On Statistical Multiplexing of Variable-Bit-Rate Video Streams in Mobile Systems

Cheng-Hsin Hsu
School of Computing Science
Simon Fraser University
Surrey, BC, Canada

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

ABSTRACT

We consider the problem of broadcasting multiple variable-bit-rate (VBR) video streams from a base station to many mobile devices over a wireless network, so that: (i) perceived quality on mobile devices is maximized, (ii) bandwidth utilization is maximized, and (iii) energy consumption of mobile devices is minimized. We show that this problem is NP-Complete. We propose an approximation algorithm for the base station to statistically multiplex and transmit multiple VBR streams to achieve these objectives. We analytically analyze the performance of our algorithm and prove that it achieves optimal bandwidth utilization and near-optimal energy saving. Our algorithm frees network operators from the manual and error-prone bandwidth reservation process, which is usually used in practice for broadcasting VBR streams. We implement the proposed algorithm in a trace-driven simulator, and conduct extensive simulations. The simulation results show that our algorithm outperforms the existing algorithms in many aspects, including number of late frames, number of concurrently broadcast video streams, and energy saving of mobile devices. We also implement the proposed algorithm in a real testbed for video broadcasting as a proof of concept. The results from the testbed confirm that the proposed algorithm: (i) does not result in playout glitches, (ii) achieves high energy saving, and (iii) runs in real time.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless Communication*

General Terms

Design

1. INTRODUCTION

Fueled by technology advances, video streaming has been getting popular in the past decade, and will continue to thrive in the future [12]. In addition, more users watch streaming videos over wireless networks using mobile devices like laptops, PDAs (personal digital assistants), smart phones, and PMPs (portable media players). Streaming videos over wireless networks, however, is challenging for several reasons. First, streaming videos at high quality is difficult, because video streams have diverse and varying bit rates and impose stringent Quality-of-Service (QoS) requirements on underlying wireless networks, which are vulnerable to fading, shadowing, and interference [23]. Second, mobiles devices are sensitive to energy consumption, because they are battery powered. Since higher energy consumption leads to shorter watch time on mobile devices, energy conservation is important for good user experience. Third, and most important, the wireless spectrum is expensive, e.g., AT&T sold a WiMAX spectrum in the southeast US to Clearwire for \$300 million [2], and Inukshuk paid \$46 million to license a WiMAX spectrum in Canada [19]. Therefore, to be commercially viable, network operators must achieve high bandwidth utilization in terms of number of concurrent video streams in their wireless networks.

In this paper, we study the problem of broadcasting multiple VBR streams from a base station to a large number of mobile devices within its coverage area over a metropolitan wireless network. The wireless network can be a WiMAX, 3G cellular network, or a dedicated broadcast network for mobile TV services such as DVB-H (Digital Video Broadcast–Handheld) [10, 16] and MediaFLO (Forward Link Only) [11] networks. The goals of the proposed solutions are to: (i) maximize perceived quality on mobile devices, (ii) maximize bandwidth utilization, and (iii) minimize energy consumption of mobile devices. To save energy on mobile devices, the base station sends each video stream in bursts with a bit rate much higher than the instantaneous bit rates of that video stream. The next burst time is computed by the base station and included in the header fields of every burst, which enables mobile devices to receive a burst of traffic, and then put their wireless interfaces in sleep mode till the next burst to save energy. Fig. 1 shows a wireless broadcast network. As illustrated in this figure, a base station time multiplexes the bursts of all video streams following a *burst schedule*, which ensures that: (i) no two bursts overlap with each other in time and (ii) no buffer overflow instances occur on mobile devices. Creating optimal burst schedules is NP-hard, and thus we propose an approximation, real time, algorithm for the base station to statistically multiplex and transmit multiple VBR streams to achieve most of the three goals mentioned above.

We analytically analyze the performance of our algorithm, and prove that it achieves optimal bandwidth utilization, and produces near-optimal energy saving on mobile devices. We also show that the time complexity of our algorithm is low, and it can run in real time. We develop a trace-driven simulator for wireless broadcast networks, and we implement the proposed algorithm in it. We compare the performance of the proposed algorithm against other existing algorithms. The simulation results show that our algorithm out-

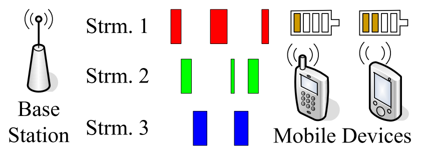


Figure 1: Broadcasting videos to mobile devices.

performs the existing algorithms in many aspects, including number of late frames, number of concurrently broadcast video streams, and energy saving of mobile devices. Finally, we implement the proposed algorithm in a real testbed for video broadcast networks that comply with the DVB-H standard in order to show its practicability and efficiency. The results from the testbed confirm that the proposed algorithm: (i) does not result in playout glitches, (ii) achieves high energy saving, and (iii) runs in real time.

We note that, in mobile *broadcast* networks, a base station composes burst schedules without considering wireless channel conditions of individual mobile devices. This is because each mobile broadcast network has numerous receivers, and thus maintaining feedback channels from these receivers to the base station is not practical. Therefore, a base station can not retransmit late or lost packets to mobile devices in poor wireless conditions. Modern mobile broadcast networks, e.g. [4,5,10,16], therefore employ forward error correction (FEC) and strong channel coding to combat fluctuating bandwidth and packet losses caused by diverse and changing wireless conditions. FEC and channel coding allow network operators to provide reliable broadcast services for mobile devices within the planned coverage map under a reasonable wireless condition.

To the best of our knowledge, optimally broadcasting multiple VBR streams over wireless networks has not been fully addressed in the literature. Most previous works, e.g., [9, 13, 15], only support constant-bit-rate (CBR) streams. In fact, recent papers [24,25] emphasize that achieving statistical multiplexing of VBR streams by burst scheduling is one of the most critical and challenging *open* problems in wireless broadcast networks.

The rest of this paper is organized as follows. Sec. 2 presents the related work in the literature and discusses limitations of current base stations. In Sec. 3, we define and formulate the considered problem. We present and analyze our algorithm in Sec. 4. We conduct extensive trace-driven simulations in Sec. 5, and we implement and evaluate the proposed solution in a real testbed in Sec. 6. Sec. 7 concludes this paper.

2. RELATED WORK AND BACKGROUND

2.1 Broadcasting VBR streams in Literature

Streaming VBR videos imposes challenges on the best-effort Internet, because insufficient network bandwidth leads to late frames, thus playout glitches. Several smoothing algorithms have been proposed in the literature, e.g., in [17, 18], which absorb bit rate variations of a VBR stream by adding buffers at both sender and receiver, and compute a constant-bit-rate (CBR) transmission schedule that results in no buffer over/underflow instances. Most smoothing algorithms are not designed for mobile networks that transmit videos in bursts to save energy, and thus cannot solve the problem considered in this paper. Camarda et al. [3] propose a smoothing algorithm for mobile broadcast networks. Their work is quite different from ours, because they consider the problem of assigning frames of a VBR stream some *predefined* transmission bursts, while we compute the optimal burst schedule. Furthermore, these smoothing algorithms [3, 6, 17, 18, 26] only consider a single VBR stream while our problem is to concurrently broadcast multiple video streams.

The energy saving of mobile devices in wireless broadcast networks that send videos in bursts has been considered in several works. For example, the authors of [29] and [9] study the energy saving achieved by a given burst schedule, but they do not solve the burst transmission problem. In [13, 15], we solve the burst transmission problem for CBR video streams. Unlike these two works, we solve the burst transmission problem for VBR video streams in this paper, which allow higher flexibility and lead to higher bandwidth utilization and energy saving for mobile devices.

