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

Partha Dutta*, *Member, IEEE,* Anand Seetharam*, *Student Member, IEEE,* Vijay Arya, *Member, IEEE,* Jim Kurose, *Fellow, IEEE,* Malolan Chetlur, *Member, IEEE,* and Shivkumar Kalyanaraman, *Fellow, IEEE*

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 QoE management a challenging problem. In this paper, we investigate scheduling algorithms to transmit multiple video streams from a base station to mobile clients. We present an epoch-by-epoch framework to fairly allocate wireless transmission slots to streaming videos. In each epoch our scheme reduces the vulnerability to stalling by allocating slots to videos in a way that maximizes the minimum 'playout lead' across all videos. 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 scheduling algorithm. Our greedy algorithm is optimal when the channel quality of a user remains unchanged within an epoch. Our experimental results, based on public MPEG-4 video traces and wireless channel traces that we collected from a WiMAX test-bed, show that the lead-aware greedy approach results in a fair distribution of stalls across the clients when compared to other algorithms, while still maintaining similar or fewer average number of stalls per client.

Index Terms—Video streaming, quality-of-experience, playout buffer management, base station scheduling.

1 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 video 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 (Figure 1). The server stores pre-encoded videos, and upon receiving requests, streams 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 and/or may *stall* the video to wait for more frames to arrive. Here we consider systems that stall in response to delayed frames. Namely, we consider the general case of VBR videos being streamed with the rate available to each wireless client varying over time.

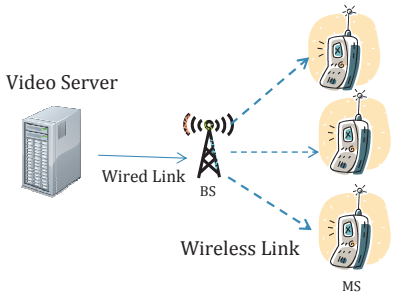


Fig. 1. A video streaming system

In this paper, we consider a wireless video streaming system where multiple mobile clients are streaming different VBR videos from a base station. Our goal is to develop a fair packet scheduling algorithm at the base station, for packet transmission over the wireless channel that minimizes playout stalls across all mobile clients. We assume that time is divided into slots and scheduling decisions are taken at beginning of an epoch (which consists of multiple slots). Prior work [2], [3] that studies the impact of video quality on user behavior demonstrate quantitatively (based on real world datasets) that frequent stalling can result

- *The first two authors are primary authors of this work.
- P. Dutta is with Xerox Research Center, India, V. Arya and M. Chetlur are with IBM Research, India, S. Kalyanaraman is with IBM Research, Australia. This work was done when all the above authors were at IBM Research, India.
  E-mail: Partha.Dutta@xerox.com,{vijay.arya, mchetlur, shivkumark}@in.ibm.com
- A. Seetharam, and J. Kurose are with the School of Computer Science, University of Massachusetts, Amherst, USA.
  E-mail: {anand, kurose}@cs.umass.edu

in users abandoning their video streams. The number of stalls per client thus appears to be a good metric to capture the quality of user experience and minimizing it can lead to reduced user abandonment.

We formulate this problem as an optimization problem that takes into account the varying rate of the video streams and wireless channel at the clients and allocates slots so as to maximize the minimum playout lead among all videos in an epoch. Our contributions are as follows. (a) We show that the optimization problem of maximizing the minimum lead is NP-complete even for two videos. (b) We develop a fast lead-aware greedy scheduling algorithm that is sub-optimal for wireless channels, but show that this algorithm is optimal when the channel quality of a user does not vary within an epoch, even with different users possibly having different channel quality. (c) 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 achieves 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 as compared to the total average bit-rate of the considered videos (a case that is interesting in practice), the greedy algorithm reduces the number of stalls by a factor of 3, when compared to other algorithms in our simulations. We also study the sensitivity of the greedy algorithm against changes in epoch duration, client's stall-recovery scheme, different video traces and poor channel conditions.

The remainder of this paper is organized as follows. Sections 3, and 4 describe the video streaming system characteristics and develop the scheduling problem formulation respectively. Hardness results are given in Section 5 followed by the greedy algorithm in Section 6. The evaluation framework and results for the experiments are given in Section 7 and Section 8, respectively. Comparison with related work is presented in Section 2. We discuss the adaptability and scalability of the greedy algorithm in Section 9 and conclude the paper in Section 10.

2 Related Work

Although compression techniques reduce the mean bit rate of video streams, it introduces considerable rate variability over several time scales [4], [5]. Resource allocation for VBR video streaming has been studied extensively for wired networks. Smoothing 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 bit rate variability under various resource constraints, such as peak bit rate, client buffer size, and initial playout delay [6], [7], [8], [9].

Rate allocation for multiple video streams is a well studied problem [10], [11], [12], [13], [14]. [10] 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 [11], the rate of each video is renegotiated at the end of each interval to provide statistical QoS guarantees. [12] presents a call-admission scheme at a statistical multiplexer and bounds the aggregate loss probability. A linear programming model is proposed in [13] to compute a globally optimized smoothing scheme to stream multiple videos. [14] 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 performs 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.

Scheduling algorithms for improving user QoE in cellular networks have also been designed ([15], [16] and the references therein). Our work is closely related to [15], [16], where the authors have proposed greedy algorithms for optimizing Mean Opinion Score (MOS) for resource allocation in wireless networks (3G and LTE). The main difference between our work and the above mentioned papers is the user QoE metric - we specifically consider video stalling whereas they mainly consider MOS. Another aspect that we consider in this work which is not explored in [15], [16] is that we demonstrate the hardness of our scheduling problem. In [17], the authors consider the problem of transmitting multiple VBR videos to mobile clients, but the work focusses on maximizing bandwidth utilization while reducing energy consumption and does not to address the issue of stalling of video playout.

