



(a) Average number of stalls

(b) Fairness

Fig. 5. Pedestrian: Distribution of stalls with variation of wireless channel resource (slots) for CIF videos

1) Vehicular Mobility: Figure 4 presents the variation of the average number of stalls for four multiplexing algorithms: equal-split, weighted-split, greedy-bit and greedy-time. Table IV provides the expected bit rate in the steady state for different values of the number of slots per interval. In our experiment, the mean bit rate of the aggregate of 8 CIF videos is approximately 5.4Mbps. Thus, from Table IV, we note that 34, 58 and 82 slots per interval correspond to the wireless channel being severely under-provisioned, average-provisioned and over-provisioned, respectively for the vehicular mobility scenario. In terms of the average number of stalls per video, both the greedy algorithms perform better than the equal-split and the weighted-split approaches, except when the network is severely under-provisioned.

The under-provisioned case is not of practical interest as the average number of stalls experienced is very high for all algorithms. We, however, offer an explanation as to why the equal-split performs the best in terms of average number of stalls in this scenario. The main reason is that 2 videos in the set of 8 videos considered, have mean bit rates much higher than the others (Table II). In the equal-split approach, all video streams are given the same number of slots and consequently a significantly larger number of stalls is experienced by the high bit rate videos in comparison to the low bit rate ones. Therefore, although the average number of stalls is lower in equal-split when compared to greedy-time, equal-split is unfair, a fact also evident from its large standard deviation for the under-provisioned scenario. To validate this observation, we performed experiments excluding the two high bit rate videos and found that the performance of the equal-split algorithm becomes similar to greedy-time algorithm in the under-provisioned case, with respect to the average number of stalls.

Number of Slots	Expected Bit-Rate (Mbps)
34	3.23
58	5.7
82	8.0

TABLE IV
EXPECTED STEADY STATE BIT RATE FOR A GIVEN NUMBER OF SLOTS

In the average and over-provisioned scenario, we observe that the greedy algorithms outperform the other two approaches. With respect to fairness, the standard deviation of the number of stalls shows that in terms of evenly distributing the stalls among the videos, our greedy-time algorithm performs significantly better than other algorithms. As discussed earlier, we observe that the greedy-bit algorithm is unfair in distributing the stalls (Figure 4), and so we will not consider the greedy-bit algorithm any further.

To highlight the performance of the greedy-time algorithm, we present results for the average number of stalls experienced for the mildly over-provisioned case (64 and 70 slots) in Table V. The mildly over-provisioned case is the scenario of interest in practice and we observe that the greedy-time

Scheme	Number of Stalls (Slots 64)	Number of Stalls (Slots 70)
Equal Split	10.25	7.25
Weighted Split	9.875	7.75
Greedy	2.75	1.875

TABLE V
NUMBER OF STALLS PER VIDEO FOR AVERAGE-PROVISIONED NETWORK

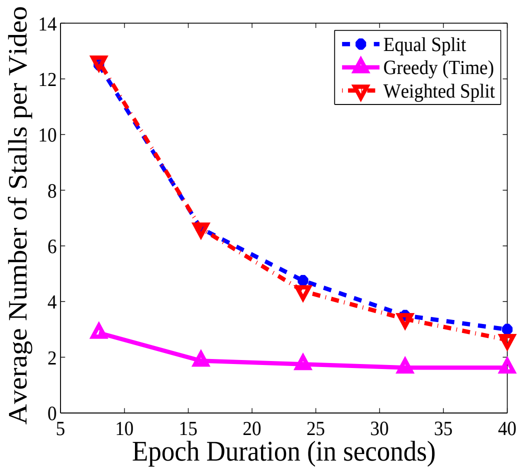


Fig. 6. Sensitivity to epoch duration

algorithm reduces the number of stalls by a factor of 3 to 4 when compared to equal-split and weighted-split. Overall, we observe that the greedy-time multiplexing algorithm gives the best performance both in terms of reducing the average number of stalls per video and evenly distributing the stalls among the videos.