Streaming multiple VBR streams over a wireless network using statistical multiplexing can be done in one of two ways: (i) rate control in video encoders, and (ii) burst scheduling on base stations. Rezael et al. [25] take the first approach and assume video encoders are collocated with the base station. They study the joint rate control problem among the encoders of multiple videos, and aim at achieving low end-to-end delays and small quality variance among videos. We take the second approach. The work in [25] is different from ours for two reasons. First, their approach does not take energy consumption into consideration, nor does it provide any analytical bounds on the performance. Our solution produces burst schedules that are optimal in terms of bandwidth utilization, and near-optimal in terms of energy saving of mobile devices. Second, and more importantly, we do not assume encoders are collocated at the base station nor that we have control of the encoders. More precisely, our algorithm constructs a burst schedule to broadcast a set of videos that can be remotely encoded or pre-encoded. This is more general as most videos, except live ones, are pre-encoded well before their transmission time with little information on the transportation medium that will carry them. Hence, our solution is applicable for both pre-encoded and live videos.

2.2 Broadcasting VBR streams in Practice

Despite the importance of broadcasting VBR streams, current base stations construct burst schedules in a fairly primitive way. For example, in Nokia Mobile Broadcast Solution (MBS) [1, 20], network operators specify a system-wide inter-burst time period ΔT sec, and a burst size b_s kb for each video stream s. The base station then schedules a burst every ΔT sec for each video stream s, where the burst size is b_s kb long. In such a base station, network operators have to *manually* choose ΔT and b_s to form a burst schedule that results in no bursts overlapping in time and no buffer overflow instances on mobile devices. This task is time consuming and error-prone. Network operators may choose a ΔT value as follows. They first relabel video streams such that $r_1 \leq r_2 \leq \cdots \leq r_S$, where r_s is the streaming bit rate of video stream s ($1 \leq s \leq S$). They next set $\Delta T = Q/r_S$ for higher energy saving, where Q kb is the buffer size of mobile devices. They then divide the air medium time ΔT among all video streams in proportional to their streaming bit rates. Finally, they choose r_s in one of two ways:

1. *VBR (variable-bit-rate).* Network operators may heuristically pick an r_s for each stream s and directly transmit VBR streams. There is a tradeoff between bandwidth utilization and video quality: choosing a high r_s leads to wasted bandwidth, but choosing a low r_s results in buffer underflow instances, and thus playout glitches. To better quantify this tradeoff, we define $F_s(r)$ as the CDF function of per-GoP (Group of Picture) bit rate of stream s. We define a VBR burst scheduling algorithm VBR$_\alpha$ as streaming each video stream s at the smallest bit rate r_s so that $F_s(r_s) \geq \alpha$.

2. *RVBR (regulated-variable-bit-rate).* Network operators may

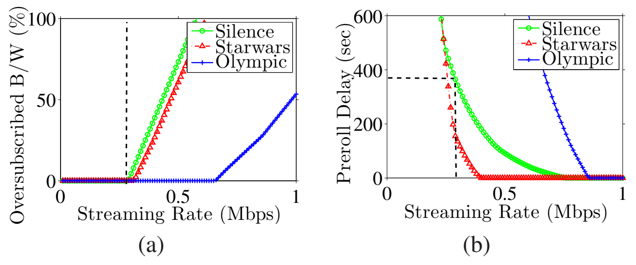


Figure 2: RVBR may result in: (a) wasted bandwidth, and (b) long preroll delay.

add *smoothing buffers* to regulate VBR streams for constant rate channels using the leaky bucket algorithm [6]. Smoothing buffers absorb VBR traffic burstiness at the expense of higher memory requirement and longer preroll delay. *Preroll delay* is the minimal buffering time before mobile devices can start rendering video streams without risking for playout glitches. The preroll delay, unfortunately, could be prohibitively long because of the smoothing buffer. To illustrate, we analyze three video streams from [28], and we use the algorithm in [26] to compute the minimum preroll delay and the oversubscribed bandwidth at various r_s. Oversubscribed bandwidth refers to the fraction of the unused bandwidth. We plot the results in Fig. 2. This figure shows that there exists a tradeoff between bandwidth utilization and preroll delay. For example, streaming *Silence of the Lambs* at a bit rate lower than 280 kbps leads to no wasted bandwidth, but it results in more than *6 minutes* preroll delay, which is clearly not acceptable to users. To better quantify this tradeoff, we define a burst scheduling algorithm RVBR$_\beta$ as streaming each video stream s using a smoothing buffer, so that the regulated VBR stream has bit rate r_s that is the smallest bit rate with $d_s(r_s) \leq \beta$ sec, where $d_s(r_s)$ represents the minimum preroll delay under streaming rate r_s.

VBR$_\alpha$ and RVBR$_\beta$ are *parameterized* algorithms that can be used in current broadcast networks. In Sec. 5, we compare these two algorithms against our proposed algorithm.

3. PROBLEM FORMULATION

3.1 Problem Statement and Hardness

We study wireless networks in which a base station transmits S video streams to many mobile devices over a shared air medium with bandwidth R kbps. We consider a broadcast time of T sec, in which each video stream has I frames, and is coded at F fps (frame-per-second). Therefore, we have $T = I/F$. We consider very general VBR streams: each frame i ($1 \leq i \leq I$) of video stream s has a size of l_i^s kb, which is flexible except that its instantaneous bit rate should be smaller than the air medium bandwidth, i.e., we assume $l_i^s F < R$. To guarantee smooth playouts, every frame i must arrive at mobile devices no later than its decoding deadline i/F sec. The base station transmits every video stream in bursts at bit rate R kbps. Therefore, once a burst of data is received, mobile devices put the wireless interfaces into sleep till the next burst in order to save energy.

We define two performance metrics for video streaming over wireless networks: energy saving and bandwidth utilization, from subscribers' and network operators' point of view, respectively. For subscribers, we define energy saving as the ratio of time that mobile devices can put their network interfaces into sleep to the total time, and we write the energy saving of video stream s as γ_s. We define the system-wide energy saving as $\gamma = \left(\sum_{s=1}^{S} \gamma_s\right)/S$. We note that similar definition of energy saving has been used in wireless broadcast networks [9, 29]. For network operators, we define the bandwidth utilization σ as the fraction of the on-time transmitted data amount, which is the aggregate size of on-time bursts, over the maximum data amount offered by the air medium, which is TR kb. This definition only considers the video data transmitted *before* their decoding deadlines, as late video data do *not* improve video quality. For example, a naive schedule that saturates the maximum data amount TR, but ignores the deadlines may result in many late frames, which are essentially useless. With these definitions, we state the problem of statistically multiplexing VBR video streams over a wireless network to mobile devices as follows.

PROBLEM 1. *Consider S VBR coded video streams to be concurrently transmitted by a base station to multiple mobile devices. Each video stream is sent as bursts of data to save energy on mobile devices. Find the optimal burst schedule for all video streams to maximize the bandwidth utilization σ* and *the energy saving γ.*

In this problem, the bandwidth utilization is the *primary objective*. Wireless spectrum is precious and concurrently streaming more video streams leads to higher revenues for network operators. The energy consumption is the *secondary objective*. Mobile devices are energy-limited and higher energy saving results in longer watch time, thus higher subscriber satisfaction. The schedule specifies for each burst the start time and its size for all video streams. The schedule cannot have burst collisions, which happen when two bursts have nonempty intersection in time. Furthermore, the schedule must ensure that there are no buffer overflow instances, which happen when a receiver has no space to store data during a burst.

Problem 1 is a generalization of the burst scheduling problem addressed in our previous works [13, 15], which considered *only one* objective function: maximizing energy saving for mobile devices. Yet, this single-objective function problem has been proved to be NP-Complete by reducing the task sequencing problem with arbitrary release times and deadlines to it [13]. Therefore, Problem 1 is clearly NP-Complete as well.

