Statistical Multiplexing of Variable-Bit-Rate Videos Streamed to Mobile Devices

CHENG-HSIN HSU, Deutsche Telekom Laboratories USA
MOHAMED HEFEEDA, Simon Fraser University

We address the problem of broadcasting multiple video streams over a broadcast network to many mobile devices, so that: (i) streaming quality of mobile devices is maximized, (ii) energy consumption of mobile devices is minimized, and (iii) goodput in the network is maximized. We consider two types of broadcast networks: closed-loop networks, in which all video streams are jointly encoded to ensure their total bit rate does not exceed the broadcast network bandwidth, and open-loop networks, in which videos are encoded using standalone coders, and thus must be carefully broadcast to avoid playout glitches. We first show that the problem of optimally broadcasting multiple videos is NP-complete. We then propose an approximation algorithm to construct burst schedules for multiple VBR (Variable-Bit-Rate) streams. The proposed algorithm frees network operators from the manual and error-prone bandwidth reservation process which is currently used in practice. We prove that the proposed algorithm achieves optimal goodput and near-optimal energy saving. We show that it produces glitch-free schedules in closed-loop networks, and it minimizes number of glitches in open-loop networks. We implement the proposed algorithm in a trace-driven simulator, and conduct extensive simulations for both open- and closed-loop networks. The simulation results show that the proposed 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. To show the practicality and efficiency of the proposed algorithm, we also implement it in a real mobile TV testbed 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

Additional Key Words and Phrases: Broadcast networks, variable-bit-rate streams, mobile TV, energy saving, goodput, DVB-H

ACM Reference Format:

Hsu, C.-H. and Hefeeda, M. 2011. Statistical multiplexing of variable-bit-rate videos streamed to mobile devices. ACM Trans. Multimedia Comput. Commun. Appl. 7, 2, Article 12 (February 2011), 23 pages.
DOI = 10.1145/1925101.1925107 http://doi.acm.org/10.1145/1925101.1925107

1. INTRODUCTION

Increasingly more users watch streaming videos over wireless networks using mobile devices such as laptops, PDAs (Personal Digital Assistants), smart phones, and PMPs (Portable Media Players). When

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and in part by the British Columbia Innovation Council.
Authors' addresses: C.-H. Hsu and M. Hefeeda (corresponding author), School of Computing Science, Simon Fraser University, 250-13450 102nd Ave., Surrey, B. C., Canada V3T 0A3; email: mhefeeda@cs.sfu.ca.

DOI 10.1145/1925101.1925107 http://doi.acm.org/10.1145/1925101.1925107

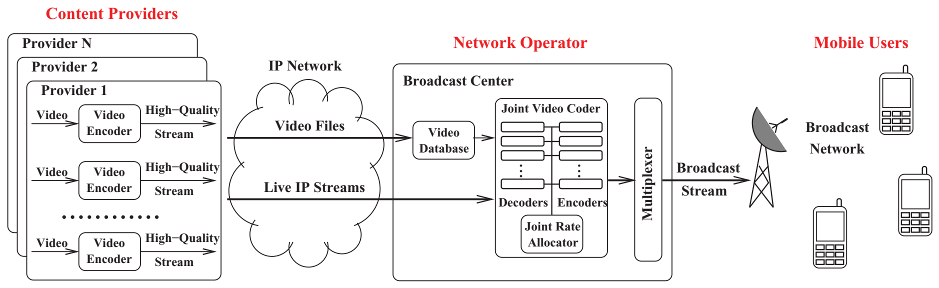


Fig. 1. Main components of a closed-loop video broadcast network.

the streaming rates are high and number of mobile devices is large, streaming videos using unicast may overload that wireless network. In contrast, streaming videos from a base station using one-to-many multicast/broadcast service supports many more mobile devices. We consider wireless networks that support multicast/broadcast, and we refer to them as *broadcast* networks. Sample broadcast networks include WiMAX networks [Wang et al. 2008], MBMS (Multimedia Broadcast Multicast Services) cellular networks [Parkvall et al. 2006], and mobile TV broadcast networks such as DVB-H (Digital Video Broadcast–Handheld) [ETSI 2004; Faria et al. 2006; Kornfeld and May 2007], MediaFLO (Forward Link Only) [Chari et al. 2007], and ATSC (Advanced Television Systems Committee) mobile DTV [ATSC Mobile DTV Standard 2009] networks.

A typical broadcast network is illustrated in Figure 1, which consists of three entities: content providers, network operators, and mobile users. Content providers are companies that create videos. Network operators are companies that manage base stations and provide services to mobile users. A network operator multiplexes several videos into a broadcast stream, and transmits it over a broadcast network with a fixed bandwidth. We consider a fixed bandwidth because feedback channels from numerous receivers are not practical in a broadcast network, and many broadcast standards, such as Kornfeld and May [2007] and Chari et al. [2007], use Forward Error Correction (FEC) and strong channel coding to combat dynamic wireless channels for reliable broadcast services in planned coverage areas. Videos are usually encoded in VBR (Variable-Bit-Rate), rather than CBR (Constant-Bit-Rate), for better video quality, shorter delay, and higher statistical multiplexing gain [Lakshman et al. 1998]. Because the broadcast network is bandwidth limited, the multiplexer must ensure that the bit rate of the broadcast stream does not exceed the network bandwidth. One way to control the bit rate is to employ *joint video coders*, which encode multiple videos and dynamically allocate available network bandwidth among them, so that the aggregate bit rate of the coded streams never exceeds the network bandwidth. As shown in Figure 1, a joint video coder consists of a joint rate allocator, several decoders, and several VBR coders. We call broadcast networks with joint video coders as *closed-loop* broadcast networks. Joint video coders may not always exist in broadcast networks, because of the high deployment cost. We call broadcast networks with standalone video coders as *open-loop* broadcast networks.

Since mobile devices are battery powered, energy consumption is critical to user experience, as higher energy consumption leads to shorter watch time. Therefore, broadcast networks must implement some energy saving techniques. One common technique is to make the base station send each video stream in bursts at a bit-rate much higher than the encoding rate of that video. This is called *time slicing* in broadcast network standards. The base station computes the next burst time and includes it in the header fields of every burst. Time slicing enables the mobile devices to receive a burst of traffic, and then put their receiving circuits into sleep until the next burst to save energy. While time slicing allows mobile devices to save energy, multiplexer must carefully compose burst schedules, which specify the start time of each burst and its size, in order to maintain good service quality.

In this article, we consider the problem of constructing burst schedules for the multiplexer in a broadcast network that transmits VBR streams in order to achieve the following three objectives.

(1) *Streaming Quality.* Video streaming is a real-time application, and multiplexers should ensure that there is no buffer violation instances on mobile devices. A buffer violation occurs when the receiver of a video stream either: (i) has no data in the buffer to play out (buffer underflow), or (ii) has no space to store the received data (buffer overflow).
(2) *Energy Consumption.* Higher energy consumption leads to shorter watch time before users replacing or recharging their batteries, and more toxic waste of primary (nonrechargeable) or rechargeable batteries. Multiplexers should reduce the energy consumption on mobile devices to increase user satisfaction and reduce pollution.
(3) *Goodput.* The wireless spectrum is expensive, for example, AT&T sold a WiMAX spectrum in the USA to Clearwire for $300 million [AT&T News 2007]. To be commercially viable, network operators must achieve high goodput in their wireless networks. The goodput refers to the fraction of the amount of video data delivered on time over the network capacity. Higher goodput in general leads to more concurrently broadcast video streams within a given spectrum, and thus higher net profits of network operators.