2) Pedestrian Mobility: Figure 5 shows the result for the experiments conducted under pedestrian mobility. We observe that the greedy-time algorithm again outperforms the equal and weighted split algorithms in terms of average number of stalls and fairness in the pedestrian mobility case as well. Due to lack of space, in the remaining sections we only present the results for the vehicular mobility case.

B. Sensitivity to Epoch Duration

In the experiments presented thus far, we have fixed the epoch duration to be 10 seconds. In Figure 6, we present the variation in the average number of stalls per video as a function of the epoch duration. The number of slots in an interval is 64. We observe that the average number of stalls for the greedy-time algorithm decreases slightly as the epoch duration increases. As we increase the epoch duration, the reason for faster decrease in the number ofstalls of the other schemes as compared to the greedy scheme is that, the greedy scheme starts with a significantly lower number of stalls, and therefore, the benefits of increase the epoch duration is not as pronounced as the other schemes. Although it is not captured in Figure 6, the total stall duration averaged over all videos increases with the increase in epoch duration due to our buffering scheme that whenever a video stalls, it stalls for an whole epoch. (We study other buffering schemes in the Section VII-C.)

C. Sensitivity to Buffering schemes

Recall that in the results presented above, we have assumed a client stall-recovery buffering scheme in which the client stalls for the entire epoch when there is not enough data present to playout for the whole epoch. However, the media players at the clients may have a different buffering scheme. Following [2], we now consider the three common buffering schemes:

- Fixed Stall Buffering Delay (FBD): Once a stall occurs, resume display only after a fixed duration of time.
- Fixed Buffered Playout Data (FPD): Once a stall occurs, resume display only after a fixed amount of data is received.
- Fixed Buffered Playout Time (FPT): Once a stall occurs, resume display only after the receiver has accumulated enough data corresponding to a fixed playout duration.

We performed experiments to determine whether our algorithm's performance is sensitive to different client buffering schemes. Figures 7(a), 7(b), and 7(c) show the variation of the average number of stalls for the FBD, FPD and FPT buffering schemes respectively. In these simulations we again considered 64 slots in each interval. In terms of playout stalls, the greedy-time algorithm still outperforms the other schemes irrespective of the buffering scheme adopted by the player at the client. We also observed that the greedy-time algorithm performs significantly better in terms of evenly distributing the stalls across the videos, although we omit the plot due to lack of space.

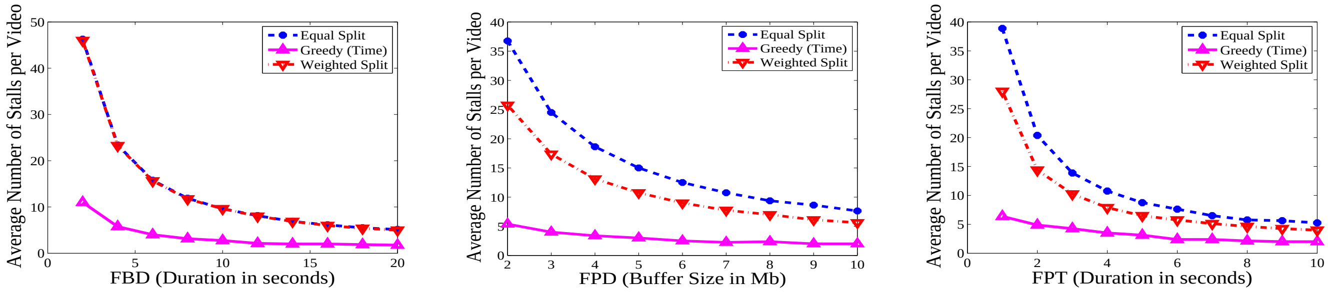
D. Sensitivity to Different Video Traces

We also conducted experiments with two sets of 8 QCIF video traces, available from [4], [5]. We observed that, for the QCIF video traces the trend obtained is similar to CIF videos, with the greedy-time algorithm outperforming the other approaches. Although the gains are not as prominent as in the case of the CIF videos in terms of average number of stalls, the greedy-time algorithm still significantly outperforms in terms of fairness. Due to lack of space, we present the results in the appendix.