Our work is closest to the work presented in [18], [19] 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. [19] develops an analytical framework to find the stall distribution while streaming a VBR video over a wireless channel. However, unlike our work, both papers consider a single video stream.

Gracefully degrading the quality of the displayed video when network conditions deteriorate is an active area of research. Scalable video coding for [20], [21], [22] and prioritization of packets [23] are two such methods used for video streaming. Recently, there have been measurement studies on the quality of videos streamed over deployed WiMAX networks [24]. The authors in [25] compare video streaming over a WiMAX network and a wired broadband network (with equal reserved rates), and demonstrate that with fine-tuning of network parameters, per-

formance over WiMAX is comparable to the wired networks in terms of the network QoS metrics. Recently, in [26], authors have investigated the impact of WiMAX network parameters on the end-user's QoE in video streaming. However, none of these papers consider mechanisms to multiplex video streams.

3 Network Model

In this section we describe the video streaming system and our wireless channel model.

3.1 Streaming system characteristics

We consider a video streaming system similar to [19], as shown in Figure 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 frame rate by the client. However, since the size of each frame varies significantly, the required transmission rate of a video varies with time.

For a video v_i, its *playback curve* $p_i(t)$ specifies the cumulative data needed in the first t time units of the video playout, in order to play the video without interruptions. Thus, $p_i(t)$ is the sum of the sizes of the first tF frames of the video, where F denotes the frame rate. The playback curve is a characteristic of a video and is independent of the underlying channel.

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* $O_i(t)$. Note that $G_i(t)$ and $O_i(t)$ depend on the channel conditions and transmission scheme at the base station, and $O_i(t)$ additionally depends on the buffering scheme of the client. In particular, unlike the playback curve, the playout curve may vary between different streaming instances of the same video. Figure 2(a) shows an example playback, receiver, and playout curve for a client. The notation used in this paper is summarized in Table 1.

We assume that clients have sufficient buffer space to buffer frames that have been received but not yet displayed. If the next frame to be displayed is not received within 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.

3.2 Timing consideration

We assume a broadband wireless system (such as WiMAX/LTE) in which scheduling decisions are taken at the time granularity of *epochs*. Epochs are divided into *intervals* (Figure 2(b)). The duration of an interval is small enough so that the channel state does not change significantly within an interval. 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 channel conditions, each client receives a certain bit rate in the allocated slots. Following [19], we assume that the wireless channel is error-free due to an error control mechanism such as ARQ.

Notation	Definition
n	number of clients
p_i, G_i, O_i	playback, receiver and playout curves (resp.)
R, A	channel rate vector and transition matrix (resp.)
$N_{ep}^{in}, N_{in}^{sl}, N_{ep}^{sl}$	#intervals/epoch, #slots/interval and #slots/epoch (resp.)
I_i	initial probability distribution of channel state
F	frames played out per second
Y_i, V_i	#bits and #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 1
Important notations (note: subscript i refers to client i and # denotes 'number of')

3.3 Channel model

We model the wireless channel between each client and the base station (i.e., bit rate received at the client), as a discrete-time Markov chain. We assume that the Markov chain changes state at the beginning of an interval. The possible channel states are identified by the transmission rates $R = (r_1, r_2, \ldots, r_K)$ (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 [19]. As the Markov chain changes state at the beginning of each interval, the bit rate for a client remains the same in all slots within an interval. Let A denote the transition matrix of the Markov chain. We assume A is available at the server, with each client's channel modeled as an independent Markov chain.

4 Modeling the Scheduling Problem

Our goal in this paper is to design a scheduling algorithm that executes periodically (at the beginning of each epoch) at the base station. Informally, the goal of the scheduling algorithm is to transmit video data to clients (some clients being allocated more transmission slots in an interval than others) in order to minimize playout stalls among all clients; we will precisely formulate this optimization problem shortly.

Minimizing the number of stalls within an epoch directly is difficult as it can incur high complexity. To

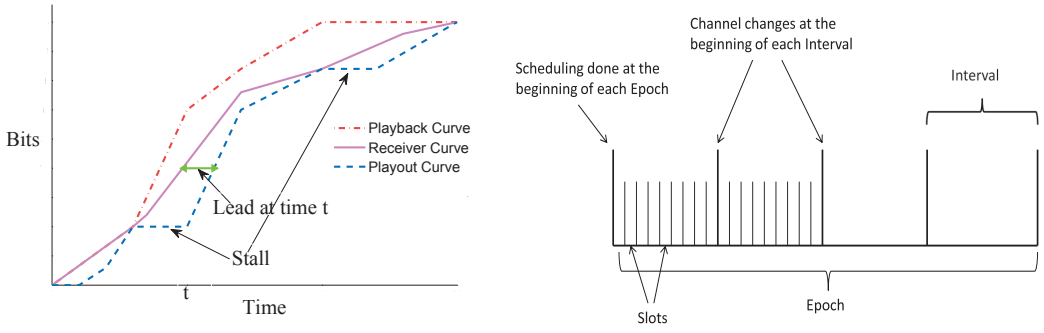


Fig. 2. (a) Playback, receiver and playout curves of a video stream (b) Epochs, Intervals, Slots

determine whether a client will stall or not during an epoch, it is necessary to determine the probability of the client receiving a specific number of bits during that epoch. Computing this probability is hard as one has to deal with summation of dependent random variables (the number of bits received in an interval for a client follows a Markov chain). We discuss this issue further, later in this section.

To motivate our strategy for allocating base station transmission slots to clients, we 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, a client's current buffer size may be an inaccurate 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 currently buffered data, takes into account the VBR nature of the video.