We note that our optimization problem is quite different from many other multi-objective scheduling problems, which are often solved by defining an overall objective function as a weighted sum of the given objective functions. Solving such multi-objective problems is tricky because the weights for objective functions are either heuristically chosen or determined by analyzing the complex tradeoffs among objective functions [22, Sec. 4.3]. More importantly, the resulting schedules are *compromised*, because they are unlikely to be optimal in terms of *either* objective function. In contrast, our problem consists of two objective functions that are *independent* of each other, which does not require us to define a weighted overall objective function. More specifically, our problem can be solved in two steps. First, we choose all burst schedules that maximize the bandwidth utilization. Then, we select the optimal burst schedule in terms of energy saving among them. Note that, the second step would not degrade the optimality achieved in the first step. In Sec. 4, we develop an efficient algorithm to solve this problem, and we prove the resulting schedule is optimal in terms of bandwidth utilization, and near-optimal in terms of energy saving.

3.2 Mathematical Formulation

We let n_s be the number of bursts scheduled for video stream s, where $1 \leq s \leq S$. We use f_k^s sec and b_k^s kb to denote the start time and burst size of burst k of video stream s, where $1 \leq k \leq n_s$.

Since the air medium has bandwidth R kbps, it takes b_k^s/R sec to transfer burst k of stream s. Notice that wireless interfaces need to be waken up earlier than the next burst time, because it takes some time to lock to the radio frequency and synchronize to the symbols before data can be demodulated. We denote the overhead duration as T_o sec. The value of T_o could be high in wireless networks, e.g., in mobile TV broadcast networks, T_o ranges from 50 to 250 msec [9, 10, 16]. As a specific example, the recent Philips mobile TV chip has an overhead duration of 150 msec [21]. Since mobile devices must turn on the wireless interfaces T_o sec earlier than the burst, the wireless interfaces stay on between $[f_k^s - T_o,\ f_k^s + b_k^s/R)$ in order to receive burst k of stream s. Last, we write the receiver buffer size as Q kb. Given these notations, we can define c_k^s kb as the buffer level of mobile devices at the beginning of burst k of video stream s. Mathematically, c_k^s is written as: $c_k^s = \max\left(0,\ \sum_{j=1}^{k-1} b_j^s - \sum_{i=1}^{h} l_i^s\right)$, where h is the maximum positive integer such that $h/F \le f_k^s$. This equation computes the volume difference between the received data (the first summation) and the consumed data (the second summation), and returns 0 if there is no received data in the buffer. Finally, we write a schedule $\mathbf{L}$ as a set of bursts: $\{<f_k^s,\ b_k^s> \mid 1 \le s \le S \text{ and } 1 \le k \le n_s\}$ for all video streams.

The burst scheduling problem for VBR streams can be formulated as:

$$Pri: \max_{\mathbf{L}} \qquad \sigma = \frac{\sum_{s=1}^{S}\sum_{j=1}^{n_s} b_j^s/R}{I/F}; \tag{1a}$$

$$Sec: \max_{\mathbf{L}} \quad \gamma = 1 - \frac{\sum_{s=1}^{S}\sum_{k=1}^{n_s}(T_o + b_k^s/R)}{I/F}\Big/S; \tag{1b}$$

$$\text{s.t.} \quad \left[f_k^s, f_k^s + \tfrac{b_k^s}{R}\right) \bigcap \left[f_{\bar{k}}^{\bar{s}}, f_{\bar{k}}^{\bar{s}} + \tfrac{b_{\bar{k}}^{\bar{s}}}{R}\right) = \varnothing; \tag{1c}$$

$$c_k^s + b_k^s - \textstyle\sum_{f_k^s \le\, j/F <\, f_k^s + b_k^s/R} l_j^s \le Q; \tag{1d}$$

$$\forall\, 1 \le s \ne \bar{s} \le S,\ 1 \le k \le n_s,\ 1 \le \bar{k} \le n_{\bar{s}}.$$

In this formulation, the primary goal is to maximize the bandwidth utilization σ, which is the fraction of the on-time transmitted data amount, $\sum_{s=1}^{S}\sum_{j=1}^{n_s} b_j^s$, over the maximum data amount, $RT = RI/F$. The secondary goal is to maximize the energy saving γ, which is the fraction of time that mobile devices can put their wireless interfaces into sleep over the total time. Consider stream s, the aggregate interface on-time is $\sum_{s=1}^{n_s}(T_o + b_k^s/R)$ sec, and the video length is I/F sec. Therefore, the energy saving of stream s can be computed by $1 - \frac{\sum_{s=1}^{n_s}(T_o + b_k^s/R)}{I/F}$. Computing the average energy saving γ among all video streams gives the system-wide energy saving. The constraints in Eqs. (1c) and (1d) guarantee that the resulting burst schedule is feasible. In particular, Eq. (1c) ensures that there are no burst intersections among all S video streams. Eq. (1d) validates the buffer level for stream s at the end time of every burst to prevent buffer overflow instances, where the third term (summation) includes all frames that have deadlines during that burst. It is sufficient to check the buffer level only at the end time, because the buffer level of mobile devices increases if and only if there is a burst at that moment.

4. PROBLEM SOLUTION

As described in Sec. 3, the problem of broadcasting VBR streams to maximize *both* the bandwidth utilization *and* energy saving for mobile devices is NP-Complete. Thus, solving it optimally along both objective functions is computationally expensive and cannot be done in real time, which is a requirement for video broadcasting systems in which the broadcast content is continuously changing. We propose, in Sec. 4.1, an approximation algorithm to solve this problem. In Sec. 4.2, we show that our algorithm achieves optimality along one objective function (bandwidth utilization) and near-optimality along the other objective function (energy saving). In addition, the algorithm is computationally very efficient and runs in real time (as verified by actual implementation in a real mobile TV testbed, described in Sec. 6).

4.1 Algorithm for Broadcasting VBR Streams

The high-level idea of our algorithms is as follows. We mathematically transform our problem to another scheduling problem for which we design an efficient approximation algorithm. We then transform the solution found by the approximation algorithm to a solution for the original problem. We analytically bound the approximation gap and prove the correctness of our algorithm.

Our transformation idea produces a simpler scheduling problem with only one constraint: no burst collision, and it gets rid of the other constraint: buffer overflow instances. This is achieved by using two separate buffers, say B and B', so that B can be drained when B' is filled up, and B' can be drained when B is filled up. More specifically, we propose to split the receiver buffer Q into two equal-sized buffers, and divide the sending time of video stream s into p_s disjoint time windows. We design a scheduling algorithm to properly send all the S video streams, so that mobile devices of any video stream s in window p, where $2 \le p \le p_s$, render the video data that *have been* received in window $p-1$, and thus are free from buffer overflow instances. That is, mobile devices use a buffer for receiving (filling up) data and another buffer for decoding (draining) data in every time window p, and they swap these two buffers upon reaching a new time window $p+1$. We notice that windows of the same video stream have different lengths in time due to the VBR nature of video streams, and window boundaries of different video streams are not aligned either.

Following are some details on our algorithm. To perform the transform, we first need to decide how many frames can be sent in each window p without resulting in buffer overflow on mobile devices. For any video streams s and any window p ($1 \le p \le p_s$), we let m_p^s be the last frame (with the largest frame index) that gets included in window p. Since the receiving buffer size is $Q/2$ kb in all windows, for any stream s, we can write m_p^s by induction as:

$$\begin{cases} m_p^s = 0, & p = 0; \\ \displaystyle\sum_{j=m_{p-1}^s+1}^{m_p^s} l_j^s \le \frac{Q}{2} < \sum_{j=m_{p-1}^s+1}^{m_p^s+1} l_j^s\ , & \forall\, 1 \le p \le p_s. \end{cases} \tag{2}$$