VIII. RELATED WORK

Although compression techniques reduce the mean bit rate of video streams, it introduces considerable rate variability over several time scales [7], [8]. Resource allocation for VBR video streaming has been studied extensively for wired networks. Smoothing the video transmission is one of the primary techniques used for reducing the effect of bit-rate variability. By pre-fetching some of the initial video frames before their display times, smoothing techniques can minimize the effect of variability in bit-rates under various resource constraints, such as peak bit rate, client buffer size, and initial playout delay [9], [10], [11], [12].

Rate allocation for multiple video streams is a well studied problem [13], [14], [15], [16], [17]. [13] investigates minimizing rate variability when transmitting multiple video streams given the client buffer size in a high-speed wired network. In the RCBR service introduced in [14], the rate of each video is renegotiated at the end of each interval to provide statistical QoS guarantees. [15] presents a call-admission scheme at a statistical multiplexer and bound the aggregate loss probability.



(a) Fixed Stall Buffering Delay Strategy(FBD)

(b) Fixed Buffered Playout Data (FPD)

(c) Fixed Buffered Playout Time (FPT)

Fig. 7. Different buffering schemes

A linear programming model is proposed in [16] to compute a globally optimized smoothing scheme to stream multiple videos. A recent work [17] derives bounds on the dropped frames, delay, and buffer requirement that can be obtained by statistically multiplexing VBR streams at the video server by using a two-tiered bandwidth allocation. Although our algorithm perform periodic rate allocation among multiple video streams, our work differs from the above papers in two crucial aspects: our primary objective of fairly managing playout stalls across the videos, and our focus on time-varying wireless channel.

Our work is closest to the work presented in [18], [2] for managing stalls. Given the initial playout delay and the receiver buffer size, [18] determines upper and lower bounds on the probability of stall-free display of a video. [2] develops an analytical framework to find the distribution of the number of stalls while streaming a VBR video over a wireless channel. However, unlike our work, both papers consider a single video stream. The problem of transmitting multiple VBR videos from a base station to mobile clients has been studied in [19], but the work focusses on maximizing bandwidth utilization while reducing energy consumption, and do not to address the issue of stalling of video playout.

IX. Conclusion

In this paper, we have presented a multiplexing scheme to manage stalls for multiple video streams that are transmitted over a time-varying bandwidth-constrained wireless channel. We considered a fairness criterion of maximizing the minimum playout lead for managing stalls. We have assumed that all server-to-client channels have the same transition matrix for the Markov channel model, which might not hold in practice. Some clients may have a poor channel condition for a protracted period of time, and in maximizing the minimum playout lead scheme, the performance of the entire system may be degraded due to these clients. We plan to address this issue in detail as part of our future work.

References

[1] Cisco, “Visual Networking Index: Global mobile data traffic forecast update, 2009-2014.”

[2] G. Liang and B. Liang, “Balancing interruption frequency and buffering penalties in VBR video streaming,” in *INFOCOM*, 2007.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[4] G. Auwera, P. David, and M. Reisslein, “Traffic and quality characterization of single-layer video streams encoded with H.264/AVC advanced video coding standard and scalable video coding extension,” *IEEE Transactions on Broadcasting*, vol. 54, no. 3, 2008.

[5] P. Seeling, M. Reisslein, and B. Kulapala, “Network performance evaluation with frame size and quality traces of single-layer and two-layer video: A tutorial,” *IEEE Communications Surveys and Tutorials*, vol. 6, no. 3, 2004, cIF Video traces at http://trace.eas.asu.edu/mpeg4/index.html and QCIF Video traces at http://trace.eas.asu.edu/cgi-bin/main.cgi.

[6] J. G. Andrews, A. Ghosh, and R. Muhamed, *Fundamentals of WiMAX: Understanding Broadband of Wireless Networking*. Prentice Hall Communications Engineering and Emerging Technologies Series, 2007.

[7] M. W. Garrett and W. Willinger, “Analysis, modeling and generation of self-similar VBR video traffic,” in *ACM SIGCOMM*, 1994.