Therefore, in our scheme the server attempts to prevent stalls by fairly maximizing the playout lead among all receivers. To ensure that stalls are evenly distributed across all videos, slots are allocated such that the minimum lead among all clients is maximized. In contrast, if the scheduler goal were to maximize the minimum current buffer size (in bits), it would refrain from allocating bits to a client with large buffer, with the effect that this client could stall multiple times in succession if that large number of bits corresponded to short amount of played-out video. Indeed, we will see later that using current buffer size as the optimization metric can result in non-uniform allocation of stalls.

To perform this scheduling algorithm, we assume that at the beginning of each epoch, clients communicate their channel state to the server, as already done in numerous wireless standards. Clients also communicate their playout lead to the server. The initial state of the client's channel and the transition matrix of the Markov chain is used to determine the expected rate available to clients in different intervals during an epoch.

We do not consider client channel state during previous epochs while scheduling slots for the current epoch. As the server obtains fresh client channel state at the beginning of each epoch along with client playout lead, considering client channel information from previous epochs does not provide any additional value given a Markovian channel model. We also do not consider subsequent epochs because wireless channel prediction for longer than an epoch may not be accurate.

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 is $N_{ep}^{sl} = N_{ep}^{in} N_{in}^{sl}$. Each video is played at the constant rate of F frames per second.

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 probability distribution of the channel state at the client at the beginning of the k^{th} interval in the epoch is $I_i A^k$.

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 $I_i A^k$ and the channel transmission rate vector R. Suppose that the server assigns 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, $E[Y_i] = \sum_{k=1}^{N_{ep}^{in}} s_{ik} E[X_{ik}] = \sum_{k=1}^{N_{ep}^{in}} s_{ik} I_i A^k R$.

Before proceeding further, we discuss briefly why determining the probability of client i stalling (p_i) in an epoch is computationally expensive. Let c_i be the amount of data that client i has to receive by the end of the epoch to avoid stalling. Then $p_i = 1 - P[Y_i > c_i] = 1 - P[\sum_{k=1}^{N_{ep}^{in}} s_{ik} X_{ik} > c_i]$. It is difficult to determine the above probability because X_{ik} are dependent random variables. To determine the probability of a client stalling, it is necessary to determine the joint distribution of $(X_{i1}, X_{i2}, ...X_{iN_{ep}^{in}})$

which is computationally expensive.

Playout Lead: The playout lead of a client at any given time being the additional duration of time that its video can be played out using the data available in its buffer, it is equal to the number of complete frames in the client buffer divided by the frame rate F. Let l_i denote the playout lead of client i at the beginning of an epoch. Let o_i denote the total amount of *time* for which the video has been played out at the client i (calculated from the playout curve). Let g_i be the time for which the data received at the client can be played out (calculated from the playout curve). Thus $l_i = g_i - o_i$, is a known constant value at the beginning of the epoch. Note that o_i and g_i account for the data consumption at the client and the amount of data received during the previous epoch, respectively. In Figure 2(a), the green bar denotes the playout lead for the video at time t.

Let L_i be the *random variable* denoting the playout lead of the video at the end of an epoch (assuming that the video stalls during the epoch), and V_i be the *random variable* denoting the number of additional frames that can be *completely* received by the end of the current epoch. Then, $L_i = l_i + (V_i/F)$.

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 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.)

Estimating expected playout lead: We know that $L_i = l_i + (V_i/F)$. As l_i is a known constant at the beginning of an epoch, $E[L_i] = l_i + E[V_i]/F$. Now $E[V_i] = E[\Phi_i(Y_i)]$. Unfortunately, since the video frame sizes can vary, the mapping Φ_i from bits to frames is non-linear, and hence, we approximate $E[V_i] \approx \Phi_i(E[Y_i])$. The main benefit of this approximation is that computation of $\Phi_i(E[Y_i])$ is simple, making the execution of our greedy algorithm in Section 6 fast. Thus the expected lead is estimated as $E[L_i] \approx l_i + (1/F)\Phi_i(E[Y_i]) = l_i + (1/F)\Phi_i(\sum_{k=1}^{N_{ep}^{in}} s_{ik} I_i A^k R)$.

The Multiplexing Problem: Our aim, 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:

$$\begin{aligned} &\textit{Objective: } \max \min\{E[L_1], \ldots, E[L_n]\} \\ &\text{subject to the constraints:} \\ &1.\ \textstyle\sum_{i=1}^{n} s_{ik} = N_{in}^{sl},\ \forall k \le N_{ep}^{in} \\ &2.\ s_{ik} \ge 0\ ,\ \forall i \le n, \forall k \le N_{ep}^{in} \end{aligned} \tag{1}$$

5 Hardness Result

We now investigate the multiplexing problem described in the previous section. We formulate it as a combinatorial problem and call it *Lead-based Multiple Video Transmission (LMVT)* problem. (We assume that all slots from all intervals of an epoch are numbered sequentially from 1 to N_{ep}^{sl}.)

Inputs. At the beginning of an epoch, the i^{th} client has an initial lead of l_i seconds i.e., its buffer contains data corresponding to the $F * l_i$ frames received after the last played frame. Let r_{ij} be the expected number of bits that can be transmitted to client i in slot j. Thus if slot j belongs to interval k, then $r_{ij} = I_i A^k R$. For ease of presentation, we also call r_{ij} the rate of client i in slot j.

The LMVT Problem. Given the above inputs, we need to find a slot allocation that maximizes the minimum lead among all clients at the end of the epoch. Here, 'lead' refers to the expected playout lead (1). A slot allocation for an epoch essentially specifies for each slot, the client to which that slot is allocated.

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 all videos have a lead of at least L seconds at the end of the epoch?

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

Proof: Clearly the decision version of LMVT is in NP. We show that the problem is NP-complete by reducing subset-sum [27] to LMVT. The decision version of subset-sum is as follows: given a set S of positive integers $\{x_1, \ldots, x_P\}$, and a positive integer B, does there exist a subset $S' \subseteq S$ such that the sum of elements in S' is exactly B [27]. Let Π denote the index set $\{1, \ldots, P\}$ and let $Y = \sum_{j \in \Pi} x_j$. It is assumed $B < Y$, otherwise the subset-sum instance is trivial to solve.