We formulate the problem of broadcasting multiple VBR streams as an optimization problem, and we show that it is NP-complete. We propose an approximation algorithm to solve this problem, and we show that the resulting burst schedules are optimal in terms of goodput and near-optimal in terms of energy saving. We then show that the proposed algorithm produces glitch-free schedules in closed-loop networks, and minimizes the number of glitches in open-loop networks. We evaluate the proposed algorithm using simulations and experiments. We develop a trace-driven simulator, and we implement the proposed algorithm in it. The simulation results show that the proposed algorithm outperforms the algorithms currently used in commercial base stations in both open- and closed-loop networks. For example, in our experiments, the proposed algorithm results in almost no missed/late frames in an open-loop network, while the current base stations lead to as high as 30% of missed/late frames. Finally, we use a real mobile TV testbed to show the practicality and efficiency of the proposed scheduling algorithm. The results from the testbed confirm that our proposed algorithm runs in real time, and produces feasible burst schedules that result in high energy saving. For instance, the energy saving for low bit-rate video streams (250kbps) can be as high as 96%, and it is at least 80% for high bit-rate video streams (768kbps) in our experiments.

We note that a preliminary version of this article was presented at the ACM Multimedia 2009 Conference [Hsu and Hefeeda 2009b].

The rest of this article is organized as follows. Section 2 summarizes the related work in the literature. In Section 3, we define and formulate the problem of broadcasting VBR streams. We solve the problem and analytically analyze our proposed solution in Section 4. Proofs of our theorems are given in Appendix C. We conduct extensive trace-driven simulations in Section 5, and we implement and evaluate the proposed algorithm in a real testbed in Appendix A. We conclude this article in Section 6.

2. RELATED WORK

The energy saving of mobile devices in broadcast networks that send videos in bursts has been considered in several works. For example, the authors of Yang et al. [2004] and ETSI [2007] study the energy saving achieved by a given burst schedule. Both works show that streaming videos in bursts enables mobile devices to put their receiving circuits into sleep for a significant fraction of time. These two works do not solve the burst scheduling problem. In Hefeeda and Hsu [2008, 2009], we optimally solve a simplified version of the burst scheduling problem where video streams are classified into a few classes and each class has a different bit rate. We present an efficient burst scheduling algorithm for video streams that can take any arbitrary bit rates in Hsu and Hefeeda [2009c, 2009a]. Our previous works [Hefeeda and Hsu 2008, 2009; Hsu and Hefeeda 2009c, 2009a] target CBR video streams, and do not consider the rate variability within each video stream. In the current article, we solve the burst scheduling problem for VBR video streams, which may lead to better streaming quality, shorter delay, and more concurrent broadcast streams [Lakshman et al. 1998].

Streaming VBR videos in the Internet is challenging. Several smoothing algorithms have been proposed in the literature, for example, in Lai et al. [2005] and Lin et al. [2006], 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 violations instances. These smoothing algorithms are based on the leaky bucket algorithm [Chou 2007; Ribas-Corbera et al. 2003], and they assume that packets are small. While the packet size assumption holds in the Internet, broadcast networks transmit videos in much larger bursts to save energy. Therefore, these smoothing algorithms cannot solve the problem considered in this article. Camarda et al. [2006] extend the Internet smoothing algorithms for mobile networks that transmit videos in bursts. They consider the problem of placing frames of a VBR stream into bursts of some predefined burst schedules, such that the mobile devices are free from buffer violation instances and the number of late frames is minimized. Their work is different from ours, because they consider a given burst schedule, while we compute near-optimal burst schedules. Furthermore, the smoothing algorithms in Lai et al. [2005], Lin et al. [2006], Chou [2007], Ribas-Corbera et al. [2003], and Camarda et al. [2006] only consider a single VBR stream while our problem is to concurrently broadcast multiple video streams.

Joint video coders have been well studied in the literature. For example, several works, such as Wang and Vincent [1996], Tagliasacchi et al. [2008], He and Wu [2008], Rezaei et al. [2008], Jacobs et al. [2008], propose joint coder designs for popular video coding standards. Wang and Vincent [1996] propose a joint coder for MPEG-2 coded streams. Tagliasacchi et al. [2008] and He and Wu [2008] propose joint coders for H.264/AVC coders. Rezaei et al. [2008] also propose a joint coder for H.264/AVC using fuzzy logic, which achieves low end-to-end delay and uniform quality among video streams. Jacobs et al. [2008] propose a joint coder that adjusts the bit rate of multiple scalable streams encoded using H.264/SVC coded coders. H.264/SVC coders significantly reduce the cost of adapting and transcoding video streams. These works on joint coders are quite different from ours, as they do not construct burst schedules. In this article, we solve the burst scheduling problem in broadcast systems with and without joint coders.

Rezaei et al. [2009] propose a burst scheduling algorithm for mobile TV networks. They divide the broadcast time into fixed-length scheduling windows, and then schedule all video streams in round-robin fashion in each window. To adapt to bit-rate variations, the burst length is flexible in each scheduling window, but bursts cannot span over more than two scheduling windows. They propose an empirical model to predict future frame sizes, and then compute the probable start time of the next burst. That is, their burst schedules always wake the receiving circuits up early so that mobile devices do not miss bursts. Our work is different from theirs in two aspects. First, our algorithm constructs schedules with completely flexible burst start times and lengths, that is, without the constraints of scheduling windows. This gives us opportunities to achieve better energy saving. Second, we assume that there is a small look-ahead window (a few seconds) for constructing burst schedules, which enables us to compute precise burst start times. This in turn allows us to avoid waking up receiving circuits too early, and thus our algorithm saves more energy. We note that a small look-ahead window is a reasonable assumption because many programs are prerecorded, while live streams are often delayed to allow censoring and editing. Since the work in Rezaei et al. [2009] probabilistically schedules bursts, its energy saving can never be better than the current base stations in closed-loop networks, which employ the same round-robin scheduling but use real frame sizes. Therefore, we only compare our algorithm against the current base stations.

3. PROBLEM FORMULATION

3.1 Problem Statement and Hardness

We study broadcast 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. We assume streams have instantaneous bit rates smaller than the air medium bandwidth, that is, $l_i^s F < R$. 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 receiving circuits into sleep until the next burst in order to save energy.

We define two performance metrics for video streaming over wireless networks: energy saving and goodput, from mobile users' and network operators' point of view, respectively. For users, we define energy saving as the ratio of time that mobile devices can put their receiving circuits 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 $\gamma = \left(\sum_{s=1}^{S} \gamma_s\right)/S$. Similar definition of energy saving has been used in broadcast networks [Yang et al. 2004; ETSI 2007]. For network operators, we define the goodput σ as the fraction of the ontime transmitted data amount, which is the aggregate size of ontime 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 cannot improve video quality. With these definitions, we state the problem of optimally broadcasting VBR video streams over a broadcast network to mobile devices as follows.