This induction stops once $m_{\hat{p}}^s = I$ for some integer $\hat{p}$. Upon m_p^s is determined, we know that frames $[m_{p-1}^s + 1, m_p^s]$ are the maximum number of frames that can be fit in the receiving buffer of window p, for $1 \le s \le S$ and $1 \le p \le p_s$. Letting y_p^s be the aggregate data amount that must be received in window p, we can write y_p^s as:

$$y_p^s = \sum_{j=m_{p-1}^s+1}^{m_p^s} l_j^s. \tag{3}$$

Furthermore, observe that mobile devices in window p always render the data received in window $p-1$. This means that the time length of window p depends on the number of frames received in window $p-1$, e.g., if 5 frames are received in the previous window, the playout time of the current window is $5/F$ sec, where F is the frame rate. Let x_p^s and z_p^s be the start and end times of window p

for video stream s. Then, we can write x_p^s and z_p^s as:

$$x_p^s = \begin{cases} 0, & p = 1; \\ (m_{p-2}^s + 1)/F, & 2 \le p \le p_s; \end{cases} \tag{4}$$

$$z_p^s = \begin{cases} \sum_{s=1}^S y_1^s/R & p = 1; \\ m_{p-1}^s/F, & 2 \le p \le p_s. \end{cases} \tag{5}$$

We mention that the windows are defined in a very dynamic way: video streams with higher instantaneous bit rates get shorter windows, while others get longer windows. This allows our algorithm to quickly adapt to the rate variations in VBR video streams. Notice that in the first window ($p = 1$) of all video streams, mobile devices have no data to playout and only receive and buffer data. Therefore, any window length could be assigned to the first window. To maximize the bandwidth utilization and minimize the preroll delay, we let the first window size be $\sum_{s=1}^S y_1^s/R$, which is the shortest possible window length to send data in the first window of all video streams. Since $y_1^s \le Q/2$ (indicated by Eqs. (2) and (3)), the preroll delay incurred by the SMS algorithm is bounded by

$$d = (SQ)/(2R). \tag{6}$$

Using these notations, we can formally write the transformed scheduling problem as:

$$Pri : \max_{\mathbf{L}} \quad \sum_{s=1}^S \sum_{j=1}^{n_s} b_j^s; \tag{7a}$$

$$Sec : \min_{\mathbf{L}} \quad \sum_{s=1}^S n_s; \tag{7b}$$

$$\text{s.t.} \quad y_p^s \ge \sum_{\forall\, x_p^s \le f_k^s < z_p^s} b_k^s; \tag{7c}$$

$$\forall\, 1 \le s \le S,\ 1 \le p \le p_s.$$

This formulation has two objective functions. It first maximizes the bandwidth utilization by maximizing the amount of on-time delivered video data in Eq. (7a). It then maximizes the energy saving by minimizing the number of bursts in Eq. (7b), as each burst incurs a constant overhead duration. The constraint in Eq. (7c) ensures that the aggregate size of scheduled bursts in every window never exceeds the aggregate size of frames associated with that window, which avoids buffer overflow instances.

To solve the transformed problem, we first define *decision points* as the time instances at which either: (i) a new window starts, i.e., at time x_p^s, (ii) a window exceeds its decoding deadline, i.e., at time z_p^s, or (iii) bursts scheduled to a window have met the required aggregate data amount y_p^s. At each decision point t, we schedule a burst for the window with the smallest end time z_p^s among all outstanding windows p' with start time $x_{p'}^s$ earlier than current time t and end time $z_{p'}^s$ later than current time t. We use outstanding window to refer to a window that needs more bursts: its accumulated data amount has not met the required amount y_p^s. Note that windows p' with $x_{p'}^s > t$ are not considered, because these windows have not started and the video data may not be available yet. Moreover, windows p' with $z_{p'}^s < t$ are not considered either, because these windows are already *late*, and late frames are essentially useless for streaming videos. The scheduling algorithm builds a schedule with a moving current time t and stops if there exist no outstanding windows, nor windows with start times in the future. Last, we define the completion time of window p of stream s as the time that window achieves the required data amount y_p^s.

We call this algorithm Statistical Multiplexing Scheduling (SMS) algorithm, and give its high-level pseudocode in Fig. 3. This algorithm constructs the first window for each video stream in lines 2–4.

The SMS Algorithm

```
1.  // Input: multiple VBR streams.
1.  // Output: burst transmission schedule for all bursts.
2.  // initial transform
3.  for $s$ = 1 to $S$
4.     generate the first window for $s$ and determine $x_1^s$, $y_1^s$,
4.     and $z_1^s$ using Eqs. (2)—(5)
5.  // burst scheduling
6.  foreach decision point of window $p$ for stream $s$ {
7.     schedule a burst from times $t$ to $t_n$ for $s$, where
7.     the window $p$ of $s$ has the smallest $z_p^s$ among all
7.     windows $p'$ with $x_{p'}^s \le t$ and $z_{p'}^s > t$, and $t$ is the
7.     current time, $t_n$ is the time of the next decision point
8.     if the window $p$ of $s$ completes or is late {
9.        generate a new window $p$ for $s$ and determine $x_p^s$,
9.        $y_p^s$, and $z_p^s$ using Eqs. (2)—(5)
10.    }
11. }
```

Figure 3: An efficient burst scheduling algorithm.

It uses the for-loop between lines 6 and 11 to traverse through all decision points in ascending order of time. It schedules a new burst in line 7 to video stream s, and then checks whether the window of s is complete or late in lines 8–10. New window is generated in line 9 if the current window either completes or is late. The algorithm stops when no more decision points exist.

We note that the SMS algorithm considers a window p for each stream s at any moment, and only advances to window $p + 1$ if window p completes or is late (lines 8–10). Thus, it only requires a small look-ahead window (in the order of a few seconds) for frame size l_i^s, and is an *online scheduling algorithm*. In addition, the SMS algorithm can handle the dynamic nature of video service. For example, to transition from a video stream to a new one, the SMS algorithm simply discards the current window and generates a new window for the new video stream, and continues to schedule bursts with no interruptions nor running-time penalty. Finally, the SMS algorithm can work with any VBR streams, and imposes no limitations on the video coders for rate control. Hence, it allows video coders to encode video streams with the maximum coding efficiency, and thus achieve the maximum perceived quality.

4.2 Analysis of the SMS Algorithm

We first prove that our algorithm produces feasible burst schedules. We then prove that the resulting schedule is optimal in terms of bandwidth utilization. We show that the resulting schedule is near optimal in terms of energy saving, and we give its approximation gap. Last, we derive its time complexity.

THEOREM 1. *The SMS algorithm gives a feasible burst schedule for the original burst scheduling problem (Problem 1).*

PROOF. The for-loop in lines 6–11 produces a schedule that has no burst collisions. This is because we assign every time interval $[t, t_n)$ to a single stream s in line 7, and we immediately advance t. Moreover, line 8 guarantees that Eq. (7c) holds, because it stops assigning bursts to p if p is complete. Hence, the SMS algorithm finds a feasible schedule for the transformed problem. Furthermore, we divide the receiver's buffer into two halves and we make sure that the aggregate data received in each window never exceeds half of

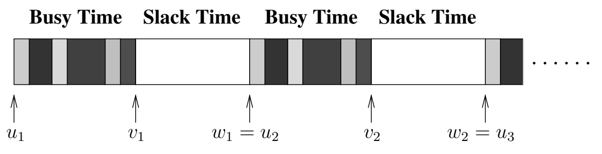
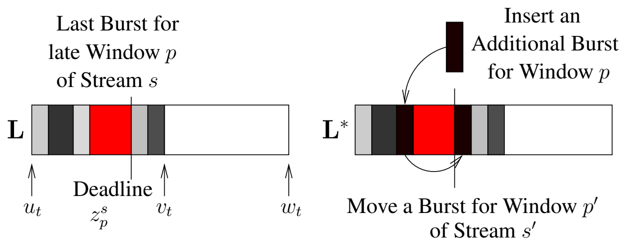


Figure 4: The resulting schedule of the SMS algorithm consists of interleaved busy and slack time periods. Different shaded blocks represent bursts for different video streams.