For an instance of subset-sum, we construct an instance of LMVT as follows. Let Π be the set of slots in the epoch with one slot per interval. Let there be two videos v_1 and v_2. Let the set S map to the rates available in each slot as follows. Let x_j be the rate available to both the videos in slot j i.e., $x_j = r_{1j} = r_{2j}$. Let the initial lead for both the videos be zero and both play at the rate of 1 frame/second. Let the inverse playback curve of v_1, $\Phi_1(b)$, be a function which is 0 for $b < B$, and 1 for $b \ge B$. 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 $b \ge Y - B$. Let the required minimum lead L for each video be 1. We now show that the above instance of subset-sum has a solution if and only if the constructed instance of LMVT has a solution.

Subset-sum to LMVT: 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 $j \in \Pi$, 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 at the end of the epoch.

LMVT to Subset-sum: 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. □

For a constant number of videos, we have designed a pseudo-polynomial time algorithm to optimally solve LMVT using dynamic programming. The time complexity of the dynamic programming algorithm is high; it is exponential in the number of videos.

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

Let us now present an optimal dynamic programming algorithm for LMVT. We present a brief description of the algorithm here while a detailed proof is presented in the Appendix (in Supplement Material).

We begin by introducing a simple definition. A transmission vector (or *Tx-vector*) is an n-tuple $< a_1, \ldots, a_n >$, where the i^{th} element indicates the number of bits to be transmitted to video i. For a Tx-vector T, we denote by $T[i]$ the i^{th} element of T. For a given number of total slots, say z, and a Tx-vector T, we say that T is z-feasible if there is a slot allocation such that, for each $1 \leq i \leq n$, video v_i receives a total of $T[i]$ bits in the allocation.

Our dynamic programming algorithm iterates over the number of slots m that varies from 1 to N_{ep}^{sl} and determines the feasible Tx-vectors. In each iteration (say for slot m), the algorithm does the following. 1) It computes the m-feasibility of the Tx-vectors based on the $(m-1)$-feasibility of a subset of Tx-vectors (computed in the previous step). 2) For each feasible Tx-vector for slot m, then computes the minimum lead considering all videos. 3) For each feasible Tx-vector T, stores an *allocation* pointer to the Tx-vector from the previous step from which its m-feasibility was computed.

Finally, in the iteration when $m = N_{ep}^{sl}$, we maintain a pointer to determine the N_{ep}^{sl}-feasible vector with the maximum value of its min-lead among all the N_{ep}^{sl}-feasible vectors. Thus, at the end of algorithm, we obtain a pointer to a N_{ep}^{sl}-feasible Tx-vector T' with the maximum value of min-lead, and we follow the N_{ep}^{sl} *allocation* pointers from T' to $< 0, \ldots, 0 >$ to obtain an optimal slot allocation.

6 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 to be allocated, the greedy algorithm allocates slots one by one (Figure 3) as follows. In each iteration, the algorithm selects a video i with the minimum expected 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 (yet to be scheduled) slots. Before moving to the next iteration, slot j is marked unavailable for all videos, and the expected 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. Note that, the client with the minimum lead that is selected by the algorithm may change between any two slot allocations. Hence, the algorithm allocates slots one by one even though each client's channel condition does not change within an interval.

Complexity analysis: The total number of slots considering all epochs and intervals is given by N_{ep}^{sl}. We now evaluate the runtime of the greedy algorithm in Figure 3.

Time Complexity: Initialization

- Lines 5 -7 : $O(\max(nN_{ep}^{sl}, nN_{in}^{sl}K^2))$. This is because the matrix multiplication $(I_i A^k R)$ will require $O(K^2)$ time.

Time Complexity: Greedy algorithm

- Line 9: $O(n)$
- Line 10 $O(N_{ep}^{sl})$
- Lines 11-13 $O(1)$ (assuming constant computation time for $\Phi(.)$)
- Lines 9-13 $O(N_{ep}^{sl}) + O(n) = O(\max(N_{ep}^{sl}, n))$ = $O(N_{ep}^{sl})$ (as $N_{ep}^{sl} > n$ usually)
- Lines 8-13 are executed (N_{ep}^{sl}) times and thus the greedy algorithm takes $O({N_{ep}^{sl}}^2)$.

Total Time Complexity : $O(\max({N_{ep}^{sl}}^2, nN_{in}^{sl}K^2))$. We provide further details in the Appendix (in Supplement Material).

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: $\forall$ client i: **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: $r_{ij} \leftarrow I_i A^k R$; $j \leftarrow j + 1$
8: **function** greedy algorithm: **while** $AvailableSlots \neq 0$ **do**
9: select a client with the lowest id i s.t. $(\forall q \leq n, lead_i \leq lead_q)$
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)

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.

7 Experimental Setup

7.1 Scheduling Algorithm: Parameters

To evaluate our epoch-by-epoch scheduling strategy based on playout lead we need to specify the epoch duration, interval size and the number of slots in an interval. Recall that in our scheduling strategy, epochs are divided into intervals, which are subdivided into slots (Figure 2(b)).

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 (except Figure 6 where we vary the epoch duration). 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 transmission time is typically large with respect to the fast fading timescale). In our experiments, we vary the number of slots in an interval. By varying the number of slots in an interval we can vary the total resource (in terms of bandwidth) available at the base station because the rates in our Markov model correspond to the number of bits received in a slot.

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 across a broad range of epoch durations, interval sizes and 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.

We assume the following buffering scheme at the client - if the client does not have enough data to playout for the whole duration of the epoch, it stalls for the entire epoch. We also assume that the clients have infinite large buffers to store all received packets.