Problem 1. Consider S VBR-encoded 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 goodput σ *and* the energy saving γ, while achieving optimal streaming quality, that is, resulting in no playout glitches on mobile devices.

In this problem, the goodput is the *primary objective*. Higher goodput in general leads to more concurrent video streams. Since the wireless spectrum is precious, concurrently streaming more video streams leads to higher profits 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 user satisfaction. A burst schedule specifies for each burst the start time and its size for all video streams. The resulting schedule cannot have burst intersections, which happen when two bursts have nonempty intersection in time. Furthermore, the schedule must ensure that there are no buffer violation instances for any channel. A buffer violation occurs when a mobile device has either no data in the buffer to pass on to the decoder for playout (buffer underflow), or has no space to store data during a burst transmission (buffer overflow).

Problem 1 is a generalization of the burst scheduling problem addressed in our previous works [Hefeeda and Hsu 2008, 2009; Hsu and Hefeeda 2009c, 2009a], where we consider CBR video streams with only one objective function: maximizing energy saving for mobile devices. Yet this single-objective function problem has been proved to be NP-complete [Hefeeda and Hsu 2009]. Therefore, Problem 1, which considers VBR streams and two objective functions, is clearly NP-complete.

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 those multi-objective problems is tricky because the weights for objective functions are either heuristically chosen or determined by analyzing the complex trade-off among objective functions [Pinedo 2008, Section 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. In Section 4, we solve this problem, and we prove that the resulting schedule is optimal in terms of goodput, and near-optimal in terms of energy saving in closed-loop networks.

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 receiving circuits 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. This time period is referred to as overhead duration T_o sec. The value of T_o could be high in wireless networks, for example, in mobile TV broadcast networks, T_o ranges from 50 to 250 msec [Kornfeld and May 2007; ETSI 2007; Faria et al. 2006]. 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 let the receiver buffer size be 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 $h/F \leq f_k^s$. 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: $\{\langle f_k^s,\ b_k^s\rangle \mid 1 \leq s \leq S \text{ and } 1 \leq k \leq n_s\}$ for all video streams.

The burst scheduling problem for VBR streams can be formulated as

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

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

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

$$c_k^s > 0; \tag{1d}$$

$$c_k^s + b_k^s - \sum_{f_k^s \leq j/F < f_k^s + b_k^s/R} l_j^s \leq Q; \tag{1e}$$

$$\forall\ 1 \leq s \neq \bar{s} \leq S,\ 1 \leq k \leq n_s,\ 1 \leq \bar{k} \leq n_{\bar{s}}.$$

In this formulation, the primary goal is to maximize the goodput σ, which is the fraction of the ontime 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 receiving circuits into sleep over the total time. Consider stream s, the aggregate receiving circuits ontime 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)–(1e) 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) checks the buffer level for stream s at the start time of every burst to prevent buffer underflow instances. Eq. (1e) 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 burst start and end times, because the buffer level of mobile devices increases if and only if there is a burst at that moment.

4. PROBLEM SOLUTION

We propose in Section 4.1 an approximation algorithm to solve the burst scheduling problem. In Section 4.2, we show that our algorithm achieves optimality along one objective function (goodput) and near-optimality along the other objective function (energy saving). We give detailed proofs in Appendix C due to the space limitations. In Section 4.3, we study the applicability of the proposed algorithm in open-loop networks.

4.1 Scheduling Algorithm for VBR Streams

The high-level idea of our algorithm 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 intersection, and it gets rid of the other constraint: no buffer violation 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 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 have different lengths in time due to the VBR nature of video streams.

Following are some details about 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

$$m_0^s = 0; \quad \text{and} \sum_{j=m_{p-1}^s+1}^{m_p^s} l_j^s \le \frac{Q}{2} < \sum_{j=m_{p-1}^s+1}^{m_p^s+1} l_j^s \ , \ \forall\ 1 \le p \le p_s. \tag{2}$$

This induction stops once $m^s_{\hat{p}} = I$ for some integer $\hat{p}$. Upon m^s_p is determined, we know that frames $[m^s_{p-1}+1, m^s_p]$ 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^s_p be the aggregate data amount that must be received in window p, we can write y^s_p as

$$y^s_p = \sum_{j=m^s_{p-1}+1}^{m^s_p} l^s_j. \tag{3}$$

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 $p-1$, for example, 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^s_p and z^s_p be the start and end times of window p for video stream s. Then, we can write x^s_p and z^s_p as

$$x^s_1 = 0; \quad \text{and} \quad x^s_p = \left(m^s_{p-2}+1\right)/F, \quad 2 \le p \le p_s. \tag{4}$$

$$z^s_1 = \sum_{s=1}^{S} y^s_1/R; \quad \text{and} \quad z^s_p = m^s_{p-1}/F, \quad 2 \le p \le p_s. \tag{5}$$

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, and utilize the receiving buffer B (or B'). 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 goodput and minimize the delay, we let the first window size be $\sum_{s=1}^{S} y^s_1/R$, which is the shortest possible window length to send data in the first window of all video streams. Since $y^s_1 \le Q/2$ (indicated by Eqs. (2) and (3)), the 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

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

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

$$\text{s.t. } y^s_p = \sum_{\forall\, x^s_p \le f^s_k < z^s_p} b^s_k; \tag{7c}$$

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

This formulation first maximizes the goodput by maximizing the amount of ontime 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 T_o. The constraint in Eq. (7c) ensures that the aggregate size of scheduled bursts in every window equals to the aggregate size of frames associated with that window, which avoids buffer violation instances (both overflow and underflow).

To solve the transformed problem, we first define *decision points* as the time instances at which either: (i) a new window starts, that is, at time x^s_p, (ii) a window exceeds its decoding deadline, that is,

Statistical Multiplexing Scheduling (SMS) Algorithm

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

Fig. 2. An efficient burst scheduling algorithm.

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 Figure 2. This algorithm constructs the first window for each video stream in lines 3–5. It uses the for-loop between lines 7 and 11 to traverse through all decision points in ascending order of time. It schedules a new burst in line 8 to video stream s, and then checks whether the window of s is complete or late in lines 9 and 10. New window is generated in line 10 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 9–10). 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 runtime penalty. Finally, the SMS algorithm does not need joint video coders, and 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 best streaming quality.