[8] A. R. Reibman and A. W. Berger, “Traffic descriptors for VBR video teleconferencing over ATM networks,” *IEEE/ACM Trans. Netw.*, vol. 3, no. 3, 1995.

[9] S. S. Lam, S. Chow, and D. K. Y. Yau, “An algorithm for lossless smoothing of MPEG video,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, 1994.

[10] T. Ott, T. V. Lakshman, and A. Tabatabai, “A scheme for smoothing delay-sensitive traffic offered to ATM networks,” in *IEEE INFOCOM ’92: Proceedings of the eleventh annual joint conference of the IEEE computer and communications societies on One world through communications (Vol. 2)*, 1992.

[11] N. B. Shroff and M. Schwartz, “Video modeling within networks using deterministic smoothing at the source,” in *INFOCOM*, 1994.

[12] S. Sen, J. Dey, J. Kurose, J. Stankovic, and D. Towsley, “Streaming CBR transmission of VBR stored video,” in *SPIE Symposium on Voice Video and Data Communications*, 1997.

[13] J. D. Salehi, S.-L. Zhang, J. Kurose, and D. Towsley, “Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing,” *IEEE/ACM Trans. Netw.*, vol. 6, no. 4, 1998.

[14] M. Grossglauser, S. Keshav, and D. N. C. Tse, “RCBR: a simple and efficient service for multiple time-scale traffic,” *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, 1997.

[15] Z.-L. Zhang, J. F. Kurose, J. D. Salehi, and D. F. Towsley, “Smoothing, statistical multiplexing, and call admission control for stored video,” *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 6, 1997.

[16] H. Stern and O. Hadar, “Optimal video stream multiplexing through linear programming,” in *IEEE International Symposium on Information Technology*, 2002.

[17] J. Londono and A. Bestavros, “A two-tiered on-line server- side bandwidth reservation framework for the real-time delivery of multiple video streams,” *BUCS-TR-2008-012, Boston University*, 2008.

[18] G. Liang and B. Liang, “Effect of delay and buffering on jitter-free streaming over random VBR channels,” *IEEE Transactions on Multimedia*, vol. 10, no. 6, 2008.

[19] C.-H. Hsu and M. Hefeeda, “On statistical multiplexing of variable-bit-rate video streams in mobile systems,” in *ACM Multimedia*, 2009.

[20] P. Sadeghi, R. A. Kennedy, P. Rapajic, and R. Shams, “Finite-state markov modeling for fading channels,” *IEEE Signal Processing Magazine*, 2008.

[21] H. Rahul, F. Edalat, D. Katabi, and C. Sodini, “Frequency-aware rate adaptation and mac protocols,” in *MobiCom*, 2009.

[22] G. Verticale and L. Musumeci, “Impact of multippath fading on the fairness of the wimax system,” in *ICLAN*, 2009.

Appendix

A. Hardness

1) Proof of Lemma 1: *Proof:* Given a slot allocation, and the playout curve for each video, it is easy to verify in polynomial time whether each video will have a lead of L seconds at the end of the epoch. Thus the decision version of LMVT is in NP.

In an instance of the subset-sum problem, given a set S of P positive integers $\{x_1, \ldots, x_P\}$, and a positive integer B, we need to find a subset of S such that the sum of the elements in the subset is exactly B [3]. Let Y be the sum of all integers in S. (We assume $B < Y$; otherwise, the subset-sum instance is trivial to solve.) Let Π denote the set $\{1, \ldots, P\}$. For this instance of the subset-sum problem, we construct an instance of LMVT as follows.

There are two videos v_1 and v_2, each with zero initial lead, and each plays at 1 frame per second. Let the inverse playback curve of v_1, $\Phi_1(b)$, be a function which is 0 for $b < B$, and 1 for $b \geq B$. (An example of such a video is one that contains a single frame of size B bits.) Similarly, let $\Phi_2(b)$ be a function which is 0 for $b < Y - B$, and 1 for $b \geq Y - B$. The required minimum lead L for each video is 1.