Last Burst for late Window p of Stream s
L
Deadline z_p^s
u_t
v_t
w_t
Insert an Additional Burst for Window p
L*
Move a Burst for Window p' of Stream s'

Figure 5: Inserting a burst requires moving another burst, as there is no gap between bursts in busy time periods.

the receiver's buffer (see Eq. (2)). Thus, the resulting schedule leads to no buffer overflow instances in the original problem. □

THEOREM 2. *The SMS algorithm returns optimal burst schedules in terms of bandwidth utilization.*

PROOF. Observe that the for-loop in lines 6–10 always schedules a burst as long as there is at least one window that is outstanding and is not late. Therefore, the resulting schedule $\mathbf{L}$ consists of interleaved *busy* time periods and *slack* time periods, as illustrated in Fig. 4. Let the t-th busy time period starts at time u_t sec and ends at time v_t sec, and the t-th slack time period starts at time v_t sec and ends at time w_t sec. During slack time periods, there is no video data to be sent: all data has been sent earlier in the corresponding busy time periods.

Next, any resulting schedule $\mathbf{L}$ falls into one of two cases. Case I: all windows complete in line 8. Case II: there is at least one window late in line 8. In case I, since all windows complete on time, the SMS algorithm meets all demands on-time. Thus, SMS is optimal in case I. For case II, we only need to show that there is no schedule better than $\mathbf{L}$. We use proof by contradiction and illustrate the argument in Fig. 5. Consider an arbitrary window p of stream s in $\mathbf{L}$, where p is not completed in busy window $[u_t, v_t)$. Assume there exists a better schedule $\mathbf{L}^*$, which allocates an *additional* d-sec burst to window p, where $d > 0$. By definition, bandwidth utilization only considers video data that arrive on-time, so this additional burst (darkened in the figure) must be inserted before z_p^s, otherwise $\mathbf{L}^*$ would not be a better schedule. Furthermore, as there is no gap among bursts in the busy time period, $\mathbf{L}^*$ must move another burst for window p' of stream s' (also darkened in the figure) to a time later than z_p^s in order to make room for the additional burst. However, line 8 says that the SMS algorithm always schedules the window with the smallest deadline, thus we know $z_{p'}^{s'} \le z_p^s$. This means that moving the burst for window p' of stream s' after time z_p^s renders it becoming a *late burst*, which cancels out the additional bandwidth utilization brought by the new burst! Therefore, the amount of on-time delivered bursts in $\mathbf{L}$ and $\mathbf{L}^*$ are the same, which contradicts the assumption. □

THEOREM 3. *The SMS algorithm produces near-optimal burst schedules in terms of energy saving with an approximation gap: $\Delta\gamma = \gamma^* - \gamma \le T_o r/Q$, where γ^* and γ are the system-wide energy savings achieved by the optimal scheduling algorithm and by the SMS algorithm, respectively, and r represents the average coding bit rate across all video streams.*

PROOF. Let n_s^* be the optimal number of bursts scheduled for video stream s. As each burst contains no more than Q kb data, we have $n_s^* \ge \sum_{i=1}^{I} l_i^s/Q$. Then, following the definition of energy saving, we write the energy saving of stream s as:

$$\gamma_s^* = 1 - \frac{\sum_{k=1}^{n_s^*}(T_o + b_k^s/R)}{I/F} = 1 - \frac{n_s^* T_o + \sum_{i=1}^{I} l_i^s/R}{I/F}$$
$$\le 1 - \frac{T_o \sum_{i=1}^{I} l_i^s/Q + \sum_{i=1}^{I} l_i^s/R}{I/F} = 1 - (\frac{T_o}{Q} + \frac{1}{R})r_s,$$

where $r_s = \sum_{i=1}^{I} l_i^s/(I/F)$ is the average coding bit rate for stream s. Following the definition of system-wide energy saving, we have:

$$\gamma^* \le 1 - (\frac{T_o}{Q} + \frac{1}{R})\sum_{s=1}^{S} r_s/S = 1 - (\frac{T_o}{Q} + \frac{1}{R})r.$$

Next, we let n_s be the number of bursts scheduled for s by the SMS algorithm. Based on Eq. (2), we let $\delta_p^s = \frac{Q}{2} - \sum_{j=m_{p-1}^s+1}^{m_p^s} l_j^s$ to represent a small portion of Q that is not fully utilized in window p. We notice that $\delta_p^s \approxeq 0$, because typical receiver buffers are much larger than frame size, e.g., media players buffer for several seconds of playout time, or hundreds of frames, before rendering videos. Since δ_p^s is insignificant, we write $p_s = \sum_{i=1}^{I} l_i^s/(Q/2)$. Then, we notice that the total number of bursts among all video streams is bounded by the number of decision points, which are defined as the time instances at which either a new window starts, completes or becomes late. Observe that, except for the boundary cases, a new window is only *created* when the previous window of the same stream completes or becomes late. This means that the number of decision points is $\sum_{s=1}^{S} p_s + S \approxeq \sum_{s=1}^{S} p_s$. Hence, we write $\sum_{s=1}^{S} n_s \le \sum_{s=1}^{S} p_s$. Then, we write the system-wide energy saving:

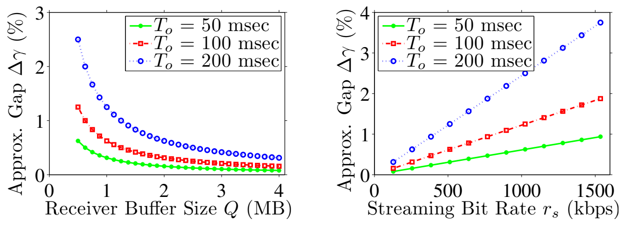
$$\gamma = 1 - \sum_{s=1}^{S} \frac{n_s T_o + \sum_{i=1}^{I} i_i^s/R}{SI/F} = 1 - \frac{T_o \sum_{s=1}^{S} n_s}{SI/F} - \frac{\sum_{s=1}^{S} r_s}{RS}.$$

Since $\sum_{s=1}^{S} n_s \le \sum_{s=1}^{S} p_s = 2\sum_{s=1}^{S}\sum_{i=1}^{I} l_i^s/Q$, we have: $\gamma \ge 1 - (\frac{2T_o}{Q} + \frac{1}{R})r$. Combining γ and γ^* yields the theorem. □

THEOREM 4. *The SMS algorithm runs in time $O(PS + S^2)$, where S is the number of video streams, and P is the maximum number of windows among all video streams.*

Proof: Since there are $\sum_{s=1}^{S} p_s + S$ decision points, and we check S windows at each decision point, the complexity of line 7 is $O(PS + S^2)$, where $P = \sum_{s=1}^{S} p_s$. Moreover, constructing windows in lines 4 and 9 takes time $O(\sum_{s=1}^{S} I)$ in total, which can be written as $O(PS)$ as the receiver buffer size Q and number of frames in each window are small constants. Thus, the SMS algorithm runs in time $O(PS + S^2) + O(PS) = O(PS + S^2)$. □

The above theorems show that the SMS algorithm produces burst schedules that are optimal in terms of bandwidth utilization, and near-optimal in terms of energy saving. Moreover, the approximation gap of energy saving given in Theorem 3 has a few desirable properties. First, the gap decreases when the overhead duration



(a) (b)

Figure 6: The proposed algorithm leads to small approximation gap with typical parameters: (a) average coding bit rate is 512 kbps, and (b) receiver buffer is 1 MB.

T_o decreases, which is expected as the hardware technology advances. Second, the gap decreases when the receiver buffer size Q increases. The receiver buffer gets larger whenever the unit price of memory chips reduces, which has been a trend for several years. Last, the gap decreases when the average coding bit rate r reduces, which is likely to happen as newer coding standards always achieve higher coding efficiency, and thus lower coding bit rates. These properties show that the SMS algorithm will even perform better as the technology advances.