7.2 Trace-Driven Experiments

To demonstrate the efficacy of the greedy algorithm, we perform trace-driven experiments. Our evaluation uses two types of traces:

(i) VBR Video Traces that provide the variation in the frame sizes of videos for emulating video playouts.
(ii) User-Level Wireless Channel Traces that provide the rates achieved by different users in every interval of each epoch.

7.2.1 VBR Video Traces

We use the publicly available MPEG-4 *VBR Video Traces* [28], [29] 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 in a scenario where 8 different videos are being simultaneously streamed to 8 different users over the shared wireless infrastructure. Unless mentioned otherwise, all results are reported for CIF videos. A brief description of the 8 CIF video traces used, is given in Table 2. The duration of the videos used in our experiments is approximately 27 minutes. Detailed information about the CIF and QCIF traces is available in [29].

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 2
CIF video trace statistics

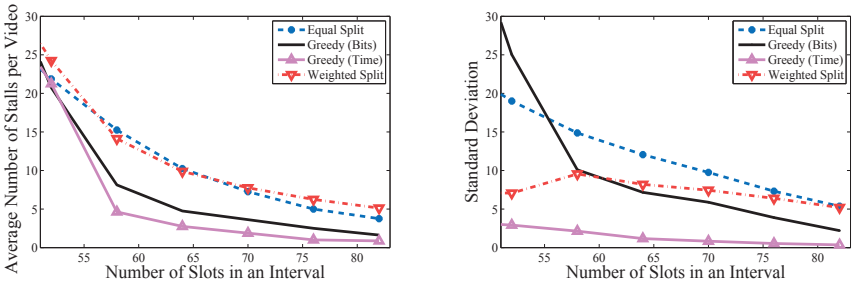
7.2.2 User-Level Wireless Channel Traces

Signal Strength Measurement: The wireless channel traces we use were obtained from signal strength measurements over a (802.16e) WiMAX network deployed in WINLAB at Rutgers University. The WiMAX base station is installed in WINLAB. During our trace collection, the base station continuously transmitted data packets, and signal strength (RSSI) was recorded at the receiver (a laptop) under vehicular and pedestrian mobility. As our interval duration is 1 second, we obtain signal strength quality one second apart from each another. To eliminate any fast fading effects, we consider the average signal strength at the beginning of each second. Additional details are available in [30]. A brief description of the parameters of the WiMAX network used in our trace collection is given in Table 3. The vehicular mobility traces were collected by driving a car around the campus multiple times while the pedestrian mobility experiments were performed by walking around the same campus. We conducted 4 vehicular and 4 pedestrian mobility experiments, each of duration approximately 10 minutes. As the base station only has a range of 500m, the entire range was effectively covered by these experiments.

RSSI-Rate Mapping: To obtain a mapping between the RSSI values and the rates achieved, we use the mapping between the modulation and coding schemes (MCS) and the SINR values for a WiMAX network provided in [31]. A common approach is to divide the SINR regime into a number of ranges and for each range there exists an MCS that maximizes throughput. The MCS indicate the rates achievable in practice. Six different rates are achievable in practice and they have the following ratio $[1, 1.5, 2, 3, 4, 4.5]$ [31]. As mentioned earlier, our base station reports RSSI values, which is similar to the SINR values reported in [31]. The minimum and maximum values of RSSI measured in our experiments are -85 dBm and -37 dBm and we map them to the corresponding SINR values in [31]. We use linear extrapolation to determine the mapping between RSSI ranges and the rates achieved. We use the RSSI-rate mapping to generate the *rate traces* (i.e., traces indicating the rates achieved over time) for the vehicular and pedestrian mobility experiments. We then generate 8 different *User-Level Wireless Channel Traces* (each 27 minutes long) emulating the real channel conditions (separately for vehicular and pedestrian mobility) from the *rate traces*.

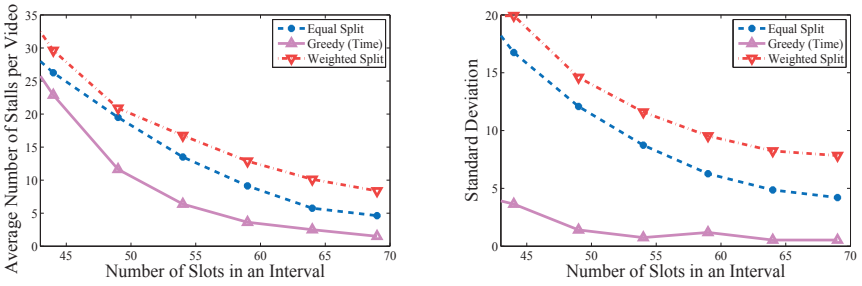
Markov Chain Model: Our Markov channel model has 6 different states corresponding to the rates achieved. The states of our Markov model correspond to the number of bits successfully transmitted in a slot. The vector of transmission rates is taken to be $R = [1, 1.5, 2, 3, 4, 4.5] * 50000$ bits for the CIF videos. SNR based Markov chain models describing the wireless channel have been well studied in literature. [27] provides a detailed description of the various models available in literature. Similarly the use of SNR to bit rate mapping is also common [28], [29]. We determine the transition matrix of the Markov chain empirically (from the *rate traces*) by counting the number of transitions from one state (say i) to other states and then normalizing them by the total number of transitions from state i.

We note here that after about 40 steps (i.e., 40 seconds), the probability distribution obtained from any starting state using the transition matrix reaches very close (5%) to the steady state distribution for both vehicular and pedestrian mobility scenarios. Therefore, the transition matrix does not reach steady state during the duration of an epoch (which is 10 seconds) and is thus useful as a prediction mechanism for making scheduling decisions.