There is only one slot in each interval, and there are P intervals. For each slot $j \in \Pi$, the rates of both the videos are equal to x_j; i.e., $x_j = r_{1j} = r_{2j}$. We now show that the above instance of subset-sum problem has a solution if and only if the constructed LMVT problem instance has a solution.

Suppose the subset-sum problem instance has a solution given by a subset S' of S. We construct a solution for the instance of LMVT as follows: for each $1 \leq j \leq P$, if $x_j \in S'$ then we allocate the slot j to video v_1, else we allocate the slot to video v_2. In either case, x_j bits are transmitted in slot j for the allocated video. Since, the sum of all elements in S' is B, this allocation results in transmission of B bits and $Y - B$ bits for v_1 and v_2, respectively. Thus, both videos have a lead 1.

For the reverse direction, assume that we have a solution of LMVT in which both the video have a lead of 1. Thus, v_1 and v_2 are transmitted at least B bits and $Y - B$ bits, respectively. In the solution, suppose that $\Pi_1 \subseteq \Pi$ be the set of slots that are allocated to v_1, and the remaining slots are allocated to v_2.

Note that, for each $j \in \Pi_1$, the number of bits transmitted to v_1 is $r_{1j} = x_j$. Since at least B bits are transmitted to v_1, $B \leq \sum_{j \in \Pi_1} r_{1j} = \sum_{j \in \Pi_1} x_j$. Similarly, for video v_2, $Y - B \leq \sum_{j \in \Pi \setminus \Pi_1} r_{2j} = \sum_{j \in \Pi \setminus \Pi_1} x_j$. However by construction, $\sum_{j \in \Pi} x_j = Y$, so $\sum_{j \in \Pi_1} x_j = B$ and $\sum_{j \in \Pi \setminus \Pi_1} x_j = Y - B$. Thus, the subset $\{x_j \ : \ j \in \Pi_1\}$ of S is a solution of the subset-sum instance. ■

B. Algorithm for proof of Lemma 2

1) An optimal pseudo-polynomial time algorithm for LMVT: We now give a dynamic programming based algorithm to solve LMVT. If the number of videos is a constant, the algorithm runs in time pseudo-polynomial in terms of the input. Note that, it follows from the proof of Lemma 1 that the LMVT problem is NP-complete even for two videos. (In this algorithm, we assume that the server is allowed to transmit any $b_{ij} \leq r_{ij}$ bits to the video i in slot j. Note that this assumption does not change the optimal value of the objective because for any solution in which the server transmits $b_{ij} < r_{ij}$ bits in slot j to video i, we can modify the solution to transmit r_{ij} bits in slot j. This modification does not decrease the objective value and does not change the slot allocation.)

We present our algorithm in two steps. First, we present an algorithm considering the number of bits transmitted to each video, and later we show how to modify the algorithm to incorporate lead. (Recall that N_{ep}^{sl} denotes the total number of slots in an epoch, and let the slots in an epoch be numbered from 1 to N_{ep}^{sl}.)

Finding feasible Tx-vectors. For a given n-tuple $< b_1, \ldots, b_n >$ (which we call a transmission-vector or *Tx-vector*) and a given number of total slots, say m, we determine whether there is slot allocation such that, for each $1 \leq i \leq n$, video v_i receives b_i bits in the allocation. If there is such a slot allocation, we say that the Tx-vector is m-feasible. Let $F(m, T)$ denotes the predicate whether Tx-vector T is m-feasible. For a Tx-vector T, we denote by $T[i]$ the i^{th} element of T. For any pair of Tx-vector $T1$ and $T2$, we define the relation $T1 \preceq T2$ to be true if and only if $T1[i] \leq T2[i]$ for each $1 \leq i \leq n$. The maximum number of bits that can be transmitted to a video v_i in the epoch is $Q(\sum_{j=1}^{N_{ep}^{sl}} r_{ij})$, and we denote it by b_i^{max}. This maximum is achieved when all slots are allocated to v_i. We compute the N_{ep}^{sl}-feasibility of all Tx-vectors whose i^{th} element is at most b_i^{max}, for each $1 \leq i \leq n$. We first state the following straightforward lemma.