To illustrate the energy saving performance of the SMS algorithm under current technology, we numerically analyze its approximation gap using a range of practical parameters. We consider overhead duration from 50 to 200 msec, receiver buffer size from 256 KB to 4 MB, and coding bit rate from 128 to 1536 kbps. We plot the numerical results in Fig. 6. Fig. 6(a) shows that the gap becomes very small if the receiver has a reasonable buffer size, e.g., the gap is less than 1.5% if receiver buffer is larger than 1 MB. Fig. 6(b) illustrates that the gap becomes smaller when coding bit rate is smaller, e.g., the gap is less than 1.25% for coding bit rate is 512 kbps and below. Notice that 512 kbps is high enough for video streaming to mobile devices, because these devices have small display resolutions. These two figures confirm that the SMS algorithm achieves a very small approximation gap on energy saving with current technology.

Last, we comment on the preroll delay incurred by the SMS algorithm, which is bounded by $(SQ)/(2R)$ as shown in Eq. (6). For illustration, we employ common network parameters, where the air medium bandwidth $R = 10$ Mbps, receiver buffer size $Q = 2$ Mb, and stream coding rate 512 kbps. We first consider a service provider who broadcasts 5 video streams, its preroll delay is less than 500 msec which is negligible. For a service provider who *saturates* the bandwidth and broadcasts 20 video streams, the preroll delay is no more than 2 sec.

5. EVALUATION USING SIMULATION

5.1 Simulation Setup

We have implemented a trace-driven simulator for broadcasting video streams in wireless networks. The simulator takes trace files of *real* VBR coded streams as inputs and considers practical network parameters. We have designed a clean interface for the simulator to facilitate various burst scheduling algorithms, and we have implemented the proposed SMS algorithm in the simulator. We have also implemented the current VBR$_\alpha$ and RVBR$_\beta$ algorithms (which are described in Sec. 2) for comparison. As discussed in Sec. 2, we are not aware of other burst scheduling algorithms in the literature that broadcast VBR streams in bursts. This, however, is not a major concern, as we *analytically prove* that our algorithm achieves optimal bandwidth utilization and almost-optimal energy saving. Furthermore, in some of our experiments, we compare our algorithm against an *upper bound* on the energy saving that can be achieved by *any algorithm*.

For the network parameters, we choose to use 16-QAM modulation scheme, 5/6 channel coding rate, 1/8 guard interval, and 5 MHz channel bandwidth. This gives us a broadcast network with bandwidth $R = 17.2$ Mbps [9]. We consider an overhead duration $T_o = 100$ msec and receiver buffer size $Q = 4$ Mb ($= 0.5$ MB). To saturate network bandwidth, we concurrently broadcast up to 20 VBR video streams, where each stream has different characteristics. We downloaded 20 trace files from [28]. These trace files are for CIF video streams coded by H.264/AVC coders at 30 fps. We follow the recommendations given in [27] to generate a realistic video traffic workload from these traces in two steps. First, we construct a 60-min trace by starting from a random time and wrapping around if the end of the original coded stream is reached. Second, we scale the frame sizes of each video stream so that it has a random average bit rate between 100 to 1250 kbps. These two steps generate a set of video trace files with diverse and varying video characteristics to mimic the video streams broadcast in real networks.

To cover all possible burst schedules that can be used in current base stations, we vary the α value of the VBR$_\alpha$ algorithm from 48% to 98% and we vary the β value of the RVBR$_\beta$ algorithm from 1 to 64 sec. If not otherwise specified, we concurrently broadcast 20 video streams for 60 minutes using each of the considered burst scheduling algorithms, and we compute three performance metrics: missed frames, bandwidth utilization, and energy saving. The missed frames include video frames that cannot be broadcast due to shortage of bandwidth reserved to video streams, and frames that are late and cannot be decoded. We define the missed frame ratio as the number of missed frames to the number of total frames, which is an important QoS metric because higher missed frame ratios result in more playout glitches that are annoying to users.

5.2 Simulation Results

Missed Frames: We compute the mean and maximal missed frame ratios of all video streams for each considered algorithm. We report the results in Fig. 7(a), which shows that the SMS algorithm produces almost no missed frames, while VBR$_{70\%}$ results in up to 33% missed frame ratio and RVBR$_1$ leads to up to 12% missed frame ratio. Clearly, the current scheduling algorithms lead to unacceptable QoS: a playout glitch every 1 and 3 secs for VBR$_{70\%}$ and RVBR$_1$, respectively. This experiment shows that the SMS algorithm results in much better perceived quality than the current scheduling algorithms.

Next, we vary the α and β values and compute the missed frame ratio for each of them. Our SMS algorithm is not shown in the figures as it does not depend on α and β, and as indicated by Fig. 7(a) it produces almost no missed frames. We plot the results of VBR$_\alpha$ algorithm with different α values in Fig. 7(b). This figure reveals that changing the α value does not solve the QoS issue at all: at least 4% of missed frame ratio is observed no matter what α value is used. This means that even if network operators *exhaustively* try all possible α values with the current VBR$_\alpha$ algorithm, no burst schedule with acceptable QoS is possible. Then, we plot results of the RVBR$_\beta$ algorithm with various β values in Fig. 7(c). This figure shows that the average missed frame ratio decreases when the preroll delay of the RVBR$_\beta$ algorithm increases. However, we observe that a preroll delay of 48 sec is required for a zero average missed frame ratio. Unfortunately, a 48-sec preroll delay significantly de-

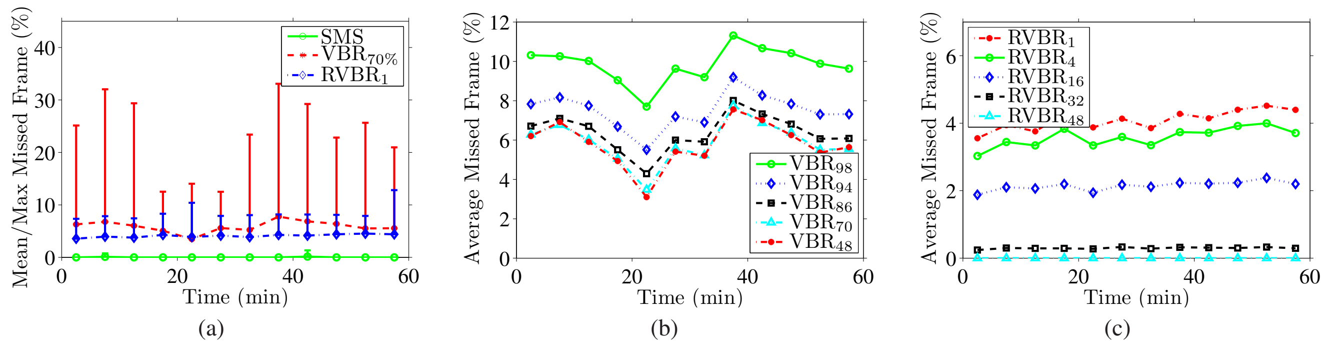


Figure 7: Missed frame ratio produced by: (a) all considered algorithms, (b) the VBR$_\alpha$ algorithm with various α values, and (c) the RVBR$_\beta$ algorithm with various β values.

grades user experience, and thus is not acceptable for mobile video services. Therefore, the current RVBR$_\beta$ algorithm can not achieve acceptable QoS either. This experiment confirms that the current scheduling algorithms can only achieve inferior perceived quality than the proposed SMS algorithm.

Bandwidth Utilization: We next study how many video streams can the current burst scheduling algorithms concurrently broadcast for a given QoS target. More precisely, we set the target mean missed frame ratio to be 0.5% in this experiment and we try to achieve this target using different scheduling algorithms. We start by broadcasting 20 video streams using the SMS, the VBR$_{70\%}$ and the RVBR$_1$ algorithms for 60 minutes. For each algorithm, we compute the average missed frame ratio over the whole broadcast period. If the average missed frame ratio is more than 0.5%, we reduce the number of concurrently broadcast video streams by one and repeat the 60-min broadcast, until we achieve the target missed frame ratio. We note that, at each iteration, we drop the video stream with the smallest bit rate. The rationale is that video streams with lower bit rates may be less important, and dropping them earlier may allow us to achieve higher bandwidth utilization. We plot the average missed frame ratio in Fig. 8(a). This figure shows that while the SMS algorithm can concurrently broadcast 20 video streams, the RVBR$_1$ algorithm can only broadcast 14 video streams and the VBR$_{70\%}$ algorithm can only broadcast 2 video streams. In Fig. 8(b), we plot the maximum number of video streams that can be concurrently broadcast by each scheduling algorithm. This figure shows that no matter what α value is used in the VBR$_\alpha$ algorithm, it can only broadcast 2 video streams. Moreover, a β value larger than 16 is required for the RVBR$_\beta$ to achieve the same number of video streams as the SMS algorithm, which significantly degrades user experience due to its excessive preroll delay of 32 sec. This experiment shows a great advantage of the SMS algorithm: it allows network operators to broadcast many more video streams under the same QoS requirements, which leads to higher revenues.