4.2 Analysis of the SMS Algorithm in Closed-Loop Networks

We first prove that the proposed algorithm produces feasible burst schedules in closed-loop networks, which employ joint rate allocators to encode multiple videos into VBR streams so that the aggregate

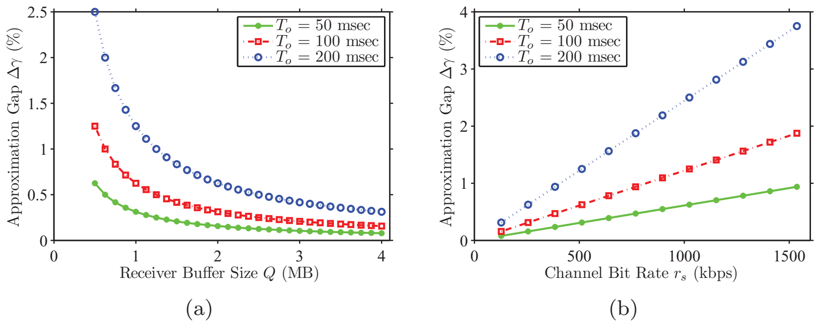


Fig. 3. The proposed algorithm leads to small approximation gap with typical parameters: (a) average coding bit rate is 512kbps, and (b) receiver buffer is 1MB.

bit rates of video streams do not exceed the bandwidth of their broadcast networks. We then prove that the resulting schedule is optimal in terms of goodput. 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 (CORRECTNESS). *The SMS algorithm returns a feasible burst schedule for the original burst scheduling problem (Problem* 1*) in closed-loop broadcast networks.*

THEOREM 2 (OPTIMALITY OF GOODPUT). *The SMS algorithm produces optimal burst schedules in terms of goodput.*

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

THEOREM 4 (TIME COMPLEXITY). *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.*

The preceding theorems show that the SMS algorithm produces burst schedules that are optimal in terms of goodput, and near-optimal in terms of energy saving. In addition, it produces glitch-free bursts in closed-loop broadcast networks. Moreover, the approximation gap of energy saving given in Theorem 3 has a few desirable properties. First, the gap decreases when the overhead duration 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 256KB to 4MB, and coding bit rate from 128 to 1536kbps. We plot the numerical results in Figure 3. Figure 3(a) shows that the gap becomes very small if the receiver has a reasonable buffer size, for example, the gap is less than 1.5% if receiver buffer is larger than 1MB. Figure 3(b) illustrates that the gap becomes smaller when coding bit rate is smaller, for example, the gap is less than 1.25% for coding bit rate is 512kbps and below. Notice that

512kbps 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 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 is 512kbps. We first consider a service provider who broadcasts five video streams, its delay is less than 500 msec which is negligible. For a service provider who saturates the bandwidth and broadcasts 20 video streams, the delay is no more than 2 sec.

4.3 The SMS Algorithm in Open-Loop Networks

Compared to open-loop broadcast networks, a closed-loop broadcast network requires additional components, such as a joint rate allocator, and several joint-coding enabled video coders. Therefore, open-loop broadcast networks are less expensive to deploy, and thus are more suitable to small-scale network operators such as local TV stations, temporary base stations, and startup broadcast companies with limited budget. The SMS algorithm proposed in Figure 2 can be used in open-loop broadcast networks. The next corollary states the sufficient condition for the SMS algorithm to construct glitch-free burst schedules in open-loop networks.

COROLLARY 1 (SUFFICIENT CONDITION). *The SMS algorithm returns glitch-free burst schedules, that is, leads to no buffer violation instances in open-loop broadcast network if the aggregate bit rate of all video streams does not exceed the broadcast network bandwidth. That is,* $\sum_{s=1}^{S} l_i^s \leq R/F$, *for all* $1 \leq i \leq I$.

This corollary is a direct result of Theorem 1, in which we use the fact that joint video coders prevent the video coders from overloading the broadcast network at all time to prove the burst schedules produced by the SMS algorithm have no buffer underflow instances. Fortunately, for small-scale network operators, not too many video streams need to be broadcast. Therefore, these network operators are unlikely to saturate the network bandwidth. Hence, these network operators may implement the SMS algorithm in the multiplexers without purchasing expensive joint video coders. When the aggregate bit rate of the video streams instantaneously exceeds the network bandwidth, the SMS algorithm will minimize the number of glitches in open-loop networks, by scheduling as much data as possible, which is proved in Theorem 2.

5. EVALUATION USING SIMULATION

In this section, we use simulations and real video traces to evaluate the proposed SMS algorithm in open-loop and closed-loop networks.

5.1 Burst Scheduling in Current Multiplexers

Many commercial multiplexers, such as UDCast [2008] and UBS [2009], implement two burst scheduling schemes: slotted and dynamic scheduling. In Appendix B, we describe how network operators use these two schemes to broadcast VBR streams, which can be summarized with three algorithms: VBR_α, RVBR_β, and DVBR_τ. In the rest of this section, we compare our SMS algorithm against these three algorithms.

5.2 Simulation Setup

We have implemented a trace-driven simulator for broadcast networks. The simulator takes trace files of *real* VBR coded streams as inputs and can simulate both open- and closed-loop networks. 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_α, RVBR_β, and DVBR_τ algorithms (which are described in Section 5.1) for comparison. We only consider these three algorithms because we are not aware of any other burst scheduling algorithm in the literature. This, however, is not a major concern, as we *analytically prove* that our algorithm achieves optimal goodput and almost-optimal energy saving. Furthermore, in some of our experiments, we compare the results of our algorithm against an *upper bound* on the energy saving that can be achieved by *any algorithm*.

We first evaluate the SMS algorithm in open-loop networks. For the network parameters, we use 16-QAM modulation scheme, 5/6 channel coding rate, 1/8 guard interval, and 5MHz channel bandwidth. This gives us a broadcast network with bandwidth $R = 17.2$Mbps [ETSI 2007]. 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 a Video Trace Library [Seeling et al. 2004]. These trace files are for CIF video streams coded by H.264/AVC coders at 30fps. We follow the recommendations given in Seeling and Reisslein [2005] 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 1250kbps. These two steps generate a set of video trace files with diverse and varying video characteristics to mimic the video streams broadcast in real open-loop networks.

To cover all possible burst schedules that can be used in current base stations, we vary the α value of the VBR_α algorithm from 48% to 98% and we vary the β value of the RVBR_β algorithm from 1 to 64 sec. If not otherwise specified, we concurrently broadcast 20 video streams for 60 min using each burst scheduling algorithm, and we compute three performance metrics: missed frames, number of concurrent video streams, 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. We define the number of concurrent video streams as the number of streams that can be broadcast by each scheduling algorithm without resulting in too many missed frames. More precisely, we choose a target missed frame ratio and we try to achieve this target using different scheduling algorithms. We start by broadcasting 20 video streams using the considered scheduling algorithms for 60 min. For each algorithm, we compute the average missed frame ratio over the whole broadcast period. If the average missed frame ratio is higher than the target ratio, 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 goodput. Finally, we consider the system-wide energy saving as a performance metric.

We then evaluate the SMS algorithm in closed-loop networks. We use the same network parameters mentioned earlier. We instruct the simulator to jointly encode 10 VBR video streams, based on the video traces from a Video Trace Library [Seeling et al. 2004]. The simulator employs Lagrangian optimization method [Sullivan and Wiegand 1998] to maximize the average video quality under the bandwidth constraint. It then concurrently broadcasts these coded streams for 60 min using various scheduling algorithms. We consider the SMS and DVBR_τ algorithm, and we vary τ value from 1 to

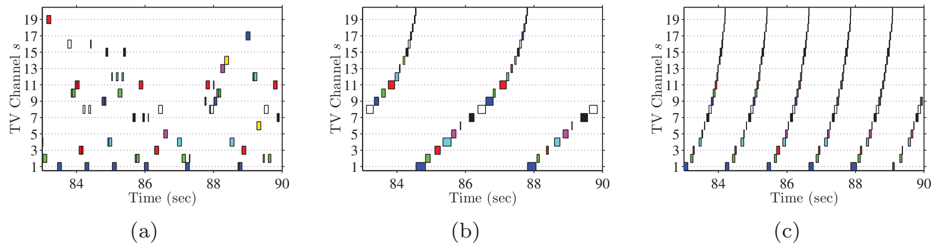


Fig. 4. Burst schedules produced by considered algorithm: (a) SMS, (b) $VBR_{70\%}$, and (c) $RVBR_1$.