Lemma 4: For any pair of Tx-vectors $T1$ and $T2$ such that $T1 \preceq T2$, and for any $m \geq 0$, if $T2$ is m-feasible then $T1$ is also m-feasible.

A slot allocation for $T1$ can be easily obtained by appropriately reducing the number of bits in the slot allocation for $T2$. We omit the proof of the lemma here. Our dynamic programming algorithm for finding feasibility of Tx-vectors immediately follows from the following lemma. (Recall that, r_{im} is the rate of video i in slot m.)

Lemma 5: Let T denote any Tx-vector. For each $1 \leq i \leq n$, let W_i denotes a Tx-vector that is identical to T except that $W_i[i]$ is maximum$(T[i] - r_{im}, 0)$. Let $F(0, T)$ be true if $T =< 0, \ldots, 0 >$, and false otherwise. Then, for $m \geq 1$, $F(m, T)$ is true if and only if at least one of the n predicates $F(m-1, W_i)$, $1 \leq i \leq n$ is true.

Proof: Suppose $F(m-1, W_i)$ is true for some $1 \leq i \leq n$. Then, there is a slot allocation using $m - 1$ slots such that for every $1 \leq j \leq n$, video v_j is transmitted $W_i[j]$ bits. Consider a slot allocation for m slots that is identical to that for W_i until slot $m - 1$, and the m^{th} slot is allocated to v_i with r_{im} (or $T[i]$, if $T[i] < r_{im}$) transmission bits. Then, in this new slot allocation, every video is transmitted the number of bits specified in Tx-vector T. Thus, $F(m, T)$ is true.

For the reverse direction, suppose $F(m, T)$ is true for some T and $m \geq 1$, and hence, there is an allocation for T using m slots. Consider the video that is allocated in slot m, say v_i, and say b bits are allocated in slot m. Then, there is slot allocation using $m - 1$ slots for the Tx-vector T' which is identical to T

except that $T'[i] = T[i] - b$. Thus, $F(m-1, T')$ is true. Since, $b \leq r_{im}$, $W_i \preceq T'$. Then, from Lemma 4, $F(m-1, W_i)$ is also true. ■

In the dynamic programming algorithm, using Lemma 5, starting from the $m = 1$ to $m = N_{ep}^{sl}$, we compute the $F(m,T)$ for all possible values of $T \preceq < b_0^{max}, \ldots, b_n^{max} >$. For each of the N_{ep}^{sl} slots, there are at most $\prod_{i=1}^{n} b_i^{max}$ Tx-vectors. To update each vector, we need to lookup n entries in the dynamic programming table. Thus, the time-complexity of the algorithm is $O(nN_{ep}^{sl} \prod_{i=1}^{n} b_i^{max})$ and the space-complexity is $O(N_{ep}^{sl} \prod_{i=1}^{n} b_i^{max})$.

Finding feasible Tx-vectors with maximum value of its min-lead. Recall that, in Section III-A, we approximated the (expected) lead of a video $E[L_i]$ by $g_i - o_i + \Phi_i(E[Y_i])$, where Y_i is the number of bits transmitted to client i in the epoch, and Φ_i is the inverse playback curve of the video for client i. Before starting our algorithm, we compute the inverse playback curve for each video. Then, at each step of the dynamic programming algorithm, after calculating the feasibility for each Tx-vector, if the vector is feasible then we also compute its min-lead (over all videos) by computing the lead of each video i using $g_i - o_i + \Phi_i(E[Y_i])$. Also, we maintain a pointer to the feasible vector with the maximum value of its min-lead, among all feasible vectors seen so far (and the pointer is updated at each step of the algorithm). Thus, at the end of algorithm, we will have a pointer to the feasible vector with the maximum value of its min-lead. This modification increases the time-complexity of each step by a factor of n, and in addition, requires the time to pre-compute the inverse playback curves.