Near-optimality on Energy Saving: We next compare the energy saving achieved by the SMS algorithm against the current burst scheduling algorithms. We also compare against a very conservative *upper bound* on the maximum achievable energy saving. We use this upper bound because the burst scheduling problem is NP-Complete, and computing the exact optimal solutions may take long time. We compute the upper bound as follows. For each video stream, we broadcast only this stream without any other streams for 60 minutes. The resulting schedule achieves maximum energy saving by allocating the largest possible bursts that can fit in receiver's buffer. The network interfaces of mobile devices are put into sleep after getting a burst until that burst is completely consumed. Clearly, the schedule leads to a conservative upper bound on the energy saving, and we denote this upper bound as UB in the figure. We repeat this experiment for 20 times: once for every video stream. Then, we run the SMS and the current burst scheduling algorithms to compute the burst schedules for all 20 video streams concurrently. Sample energy saving achieved by different burst scheduling algorithms are reported in Fig. 9; results for other video streams are similar. We draw two observations out of this figure. First, the SMS algorithm achieves near-optimal energy saving: as close as 2% lower than the *conservative* upper bound, and up to 7%. Second, the SMS algorithm achieves higher energy saving than the current VBR$_{98\%}$ and RVBR$_{16}$ with a margin as high as 12% and 5%, respectively. This experiment shows that the proposed SMS algorithm achieves energy saving that is very close to the optimal, and is better than that of the current scheduling algorithms.

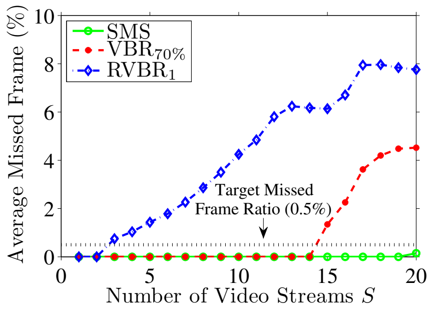
6. EVALUATION IN REAL TESTBED

6.1 Testbed Setup

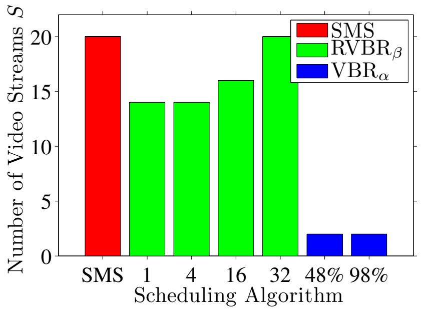
In this section, we evaluate the proposed algorithm in a real mobile TV network that complies with the DVB-H standard [10, 16], which is an open international standard for wireless broadcast networks.

We have implemented the proposed SMS algorithm in a complete testbed for mobile TV networks as a proof of concept. We have set up this testbed in our Lab, and it consists of two parts: a base station and several receivers. We use a commodity Linux PC as the base station, and install a PCI modulator card [7] in it. This modulator implements the physical layer of the DVB-H standard and is connected to an indoor antenna via a low-power amplifier. In order to drive the modulator to transmit DVB-H compliant signals, we have designed and implemented a software package for the base station. In addition, we have implemented the SMS algorithm in the base station. We use Nokia N92 and N96 cellular phones as receivers, which allow us to assess the visual quality of video streams. To gather and analyze the low-level signals, we use a DVB-H Analyzer [8]. More details on our testbed can be found in [14].

For the experiments, we configured the modulator to use an 8 MHz radio channel with QPSK (Quadrature Phase-Shift Keying) modulation scheme. According to the DVB-H standard documents, this leads to 8.289 Mbps shared air medium bandwidth [9]. We set the overhead duration $T_o = 100$ msec, and the receiver buffer size $Q = 4$ Mb. To form a realistic set of video streams, we use



(a)



(b)

Figure 8: (a) Missed frame ratio achieved by various scheduling algorithms with different number of video streams. (b) Maximum number of video streams that can be broadcast.

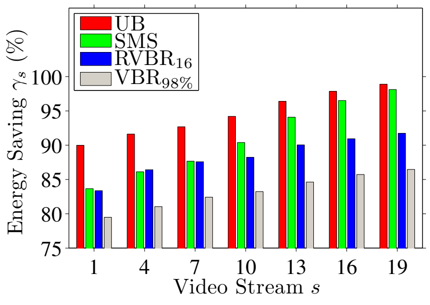


Figure 9: Energy saving achieved by considered burst scheduling algorithms and a conservative upper bound.

five production-quality video sequences provided by the Canadian Broadcasting Corporation, which is the largest content provider and broadcaster in Canada. These video sequences include documentary, talk show, soap opera, TV game show, and sports event. Thus, the test sequences have quite diverse video characteristics. Each sequence lasts for 5 minutes. We encode each video sequence into two H.264/AVC coded VBR streams, with average bit rates of 250 and 768 kbps, respectively. That is, we get 10 coded streams in total. We also encode the audio at 96 kbps using an MPEG-4 AAC encoder. We then multiplex the video and audio tracks into mp4 files, which are supported by the streaming server implemented in our testbed. We concurrently broadcast 20 video streams (each mp4 file is broadcast over two channels) using the SMS algorithm for three minutes, and we collect detailed logs at the base station. The logs contain the start and end times (in microsecond) of every burst of data and its size. We developed several software utilities to analyze the logs for three performance metrics: cumulative received bits, time spacing between successive bursts, and energy saving.

6.2 Results from Mobile TV Testbed

Correctness of the SMS Algorithm: We first validate the correctness of the SMS algorithm, i.e., it produces burst schedules that adapt to bit rate variation in VBR streams, and results in no burst conflicts. To study the bit rate adaptation, we compute the cumulative received bits (from the broadcasting base station) as the time progresses. We observe that the SMS algorithm adapts to the bit rate variations quite well, and it allocates dynamic inter-burst time to each video stream. Dynamic inter-burst time allows the SMS algorithm to send bursts as long as possible for higher energy saving.

Next, we compute the time spacing between all bursts to validate the nonexistence of burst conflicts. We first sort bursts of all video streams based on their start times. Then, we sequentially compute the time spacing between the start time of a burst and the end time of its immediate, previous, burst. The results confirm that there are no conflicts among the resulting bursts.

Energy Saving of the SMS Algorithm: We report the energy saving achieved by receivers of different video streams when the SMS algorithm is used. Fig. 10 shows the energy saving of four representative video streams; the energy saving of other streams are not shown for the clarity of the figure. We observe that the energy saving for low bit rate video streams (250 kbps) can be as high as 96%, while it is at least 80% for high bit rate video streams (768 kbps). This figure shows that the SMS algorithm achieves fairly high energy saving in a real testbed.

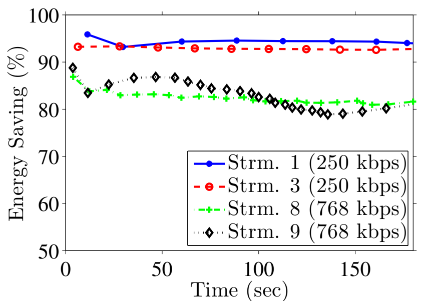


Figure 10: Energy saving achieved by our algorithm for individual video streams.