(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. Mixture of vehicular and pedestrian mobility: Distribution of stalls with variation of wireless channel resource (slots) for CIF videos

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 3
WiMAX system parameters for trace collection

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

TABLE 4
Expected steady state bit rate vs. number of slots

8 Results

In this section we present and discuss 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 unfair, and have lower overall performance 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 6 (we denote our algorithm from Section 6 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 (buffer size) 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.

8.1 Distribution of Stalls

In this subsection we study stall distribution as a function of the number of slots in an interval (keeping the interval duration constant). Using the steady state probabilities of the Markov model, one can compute the expected number of bits received per slot. In Figures 4 and 5, the number of slots is varied from 50 to 80, so that the slot duration corresponds roughly to the frame duration.

8.1.1 Vehicular Mobility

Figure 4 shows the variation of the average number of stalls for four scheduling algorithms: equal-split, weighted-split, greedy-bit and greedy-time. Table 4

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

TABLE 5
stalls per video for average-provisioned network

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 8 CIF videos is approximately 5.4 Mbps. Thus, from Table 4, we note that 34, 58 and 82 slots per interval correspond to the wireless channel being 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 for the average and over-provisioned scenarios. 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. We observe that the greedy-bit algorithm is unfair in distributing stalls (Figure 4), and so we will not consider this algorithm 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 5. The mildly over-provisioned case is the scenario of interest in practice and we observe that the greedy-time 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.

8.1.2 Pedestrian Mobility

We also conducted experiments under pedestrian mobility. We observe that the greedy-time algorithm again outperforms the equal and weighted split algorithms in terms of both average number of stalls and fairness. Due to lack of space we omit the figures; they are available in [32].

8.1.3 Mix of Vehicular and Pedestrian Mobility

In practical situations, we will usually have a mix of pedestrian and vehicular users, streaming different videos from the base station. Figure 5 shows the simulation results considering 4 vehicular and 4 pedestrian users. We observe that the greedy-time algorithm outperforms the other two schemes. Interestingly, in Figure 5(b), the weighted split algorithm has higher standard deviation when compared to the vehicular (Figure 4(b)) and pedestrian mobility scenarios. This is because unlike the vehicular and pedestrian mobility cases, where all users have similar channel quality, in Figure 5(b) we have both pedestrian and vehicular users and the weighted split approach (which divides the number of available slots proportional to the mean bit rate of the videos without taking the channel conditions into account) results in unfair distribution of stalls. In contrast to this, the greedy-time heuristic continues to distribute the stalls fairly. We note that similar to Figure 4(b), the greedy-bit algorithm is unfair in distributing the stalls for the experiments conducted in sections 8.1.2 and 8.1.3 as well. Due to lack of space, in the remaining sections we only present the results for the vehicular mobility case.

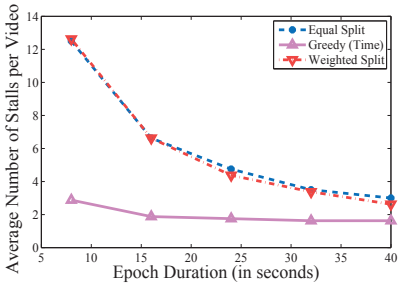


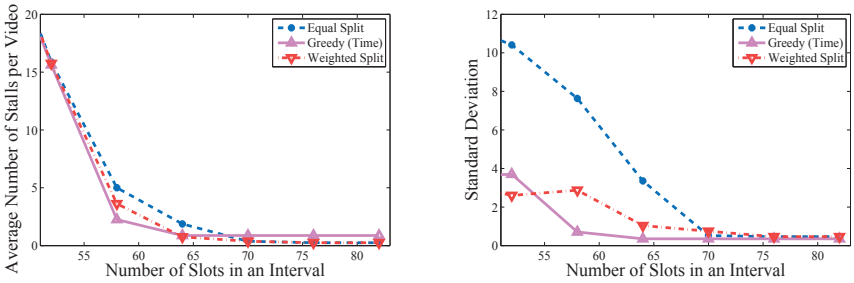
Fig. 6. Sensitivity to epoch duration

8.2 Sensitivity to Epoch Duration

In the experiments presented thus far, the epoch duration was fixed at 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 the epoch duration increases, the number of stalls for the other schemes decreases faster in comparison to the greedy scheme. This is because as the greedy scheme starts with a significantly lower number of stalls, increasing epoch duration does not benefit it much. We note, however, the total stall duration averaged over all videos increases with increasing epoch duration.

8.3 Sensitivity to Different Video Traces

We also conducted experiments with two sets of 8 QCIF video traces, available from [28], [29]. We show results for one set of QCIF videos here while the other one is available in the Appendix (in Supplement Material). The results, plotting the average number of stalls and the standard deviation of stalls versus the number of slots in an interval, are shown in Figure 7. Given 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 algorithm performance near the average provisioned and mildly over provisioned cases, which are the scenarios that are interesting in



(a) QCIF set 1: Average Stalls (b) QCIF set 1: Fairness

Fig. 7. Distribution of stalls with variation of slots for QCIF videos

practice. For QCIF videos, we observe that the greedy-time algorithm outperforms the other approaches in terms of fairness, but its performance is similar to the weighted split algorithm in terms of average number of stalls. Note that when the number of slots in an interval is larger than 65 (this corresponds to the highly overprovisioned case), the other algorithms slightly outperform the greedy algorithm. The total number of stalls experienced by any video in this case is only 0 or 1; the difference between the algorithms is that some videos experience a stall in case of the greedy algorithm while no stalls occur for the other algorithms.

8.4 Sensitivity to Poor Channel Condition

A potential drawback of maximizing the minimum playout lead is the case where some clients have poor channel condition for a protracted period of time. Maximizing the minimum playout lead in this situation can degrade entire system performance. One way to tackle this issue is to restrict the maximum number of slots that can be allocated to any user.