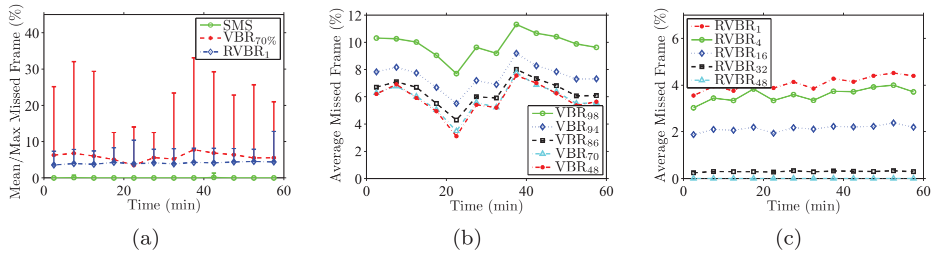


Fig. 5. Missed frame ratio produced by: (a) all considered algorithms, (b) the VBR_α algorithm with various α values, and (c) the $RVBR_\beta$ algorithm with various β values.

2 sec. For all considered algorithms, we compute the energy saving for each channel. We report the mean, maximum, and minimum per-channel energy saving. For the $DVBR_\tau$ algorithm, we also calculate the number of overflow/missed frames.

5.3 Simulation Results

Visual validation. We first plot the burst schedules computed by each considered algorithms in Figure 4 to visually validate their correctness. We zoom into a short period of 7 sec; burst schedules during other time periods are similar. We observe that the bursts scheduled by the SMS algorithm are in variable size and they come in various frequencies. This is because the SMS algorithm quickly adapts to the instantaneous bit-rate variations of VBR streams. In contrast, the current algorithms, both $VBR_{70\%}$ and $RVBR_1$, schedule burst in round-robin fashion. We can draw two observations on the burst schedules computed by the current algorithms. First, they contain slack time, for example, the air medium is idle around the time 86 sec in Figure 4(b). This means the current scheduling algorithms are not optimal in terms of goodput. Second, due to the round-robin nature of current algorithms, video streams with lower bit rates, such as stream 20 in Figure 4(c), have very short bursts, which lead to low energy saving. Therefore, current scheduling algorithms are not optimal in terms of energy saving either.

Missed frames. We compute the mean and maximal missed frame ratios of all video streams in 5-min intervals for the considered algorithm. We report the results in Figure 5(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 may 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.

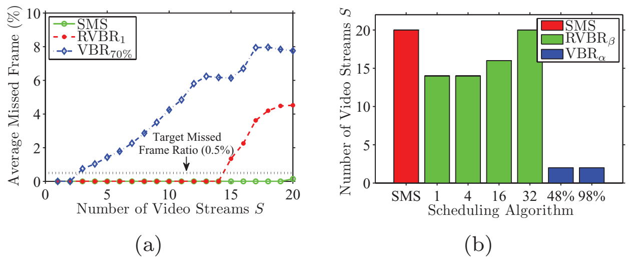


Fig. 6. (a) Missed frame ratio achieved by various scheduling algorithms with different number of video streams. (b) Maximum number of video streams that can be broadcast.

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 Figure 5(a) it produces almost no missed frames. We plot the results of VBR$_\alpha$ algorithm with different α values in Figure 5(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 Figure 5(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 degrades user experience, and thus is not acceptable for mobile video services. Therefore, the current RVBR$_\beta$ algorithm cannot achieve acceptable QoS either. This experiment confirms that the current scheduling algorithms can only achieve inferior perceived quality than the proposed SMS algorithm in open-loop networks.

Number of concurrent video streams. We next study how many video streams can the burst scheduling algorithms concurrently broadcast for a given QoS target: 0.5% missed frame ratio. We iteratively reduce the number of concurrent video streams as outlined in Section 5.2, and we compute the missed frame ratio at each step. We plot the average missed frame ratio throughout the broadcasts in Figure 6(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 Figure 6(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 that the SMS algorithm 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 min. The resulting schedule achieves maximum energy saving by allocating the largest possible bursts that can fit in receiver's buffer. The receiving circuits of mobile devices are put into sleep after getting a burst until that burst is completely consumed. Clearly, the schedule leads

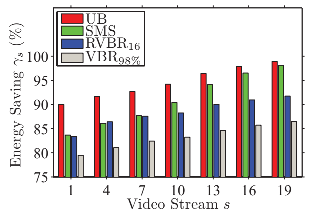
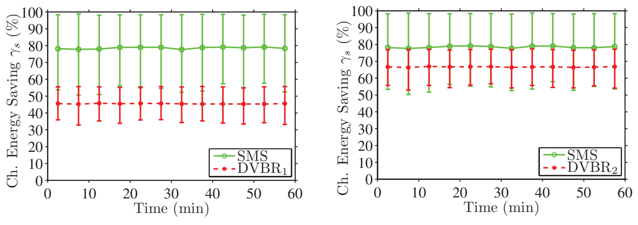


Fig. 7. Energy saving achieved by considered burst scheduling algorithms and a conservative upper bound.



(a) (b)

Fig. 8. Per-channel energy saving comparison between the SMS and DVBR$_\tau$ algorithms, with: (a) $\tau = 1$ and (b) $\tau = 2$.

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 Figure 7; 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 in open-loop networks.

Applicability in closed-loop networks. Next, we compare the performance of the SMS algorithm against the current burst scheduling algorithm in closed-loop networks. We report the mean, maximum, and minimum per-channel energy saving of the SMS and DVBR$_\tau$ algorithm in Figure 8. We draw two observations on this figure. First, the SMS algorithm constantly results in higher energy saving than the current algorithm. For example, Figure 8(a) shows that the SMS algorithm achieves about 80% average energy saving at all time, while the DVBR$_1$ algorithm only achieves about 45%. Second, the DVBR$_\tau$ algorithm achieves higher energy saving when τ increases. For example, Figure 8(b) reveals that the DVBR$_2$ algorithm achieves about 65% energy saving on average, which is better than that of DVBR$_1$. However, higher τ value may result in lost frames due to buffer overflow on mobile devices, which in turn lead to playout glitches, and thus cannot be used in commercial base stations. To understand whether DVBR$_2$ produces a glitch-free burst schedule in the simulation, we plot the number of missed frames in Figure 9. This figure shows that while DVBR$_2$ results in higher energy saving than DVBR$_1$, it also leads to playout glitches. This experiment illustrates that the current DVBR$_\tau$ aalgorithm leads to low energy saving in closed-loop networks, and using our proposed SMS algorithm can improve the average energy saving by about 80%/45% $\approx$ 1.7 times.