Running Time of the SMS Algorithm: In all of the above experiments, the SMS algorithm was running in *real time* on a commodity PC. The running time of scheduling bursts for the whole experiment (3 minute period) was in the order of tens of milliseconds. Note that, in our testbed, the same PC also runs several video streaming servers and modulation software as background threads. These threads impose realistic loads on the PC, and confirm that the proposed algorithm is practicable and efficient.

7. CONCLUSIONS AND FUTURE WORK

We studied the problem of broadcasting multiple VBR streams over a metropolitan wireless network to mobile devices. These streams are broadcast in bursts to enable mobile devices to save energy by frequently putting their network interfaces into sleep. We defined and formulated a burst scheduling problem that adopts: (i) bandwidth utilization as the primary objective function, and (ii) energy saving as the secondary objective function. We showed that this burst scheduling problem is NP-Complete. We then proposed an efficient, approximation algorithm, called Statistical Multiplexing Scheduling (SMS), to solve it. We proved that the SMS algorithm achieves optimal bandwidth utilization and it provides near-optimal energy saving. Our analysis indicates that a small energy saving gap of 1.5% from the optimal is achieved under typical network parameters. The SMS algorithm is an online scheduling algorithm, and can handle the dynamic nature of the video broadcast service. In addition, it imposes no rate constraints on the video coders encoding the VBR streams. This leads to high coding efficiency, and thus optimal visual quality.

We conducted extensive trace-driven simulations, in which we concurrently broadcast 20 VBR video streams. The simulation results reveal that the SMS algorithm outperforms the current burst scheduling algorithms, in terms of: (i) number of frames that miss the deadlines, (ii) number of concurrently broadcast video streams, and (iii) energy saving of mobile devices. In addition, our proposed approach for efficiently broadcasting VBR video streams is general and can be employed in different wireless networks. We achieve this generality by abstracting away the peculiarities of different networks in the formulation of the problem and the proposed transmission scheduling algorithm. To demonstrate the practicality of our approach, we have implemented it in a real testbed for mobile TV (DVB-H) services. We encoded different types of videos into VBR streams, where each stream consists of both video and audio tracks. We concurrently broadcast 20 streams using the testbed to mobile phones, and we collected detailed logs for performance analysis. The results from the testbed confirm that the SMS algorithm: (i) does not result in playout glitches, (ii) achieves high energy saving, and (iii) runs in real time.

The work in this paper can be extended in multiple directions. For example, in Sec. 4.1, the SMS algorithm always divides the receiver buffer by half. This is because the algorithm is designed as an online algorithm with a small look-ahead window. If larger look-ahead windows are possible, better performance could be achieved by adaptively dividing the receiver buffer based on the bit rate variations. This is one of our future works.

8. REFERENCES

[1] Private communication with Nokia's engineers managing mobile TV base stations, December 2008.

[2] AT&T sells wireless spectrum in southeast to Clearwire corporation, 2007. `http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=23428`.

[3] P. Camarda, G. Tommaso, and D. Striccoli. A smoothing algorithm for time slicing DVB-H video transmission with bandwidth constraints. In *Proc. of ACM International Mobile Multimedia Communications Conference (MobiMedia'06)*, Alghero, Italy, September 2006.

[4] M. Chari, F. Ling, A. Mantravadi, R. Krishnamoorthi, R. Vijayan, G. Walker, and R. Chandhok. FLO physical layer: An overview. *IEEE Transactions on Broadcasting*, 53(1):145–160, March 2007.

[5] S. Cho, G. Lee, B. Bae, K. Yang, C. Ahn, S. Lee, and C. Ahn. System and services of Terrestrial Digital Multimedia Broadcasting (T-DMB). *IEEE Transactions on Broadcasting*, 53(1):171–178, March 2007.

[6] P. Chou. Streaming media on demand and live broadcast. In M. van der Schaar and P. Chou, editors, *Multimedia Over IP and Wireless Networks*, chapter 14, pages 453–502. Academic Press, March 2007.

[7] Dektec DTA-110T PCI modulator, 2008. `http://www.dektec.com/Products/DTA-110T/`.

[8] Divi Catch RF-T/H transport stream analyzer, 2008. `http://www.enensys.com/`.

[9] Digital Video Broadcasting (DVB); DVB-H implementation guidelines. European Telecommunications Standards Institute (ETSI) Standard EN 102 377 Ver. 1.3.1, May 2007.

[10] G. Faria, J. Henriksson, E. Stare, and P. Talmola. DVB-H: Digital broadcast services to handheld devices. *Proceedings of the IEEE*, 94(1):194–209, January 2006.

[11] FLO technology overview, 2009. `http://www.mediaflo.com/news/pdf/tech_overview.pdf`.

[12] Global IPTV market analysis (2006–2010), 2006. `http://www.rncos.com/Report/IM063.htm`.

[13] M. Hefeeda and C. Hsu. Energy optimization in mobile TV broadcast networks. In *Proc. of IEEE Innovations in Information Technology (Innovations'08)*, pages 430–434, Al Ain, United Arab Emirates, December 2008.

[14] M. Hefeeda, C. Hsu, and Y. Liu. Testbed and experiments for mobile TV (DVB-H) networks. In *Proc. of ACM Multimedia'08 Demo Session*, Vancouver, Canada, October 2008.

[15] C. Hsu and M. Hefeeda. Time slicing in mobile TV broadcast networks with arbitrary channel bit rates. In *Proc. of IEEE INFOCOM'09*, Rio de Janeiro, Brazil, April 2009.

[16] M. Kornfeld and G. May. DVB-H and IP Datacast – broadcast to handheld devices. *IEEE Transactions on Broadcasting*, 53(1):161–170, March 2007.

[17] H. Lai, J. Lee, and L. Chen. A monotonic-decreasing rate scheduler for variable-bit-rate video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(2):221–231, February 2005.

[18] J. Lin, R. Chang, J. Ho, and F. Lai. FOS: A funnel-based approach for optimal online traffic smoothing of live video. *IEEE Transactions on Multimedia*, 8(5):996–1004, October 2006.

[19] WiMAX and wireless mesh worldwide market opportunities for Canadian companies, 2008. `http://www.ic.gc.ca/eic/site/ict-tic.nsf/vwapj/0107896e.pdf`.

[20] Nokia mobile broadcast solution, February 2009. `http://www.mobiletv.nokia.com/solutions/mbs/`.

[21] Philips SDIO TV1000/TV1100 mobile/portable TV solutions, 2006. `http://www.nxp.com/acrobat_download/other/products/rf/SDIO_TV_final.pdf`.

[22] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition, 2008.

[23] T. Rappaport. *Wireless Communications: Principles & Practice*. Prentice Hall, 1st edition, January 1996.

[24] M. Rezaei. Video streaming over DVB-H. In F. Luo, editor, *Mobile Multimedia Broadcasting Standards*, chapter 4, pages 109–131. Springer US, November 2009.

[25] M. Rezaei, I. Bouazizi, and M. Gabbouj. Joint video coding and statistical multiplexing for broadcasting over DVB-H channels. *IEEE Transactions on Multimedia*, 10(7):1455–1464, December 2008.

[26] J. Ribas-Corbera, P. Chou, and S. Regunathan. A generalized hypothetical reference decoder for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):674–687, July 2003.

[27] P. Seeling and M. Reisslein. Evaluating multimedia networking mechanisms using video traces. *IEEE Potentials*, 24(4):21–25, October/November 2005.

[28] Web Page of Video Traces Research Group, 2009. `http://trace.eas.asu.edu/h264avc/`.

[29] X. Yang, Y. Song, T. Owens, J. Cosmas, and T. Itagaki. Performance analysis of time slicing in DVB-H. In *Proc. of Joint IST Workshop on Mobile Future and Symposium on Trends in Communications (SympoTIC'04)*, pages 183–186, Bratislava, Slovakia, October 2004.