For simulations we consider a vehicular mobility scenario where two out of eight clients have poor channel quality: these clients transition only between the lowest two rates of the Markov model with probability 0.5. Since we do not have real world traces mimicking this kind of channel behavior we create synthetic traces for these two users. For generating the synthetic traces we assume that in any interval, each of two users can be in one of the two lowest rates with probability 0.5. Figure 8 shows the result for this simulation. The plot x-Thd in the figure signifies our greedy-time algorithm with the modification that the maximum number of slots allowed for any client is $x\frac{TotalSlots}{n}$, where n is the number of videos.

We observe that if there is no restriction on the maximum number of slots allocated for a client (i.e., our original greedy-time algorithm), the algorithm performs worse than the baseline approaches with respect to the average number of stalls when the number of slots is small and has superior performance for the overprovisioned case. We can observe from Figure 8 that clearly there is a tradeoff in the performance of the greedy-time algorithm between the number of slots in an interval and the threshold imposed. The greedy-time algorithm with a low threshold performs best when the number of slots is small, while the opposite is true when the number of slots is large. As expected, as the threshold is increased, the performance of the greedy-time algorithm tends to the original algorithm with no threshold. In terms of standard deviation, as expected the Greedy (Original) algorithm performs best with the standard deviation increasing as we impose a lower threshold. Overall we observe that the $2.0x$-Thd greedy algorithm performs the best for the scenario chosen in this experiment.

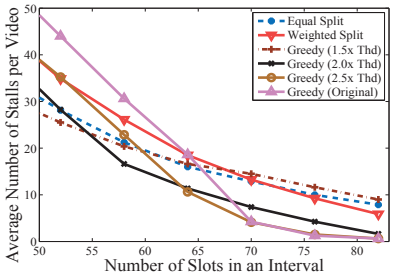


Fig. 8. Effect of poor channel quality

We also performed experiments with different client buffering schemes. Due to lack of space we omit them here; they are available in [32].

9 Discussion

In this section we discuss issues related to the adaptability and scalability of the greedy algorithm. In this paper we have only considered video streaming applications, but our algorithm can also be adapted for the case when there is other concurrent traffic through the base station. The other applications will consume a fraction of the base station resources (time slots in this case); the QoE requirement of these applications being different from video streaming, our greedy algorithm can execute on the remaining timeslots (after timeslots required by other applications have been allocated). Since our approach operates using the slots available to video, it would find application in any scheme (even dynamic) in which a provider coarsely partitioned slots among applications.

Our greedy algorithm can also be adapted to work with two dimensional (time-slot and sub-carrier) allocation of data - as such the model can be enriched by having a separate Markov chain for the wireless channel on each subcarrier. The expected rate received in the various time slots for the different subcarriers can be determined using the Markov chains. Note that our greedy algorithm assumes that channel quality remains unchanged within an interval. So long this assumption holds, the greedy algorithm can be applied (it does not matter whether slots within an interval are divided in time domain, frequency domain or both).

We have also not considered the scenario where users can join/depart in the middle of an epoch. Our algorithm can easily be adapted to this situation. Users departing from the system will cause resources (slots) allocated to them for that epoch to be unused. This issue can be dealt with by randomly allocating the freed slots in the epoch among the different clients. If a new user joins in the middle of an epoch, this user will not have data sent to it during that epoch because all slots have already been allocated to other users a priori. This will cause an additional delay (with maximum duration of one epoch) to the new user. However in the beginning of the next epoch, this user will be given preference by the greedy algorithm (and thereby more slots allocated to it) as it will have playout lead equal to zero.

In this paper we address the problem of streaming stored video to various clients. The stored video might be considered as videos cached at devices at the edge of the telecommunication network. The video playback curve is just the set of frame sizes. This information regarding frame sizes can be made easily available at the base station. For example, Netflix manifest files already contain this information on a per-chunk basis, where a chunk is approximately 4 seconds of data [33]. As the frame rate is only 30 frames/sec, the amount of information that is to be stored per video is not quite small (in the order of a few Mbits). Hence the memory required for storing video playout information is small. The runtime of the algorithm $O({N_{ep}^{sl}}^2)$ and thus the greedy algorithm is easily scalable. Nowadays computational power is available at the base station [34] and thus base stations should be able to periodically execute the low complexity greedy algorithm.

Another issue might be the communication overhead for the greedy algorithm. Though overhead is not explicitly modeled in this paper, the information required to be communicated by each client at the beginning of an epoch is only the playout lead and the current channel state (which is only a few bytes of information per client).

10 Conclusion

In this paper, we investigated scheduling schemes for transmitting multiple video streams from a base station to mobile clients. We showed that the problem of allocating slots fairly is NP-complete even for a constant number of videos. We then presented a greedy algorithm based on a criterion of maximizing the minimum playout lead to manage stalls for multiple video streams transmitted over a time-varying bandwidth-constrained wireless channel. We demonstrated that the greedy algorithm is fair and is also capable of minimizing the average number of playout stalls.

References

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

[2] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the impact of video quality on user engagement," in *SIGCOMM*, 2011.

[3] S. S. Krishnan and R. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," in *IMC*, 2012.

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

[5] 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.

[6] 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.

[7] T. Ott, T. V. Lakshman, and A. Tabatabai, "A scheme for smoothing delay-sensitive traffic offered to ATM networks," in *IEEE INFOCOM*, 1992.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] S. Thakolsri, S. Khan, E. Steinbach, and W. Kellerer, "Qoe-driven cross-layer optimization for high speed downlink packet access," *Journal of Communications*, vol. 4, no. 9, 2009.

[16] M. Shehada, S. Thakolsri, Z. Despotovic, and W. Kellerer, "Qoe-based cross-layer optimization for video delivery in long term evolution mobile networks," in *WPMC*, 2011.

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

[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] ——, "Balancing interruption frequency and buffering penalties in VBR video streaming," in *INFOCOM*, 2007.