C. Details of Experimental Setup

1) Markov Chain Model: In this subsection we describe the Markov Chain model used in our experiments. We use the mapping between the modulation and coding schemes (MCS) and the SINR values for a WiMAX network provided in [6] to generate the Markov Chain. The SINR regime is divided into a number of ranges and there exists an MCS for each range that maximizes throughput. The MCS in turn indicate the different rates that are achievable in practice. Hence we can consider each of these SINR ranges to correspond to a state in the rate based Markov Chain model. Finite State Markov chains based on SNR to model the wireless channel have been well studied in literature. [20] provides a detailed description of the various models available in literature. Further use of SNR to bitrate mappings is also common [21], [22]. We would like to note here that the MCS only inform us how the rates achieved are related. For example, assuming a loss less channel using 16QAM, 3/4 the rate achieved will be twice that obtained by using QPSK, 3/4. The actual number of bits received however depends on the bandwidth (in Hz) of the channel.

Our Markov channel model for the wireless channel thus has 6 different states [6]. Considering the different MCS, the rates achieved in the 6 states of the Markov Chain have the following ratio $[1, 1.5, 2, 3, 4, 4.5]$. Our base station reports RSSI values whose unit of measurement is different from the SINR value reported in [6]. The minimum and maximum values of RSSI measured in our experiments is -85 dBm and -37 dBm. We map the maximum and minimum values of our RSSI measurements to the corresponding ones in [6] and do a linear extrapolation to determine the mapping between RSSI ranges and the rates achieved.

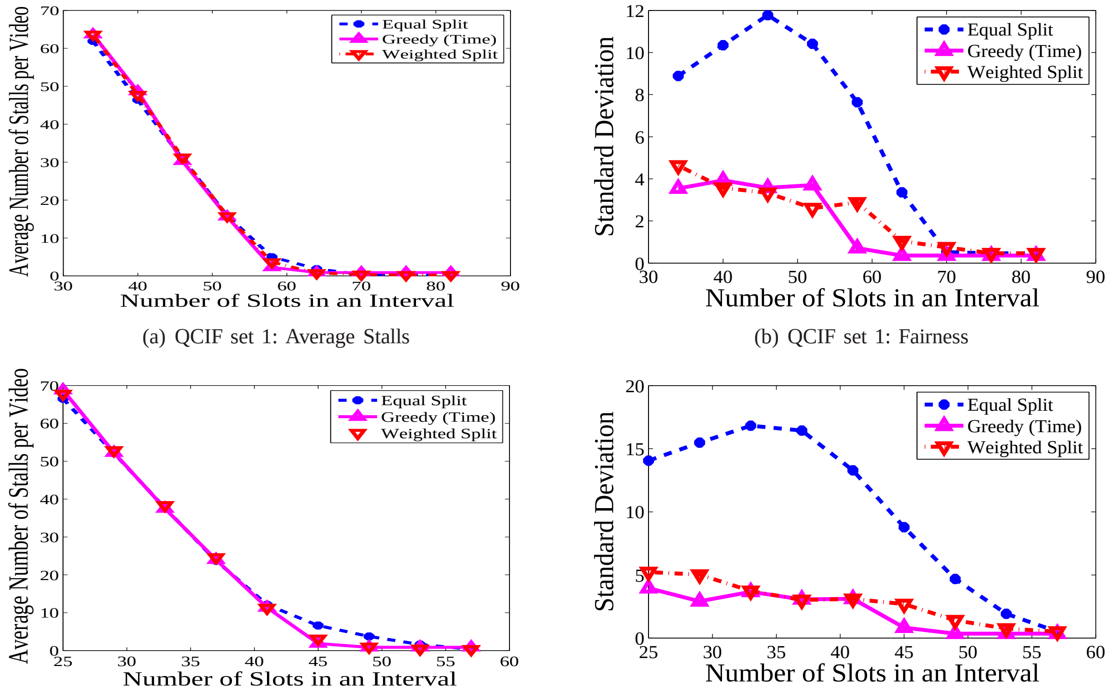
We know that the states of the Markov model correspond to the number of bits that can be successfully transmitted in a slot. The vector of transmission rates is taken to be $R = [1, 1.5, 2, 3, 4, 4.5] * 50000$ bits. We conduct experiments by changing the number of slots in an interval. Varying the total number of slots in an interval, while keeping R unchanged, is equivalent to varying the bit rate at which the base station can transmit the video streams. A larger number of slots can be viewed as an increase in the available wireless channel resources, such as the channel bandwidth (in Hz).