Running time. Finally, we report the running time of the SMS algorithm on a commodity PC with a 2.33 GHz processor and runs Linux. It takes the proposed algorithms less than 1 sec to construct

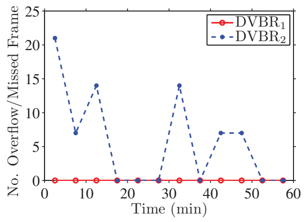


Fig. 9. DVBR$_2$ results in overflow/missed frames on mobile devices.

the burst schedule for the 60-min simulation. This clearly shows that the proposed algorithms incur negligible processing overhead, and can run in real time.

6. CONCLUSIONS

We studied the problem of broadcasting multiple Variable-Bit-Rate (VBR) streams over a broadcast network to many mobile devices. These streams are broadcast in bursts to enable mobile devices to save energy by frequently putting their receiving circuits into sleep. We considered two types of the broadcast networks: closed-loop, in which the aggregate bit rate of all video streams is controlled by a joint video coder and never exceeds the network bandwidth, and open-loop, in which the video streams may occasionally overload the broadcast network since their coding rates are not jointly controlled. We formulated a burst scheduling problem that adopts: (i) goodput 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 the problem. We proved that the SMS algorithm achieves optimal goodput and it provides near-optimal energy saving. Our analysis indicates that a small energy saving gap of at most 1.5% from the optimal is achieved under typical network parameters. We analytically showed that the SMS algorithm produces glitches-free schedules in closed-loop networks, and minimizes the number of glitches in open-loop networks. The SMS algorithm is an online scheduling algorithm with a small look-ahead window, and can handle the dynamic nature of the video broadcast service.

We conducted extensive trace-driven simulations. For open-loop networks, we concurrently broadcast 20 VBR video streams using the SMS algorithm and the scheduling algorithms used in current base stations. The simulation results reveal that the SMS algorithm outperforms the current burst scheduling algorithms in terms of: (i) missed frame ratio, (ii) number of concurrent video streams, and (iii) energy saving. For closed-loop networks, we broadcast 10 jointly coded VBR streams using the SMS algorithm and the algorithm currently used in practice. Our simulation results showed that the SMS algorithm can achieve much higher energy saving: about 1.7 times improvement was observed.

The proposed algorithm for efficiently broadcasting VBR video streams are general and can be employed in different broadcast networks. We achieve this generality by abstracting away the peculiarities of different networks in the formulation of the problem and the proposed scheduling algorithm. To demonstrate the practicality of our proposed algorithm, we have implemented the SMS algorithm 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.

APPENDIXES

A. EVALUATION IN MOBILE TV TESTBED

In this section, we evaluate the SMS algorithm in a real mobile TV network that complies with the DVB-H standard [Faria et al. 2006; Kornfeld and May 2007].

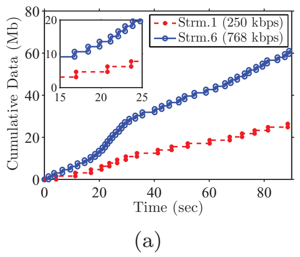
A.1 Testbed Setup

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 [Hefeeda et al. 2008], 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 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. We also use a DVB-H analyzer to gather and analyze the low-level signals.

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 8.289Mbps shared air medium bandwidth [ETSI 2007]. 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 five production-quality video sequences provided by the Canadian Broadcasting Corporation (CBC). CBC 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 min. We encode each video sequence into two H.264/AVC coded VBR streams, with average bit rates of 250 and 768kbps, respectively. That is, we get 10 coded streams in total. We also encode the audio at 96kbps 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 min, 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.

A.2 Results from Mobile TV Testbed

Correctness. We first validate the correctness of the SMS algorithm, that is, it produces burst schedules that adapt to bit-rate variation in VBR streams, and results in no burst conflicts. To show the bit-rate adaptation, we compute the cumulative received bits (from the broadcasting base station) as the time progresses. Sample results are presented in Figure 10(a) for two video streams with different average bit-rates; results for other streams are similar. The figure shows the dynamics of the received bits, and reveals that the SMS algorithm adapts to the bit-rate variations quite well. For example, the bit rate of video stream 6 between 25 and 35 sec is higher than other time periods. Furthermore, we notice the SMS algorithm allocates dynamic inter-burst time to each video stream: bursts are further apart when the instantaneous bit rate is lower, and they are closer otherwise. This is shown by the variable widths of the steps in the staircase lines in the figure. Dynamic inter-burst time allows the SMS algorithm to send bursts as long as possible, which results in high energy saving. Note that this figure shows shorter time period, 90 sec, for the clarity. The results are similar for the whole streaming period.



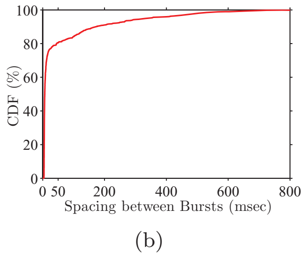


Fig. 10. (a) Buffer dynamics for the SMS algorithm, and (b) time spacing between successive bursts.

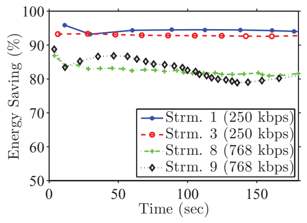


Fig. 11. Energy saving of our algorithm.

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. Note that a negative time spacing indicates bursts intersect with each other. In Figure 10(b), we plot the CDF of the time spacing between two adjacent bursts. This figure clearly shows that there are no conflicts among the resulting bursts.

Energy saving. We report the energy saving achieved by receivers of different video streams when the SMS algorithm is used. Figure 11 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 (250kbps) can be as high as 96%, while it is at least 80% for high bit-rate video streams (768kbps). This figure shows that the SMS algorithm achieves fairly high energy saving in a real testbed.

Running time. In all of the preceding experiments, the SMS algorithm was running in *real time* on a commodity PC. The running time of scheduling bursts for the whole experiment (5 min long) 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.

B. SLOTTED AND DYNAMIC SCHEDULING ALGORITHMS

Slotted scheduling. In slotted scheduling, network operators specify a system-wide interburst time period ΔT sec, and a burst size b_s kb for each video stream s. The multiplexer then schedules a burst every ΔT sec for every stream s, where each burst is b_s kb long. Network operators may directly broadcast VBR streams or add a rate regulator for each VBR stream. We describe both approaches next.

—*VBR (Variable-Bit-Rate):* When choosing a streaming rate r_s for video stream s, network operators face a trade-off between wasted bandwidth and video quality: high r_s may lead to wasting of bandwidth, while low r_s may result in buffer underflow instances. To better quantify this trade-off,

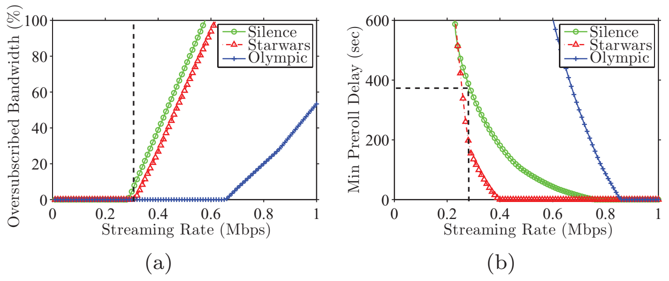


Fig. 12. Regulating VBR streams using buffers may cause: (a) high oversubscribed, wasted bandwidth, and (b) prohibitively long preroll buffering delay.

we compute the CDF (Cumulative Distribution Function) curve $F_s(r)$ of encoding bit-rate for each stream s. We then define a VBR burst scheduling algorithm VBR$_\alpha$ as streaming each video stream s $(1 \leq s \leq S)$ at the smallest bit-rate r_s so that $F_s(r_s) \geq \alpha$. Once the r_s is determined, we compute ΔT and b_s to maximize the energy saving. Specifically, we set $\Delta T = Q/r_S$, where r_S is the highest streaming rate among all videos. We then let $b_s = R\frac{r_s}{\sum_{i=1}^{S} r_i}\Delta T$, where R kbps is the network bandwidth.