[20] H. Radha, M. van der Schaar, and Y. Chen, "The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP," *IEEE Transactions on Multimedia*, vol. 3, no. 1, 2001.

[21] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 17, no. 9, 2007.

[22] G.-M. Su and M. Wu, "Efficient bandwidth resource allocation for low-delay multiuser video streaming," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 15, no. 9, 2005.

[23] J. Huang, C. Krasic, J. Walpole, and W. chi Feng, "Adaptive live video streaming by priority drop," in *IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2003.

[24] B. Sousa, K. Pentikousis, and M. Curado, "Experimental evaluation of multimedia services in WiMAX," in *International Mobile Multimedia Communications Conference (MobiMedia)*, 2008.

[25] W. Hrudey and L. Trajković, "Streaming video content over IEEE 802.16/WiMAX broadband access," in *OPNETWORK*, 2008.

[26] A. Vishwanath, P. Dutta, M. Chetlur, P. Gupta, S. Kalyanaraman, and A. Ghosh, "Perspectives on quality of experience for video streaming over WiMAX," *Mobile Computing and Communications Review*, vol. 13, no. 4, 2009.

[27] 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.

[28] 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.

[29] 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 availalble at http://trace.eas.asu.edu/mpeg4/index.html and QCIF Video traces available at http://trace.eas.asu.edu/cgi-bin/main.cgi.

[30] A. Seetharam, J. Kurose, D. Goeckel, and G. Bhanage, "A markov chain model for coarse timescale channel variation in an 802.16e wireless network," in *INFOCOM*, 2012.

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

[32] P. Dutta, A. Seetharam, V. Arya, M. Chetlur, S. Kalyanaraman, and J. Kurose, "On managing quality of experience of multiple video streams in wireless networks," in *INFOCOM*, 2012.

[33] V. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z. Zhang, "Unreeling netflix: Understanding and improving multi-cdn movie delivery," in *IEEE INFOCOM*, 2012.

[34] M. Chetlur, P. Dutta, U. Devi, P. Gupta, L. Chen, Z. Zhu, S. Kalyanaraman, and Y. Lin, "A software wimax medium access control layer using massively multithreaded processors," *IBM Journal of Research and Development*, vol. 54, 2010.

Partha Dutta is a Senior Researcher with Xerox Research Centre India. Before joining Xerox, he has held researcher positions with IBM India Research Lab and Bell Labs Research India. Partha received his B.Tech. from Indian Institute of Technology Kanpur (1999), and his Ph.D. from Swiss Federal Institute of Technology Lausanne (2005). His research interests are in networking and distributed computing, and his recent work is on network reservation and virtual machine placement in multi-tenant data centers. Partha is a member of IEEE and ACM.

Anand Seetharam is a doctoral student in the School of Computer Science at the University of Massachusetts (UMASS) Amherst. Prior to joining UMASS, he obtained his Bachelor's degree in Electronics and Telecommunication Engineering from Jadavpur University, India. His research interests encompass various aspects of wireless networks including mobile ad-hoc, sensor, cache and cellular networks.

Vijay Arya is a Research Staff Member at IBM Research - India since 2010. Prior to joining IBM, he worked at National ICT Australia (NICTA) and completed his doctoral studies in Computer Science from INRIA Sophia Antipolis, France in 2005. His research interests include measurements and modeling, computer networks, and smart grids.

Jim Kurose received his Ph.D. degree in computer science from Columbia University. He is currently Distinguished University Professor in the Department of Computer Science at the University of Massachusetts Amherst. His research interests include network protocols and architecture, network measurement, mobile networks, and modeling and performance evaluation. Dr. Kurose has served as Editor-in-Chief of the IEEE Transactions on Communications and was the founding Editor-in-Chief of the IEEE/ACM Transactions on Networking. He has served a Technical Program co-chair for IEEE Infocom, ACM SIGCOMM, ACM SIGMETRICS and ACM IMC. He has received the IEEE Infocom Achievement Award, the ACM SIGCOMM Test of Time Award and a number of teaching awards, including the IEEE Taylor Booth Education Medal. With Keith Ross, he is the co-author of the textbook, Computer Networking, a top down approach (6th edition) published by Addison-Wesley. He is a Fellow of the IEEE and ACM.

Malolan Chetlur is a Research Staff Member in IBM Research India (IRL). His research interests include Mobile and Telecom technologies, parallel and distributed simulation. Dr. Chetlur is a member of ACM and has served as the TPC of networking and cloud computing conferences (LANMAN2010-11, INFOCOM 2011-12, and IEEE-Cloud 2010-13). Dr. Chetlur received his M.S. and Ph.D. degrees in computer engineering from the University of Cincinnati. Prior to joining IBM Research, he was a Principal Technical Staff Member with AT&T, developing system automation solutions and enterprise solutions. He is currently exploring scalable personalized education methodologies enabled by IT as part of Smarter Education research.

Shivkumar Kalyanaraman (S93M97SM07 F10) received the B.Tech. degree in computer science from the Indian Institute of Technology, Madras, India, in 1993, the M.S. and Ph.D. degrees in computer and information sciences from the Ohio State University, Columbus, in 1994 and 1997, respectively. He also received the Executive M.B.A. degree from Rensselaer Polytechnic Institute, Troy, NY, in 2005. He is the Chief Scientist at IBM Research - Australia, and Co-Director of the UBD — IBM Centre, Brunei. Previously he was a Senior Manager, Smarter Planet Solutions with IBM Research India, Bangalore, India, and was a Professor with the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute. His research in IBM is at the intersection of emerging wireless technologies, smarter energy systems, and IBM middleware and systems technologies. Dr. Kalyanaraman is an ACM Distinguished Scientist. He is a Visiting Professor at Universiti Brunei Darussalam.