2) Generating User-Level Traces: Signal strength measurements (RSSI values) are collected by making the base station continuously transmit data packets and receiving them on a laptop. To capture different kinds of behavior of mobile users we consider both pedestrian and vehicular mobility scenarios. The vehicular measurements were gathered by placing the laptop in a car and driving around the WINLAB campus. Signal strength measurements under pedestrian mobility are gathered by walking around the same campus with the laptop. The base station only has a transmission range of 500 meters. Therefore, we collect received power measurements for only 3 vehicular and pedestrian mobility experiments, each of duration 10 minutes. The entire transmission range of the base station was exhaustively covered by these experiments and so we did not perform additional experiments.

Each interval is assumed to be of 1 second duration and hence actual channel conditions (i.e. RSSI) at the beginning of each second need to be known for generating the *User-Level Traces*. We use the RSSI-rate mapping to determine the rate achieved in any interval. Using this approach we generate *rate traces* from the RSSI measurements consisting of the rates achieved in each interval for the 3 vehicular and pedestrian mobility cases.

In our experiments the duration of each video being streamed by the 8 users is approximately 27 minutes or 48000 frames (the frame rate is $1/30^{th}$ of a second). We synthetically generate 8 different *User-Level Traces* (each 27 minutes) emulating the real channel conditions (separately for vehicular and pedestrian mobility) by randomly choosing sections from these *rate traces* and stitching them together.

3) Determining the Transition Matrix of the Markov Chain: To obtain the transition matrix of the Markov Chain we first combine all the *rate traces* into a single one. From this combined*rate trace* one can determine the sequence of states through which the Markov Chain has progressed. We then find out the number of transitions from each state to the others by observing the sequence of states. For example, let us consider the sequence of states is $\{......2, 4, 6, 2, 4......\}$. The subsequence $\{2, 4\}$ means that we increment the number of transitions from state 2 to state 4 by one. The next transitions are from states 4 to 6, 6 to 2 followed by another transition



(a) QCIF set 1: Average Stalls

(b) QCIF set 1: Fairness

(c) QCIF set 2: Average Stalls

(d) QCIF set 2: Fairness

Fig. 8. Distribution of stalls with variation of slots for QCIF videos

from 2 to 4. Once all the transitions have been considered we normalize each entry (i..e say transition from state 1 to state 2) by the sum of all transitions from state 1. Following this approach the transition matrix is easily determined. For the vehicular mobility case we observed that the range of the received signal was such that the highest rate was never achieved. Most of the values of the received signal strength were confined to the first 4 states of the Markov Chain. This is primarily due to the fact that vehicular motion is confined to roads and we are not able to reach very close to the base station. In the pedestrian mobility scenario however, transitions among all the 6 states is observed.

D. Additional Experimental Results

1) Results for second set of QCIF videos: We present the results for the two sets of QCIF videos in Figure 8. The results, depicting the average number of stalls and the standard deviation of stalls, are shown in Figure 8. Keeping in mind the low mean bit-rate requirement of the QCIF videos, all the rates in the Markov channel model i.e. the number of bits received in a slot, were scaled down by 10. This scaling down is done to investigate the algorithm performance near the average provisioned and over provisioned cases, scenarios that are interesting in practice.

2) A note on our results on different buffering schemes: We would like to note here that irrespective of the client buffering strategy, the total duration of stalls remains approximately the same for all approaches as it is contingent on the amount of data transmitted by the base station to the client. In all the three playout strategies, the average number of stalls decreases as the x-axis increases. The total duration of each stall however increases. For example, the average number of stalls is 6.4 and 3.1 when the FPT duration is 1 and 5 seconds respectively. The average duration of each stall is 4.6 and 9.6 seconds respectively. This is expected because when the condition for resuming playout is less restrictive it is satisfied easily. Therefore the user resumes playout but is forced to stop more often because enough data has not been buffered.