—*RVBR (Regulated-Variable-Bit-Rate):* Traffic regulators absorb VBR traffic burstiness at the expense of higher memory requirements, longer preroll delays, and oversubscribed bandwidth. *Preroll delay* is the minimal buffering time to fill the regulator buffer before mobile devices can start getting data out of it without risking for playout glitches. We use the H.264/AVC HRD (Hypothetic Reference Decoder) model [Ribas-Corbera et al. 2003] to compute the oversubscribed bandwidth and minimum preroll delay at various r_s, and we plot the results in Figure 12. This figure indicates that long preroll delay is required if network operators want to avoid wasting of bandwidth. For example, streaming *Silence of the Lambs* at a bit-rate lower than 280kbps leads to no wasted bandwidth, but it results in more than *6 min* preroll delay. To better quantify the trade-off between wasted bandwidth and user experience, we define a burst scheduling algorithm RVBR$_\beta$ as streaming each video stream using a rate regulator. The regulated stream has bit-rate r_s that is the smallest bit rate such that $d_s(r_s) \leq \beta$ sec, where $d_s(r_s)$ represents the minimum preroll delay under streaming rate r_s. Once the r_s is determined, ΔT and b_s can be computed as mentioned.

Dynamic scheduling. The dynamic scheduling requires a joint video coder to work. It allocates each video stream a burst with the size of its aggregate frame size in every ΔT scheduling window. The joint coder does not monitor the buffer states of mobile devices, and network operators must manually choose a proper ΔT value to avoid buffer overflow instances. We describe a general approach to determine ΔT next.

—*DVBR (Dynamic-Variable-Bit-Rate):* One way to avoid overflow instances is to set $\Delta T = Q/R$, which prevents multiplexers from sending any burst longer than Q kb. Doing so, however, may result in too many short bursts and is not efficient in terms of energy saving. Therefore, network operators may increase ΔT for higher energy saving at the expense of potential buffer overflow instances. We define DVBR$_\tau$ as the dynamic scheduling algorithm with a scheduling window size $\Delta T = \tau Q/R$, where $\tau \geq 1$ is a system parameter. Once ΔT is determined, we let the size of each burst be the aggregate frame size in every window.

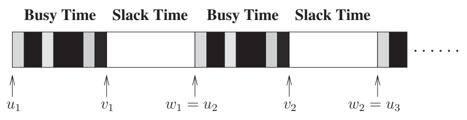


Fig. 13. The resulting schedule of the SMS algorithm consists of interleaved busy and slack time periods. Different shaded blocks indicate bursts of different video streams.

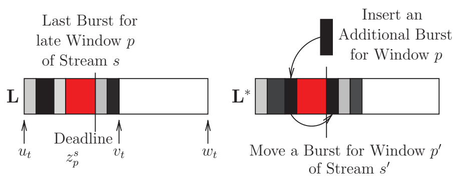

Last Burst for late Window p of Stream s
L
Deadline z_p^s
u_t
v_t
w_t
Insert an Additional Burst for Window p
L*
Move a Burst for Window p' of Stream s'

Fig. 14. Inserting a burst requires moving another burst, as there is no gap between bursts in busy time periods.

C. PROOFS OF THEOREMS

PROOF OF THEOREM 1. The for-loop in lines 7–11 produces a schedule that has no burst intersections. This is because we assign every time interval $[t, t_n)$ to a single stream s in line 8, and we immediately advance t to t_n. Moreover, line 9 guarantees that $y_p^s \geq \sum_{\forall\, x_p^s \leq f_k^s < z_p^s} b_k^s$ holds, because it stops assigning bursts to p if p is complete or late. To show that Eq. (7c) holds, we prove $y_p^s \leq \sum_{\forall\, x_p^s \leq f_k^s < z_p^s} b_k^s$ in the following. Notice that the joint video coder in a closed-loop broadcast network prevents the aggregate bit-rate of all video streams from exceeding the broadcast network bandwidth. Therefore, the multiplexer always has enough air medium time to broadcast all video streams ontime, that is, any burst scheduling algorithm that achieves optimal goodput leads to no late data. Next, we borrow the result from Theorem 2, which states the SMS algorithm maximizes the goodput. This means that the SMS algorithm produces burst schedules with late data, that is, $y_p^s \leq \sum_{\forall\, x_p^s \leq f_k^s < z_p^s} b_k^s$, which yields Eq. (7c). Hence, the SMS algorithm finds a feasible schedule for the transformed problem. Since we divide the receiver's buffer into two halves and we make sure that the aggregate data received in each window equals to half of the receiver's buffer (see Eq. (2)), the resulting schedule leads to no buffer violation instances in the original problem. □

PROOF OF THEOREM 2. Observe that the for-loop starting in line 7 always schedules a burst as long as there is at least one window that is outstanding and is not late. Therefore, the resulting schedule **L** consists of interleaved *busy* time periods and *slack* time periods, as illustrated in Figure 13. 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 **L** falls into one of two cases. Case I: all windows complete in line 9. Case II: there is at least one window late in line 9. 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 **L**. We use proof by contradiction and illustrate the argument in Figure 14. Consider an arbitrary window p of stream s in **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* θ-sec burst to window p, where $\theta > 0$. By definition, goodput 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 9 says that the SMS algorithm always schedules the window with the smallest deadline, thus we know $z^{s'}_{p'} \le z^s_p$. This means that moving the burst for window p' of stream s' after time z^s_p renders it becoming a *late burst*, which cancels out the additional goodput brought by the new burst! Therefore, the amount of ontime delivered bursts in $\mathbf{L}$ and $\mathbf{L}^*$ are the same, which contradicts the assumption. □

PROOF OF THEOREM 3. 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 $n^*_s \ge \sum_{i=1}^{I} l^s_i/Q$. Then, following the definition of energy saving, we write the energy saving of stream s as

$$\gamma^*_s = 1 - \frac{\sum_{k=1}^{n^*_s}\left(T_o + b^s_k/R\right)}{I/F} \le 1 - \frac{T_o\sum_{i=1}^{I} l^s_i/Q + \sum_{i=1}^{I} l^s_i/R}{I/F} = 1 - \left(\frac{T_o}{Q} + \frac{1}{R}\right)r_s,$$

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

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

Next, we let n_s be the number of bursts scheduled for s by the SMS algorithm. Based on Eq. (2), we use $\delta^s_p = \frac{Q}{2} - \sum_{j=m^s_{p-1}+1}^{m^s_p} l^s_j$ to represent a small portion of Q that is not fully utilized in window p. We notice that $\delta^s_p \approxeq 0$, because typical receiver buffers are much larger than frame size, for example, media players buffer for several seconds of playout time, or hundreds of frames, before rendering videos. Since δ^s_p is insignificant, we write $p_s = \sum_{i=1}^{I} l^s_i/(Q/2)$. 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:

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

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

PROOF OF THEOREM 4. 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 8 is $O(PS + S^2)$, where $P = \sum_{s=1}^{S} p_s$. Moreover, constructing windows in lines 5 and 10 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 $O(PS + S^2) + O(PS) = O(PS + S^2)$. □

REFERENCES

ATSC MOBILE DTV STANDARD. 2009. ATSC mobile DTV standard. http://www.openmobilevideo.com/about-mobile-dtv/standards/

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

CAMARDA, P., TOMMASO, G., AND STRICCOLI, D. 2006. A smoothing algorithm for time slicing DVB-H video transmission with bandwidth constraints. In *Proceedings of the ACM International Mobile Multimedia Communications Conference (MobiMedia'06).*

Chari, M., Ling, F., Mantravadi, A., Krishnamoorthi, R., Vijayan, R., Walker, G., and Chandhok, R. 2007. FLO physical layer: An overview. *IEEE Trans. Broadcast. 53,* 1, 145–160.

Chou, P. 2007. Streaming media on demand and live broadcast. In *Multimedia Over IP and Wireless Networks*, M. van der Schaar and P. Chou Eds., Academic Press, Chapter 14, 453–502.

ETSI. 2004. Digital video broadcasting (DVB); transmission system for handheld terminals (DVB-H). European Telecommunications Standards Institute (ETSI) Standard EN 302 304 Ver. 1.1.1.

ETSI. 2007. Digital video broadcasting (DVB); DVB-H implementation guidelines. European Telecommunications Standards Institute (ETSI) Standard EN 102 377 Ver. 1.3.1.

Faria, G., Henriksson, J., Stare, E., and Talmola, P. 2006. DVB-H: Digital broadcast services to handheld devices. *Proc. IEEE 94,* 1, 194–209.

He, Z. and Wu, D. 2008. Linear rate control and optimum statistical multiplexing for H.264 video broadcast. *IEEE Trans. Multimedia 10,* 7, 1237–1249.

Hefeeda, M. and Hsu, C. 2008. Energy optimization in mobile TV broadcast networks. In *Proceedings of the IEEE International Conference on Innovations in Information Technology (Innovations'08)*. 430–434.

Hefeeda, M. and Hsu, C. 2009. On burst transmission scheduling in mobile TV broadcast networks. *IEEE/ACM Trans. Netw. 18*, 2, 610–623.

Hefeeda, M., Hsu, C., and Liu, Y. 2008. Testbed and experiments for mobile TV (DVB-H) networks. In *Proceedings of the ACM Multimedia'08 Demo Session*.

Hsu, C. and Hefeeda, M. 2009a. Broadcasting video streams encoded with arbitrary bit rates in energy-constrained mobile TV networks. *IEEE/ACM Trans. Netw. 18*, 3, 681–694.

Hsu, C. and Hefeeda, M. 2009b. On statistical multiplexing of variable-bit-rate video streams in mobile systems. In *Proceedings of the ACM Multimedia'09*. 411–420.

Hsu, C. and Hefeeda, M. 2009c. Time slicing in mobile TV broadcast networks with arbitrary channel bit rates. In *Proceedings of the Annual Joint Conference of the IEEE Communications Societies (InfoCom'09)*. 2231–2239.

Jacobs, M., Barbarien, J., Tondeur, S., de Walle, R. V., Paridaens, T., and Schelkens, P. 2008. Statistical multiplexing using SVC. In *Proceedings of the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB'08)*. 1–6.

Kornfeld, M. and May, G. 2007. DVB-H and IP Datacast—Broadcast to handheld devices. *IEEE Trans. Broadcast. 53,* 1, 161–170.

Lai, H., Lee, J., and Chen, L. 2005. A monotonic-decreasing rate scheduler for variable-bit-rate video streaming. *IEEE Trans. Circ. Syst. Video Technol. 15,* 2, 221–231.

Lakshman, T., Ortega, A., and Reibman, A. 1998. VBR video: Tradeoffs and potentials. *Proc. IEEE 86,* 5, 952–973.

Lin, J., Chang, R., Ho, J., and Lai, F. 2006. FOS: A funnel-based approach for optimal online traffic smoothing of live video. *IEEE Trans. Multimedia 8,* 5, 996–1004.

Parkvall, S., Englund, E., Lundevall, M., and Torsner, J. 2006. Evolving 3G mobile systems: Broadband and broadcast services in WCDMA. *IEEE Comm. Mag. 44,* 2, 30–36.

Pinedo, M. 2008. *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer.

Rezaei, M., Bouazizi, I., and Gabbouj, M. 2008. Joint video coding and statistical multiplexing for broadcasting over DVB-H channels. *IEEE Trans. Multimedia 10,* 7, 1455–1464.

Rezaei, M., Bouazizi, I., and Gabbouj, M. 2009. Implementing statistical multiplexing in DVB-H. *Int. J. Digital Multimedia Broadcast.*

Ribas-Corbera, J., Chou, P., and Regunathan, S. 2003. A generalized hypothetical reference decoder for H.264/AVC. *IEEE Trans. Circ. Syst. Video Technol. 13,* 7, 674–687.

Seeling, P. and Reisslein, M. 2005. Evaluating multimedia networking mechanisms using video traces. *IEEE Potentials 24,* 4, 21–25.

Seeling, P., Reisslein, M., and Kulapala, B. 2004. Network performance evaluation using frame size and quality traces of single-layer and two-layer video: A tutorial. *IEEE Comm. Surv. Tutor. 6,* 2, 58–78.

Sullivan, G. and Wiegand, T. 1998. Rate-Distortion optimization for video compression. *IEEE Signal Process. Mag. 15,* 6, 74–90.

Tagliasacchi, M., Valenzise, G., and Tubaro, S. 2008. Minimum variance optimal rate allocation for multiplexed H.264/AVC bitstreams. *IEEE Trans. Image Process. 17,* 7, 1057–1143.

UBS. 2009. UBS DVB-H IP encapsulator DVE 6000. `http://www.uniquesys.com/DVB-H-IP-Encapsulator-DVE-6000-SPEC-VER1.2.pdf`

UDCast. 2008. UDCast DVB-H/DVB-SH encapsulator (IPE-10). `http://www.udcast.com/products/downloads/DVB-H_DVB-SH_IPE.pdf`

WANG, F., GHOSH, A., SANKARAN, C., FLEMING, P., HSIEH, F., AND BENES, S. 2008. Mobile WiMAX systems: Performance and evolution. *IEEE Comm. Mag. 46,* 10, 41–49.

WANG, L. AND VINCENT, A. 1996. Joint rate control for multi-program video coding. *IEEE Trans. Consumer Electron. 42,* 3, 300–305.

YANG, X., SONG, Y., OWENS, T., COSMAS, J., AND ITAGAKI, T. 2004. Performance analysis of time slicing in DVB-H. In *Proceedings of the Joint IST Workshop on Mobile Future and Symposium on Trends in Communications (SympoTIC'04)*. 183–186.

Received October 2009; accepted December 2009